ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ 1: ΣΧΕΔΙΑΣΗ ΕΠΕΡΓΑΣΤΗ ΕΝΟΣ ΚΥΚΛΟΥ

ΦΑΣΗ 1

Σχεδιάστηκε μία μονάδα αριθμητικών και λογικών πράξεων (ALU) η οποία είναι ένα ασύγχρονο συνδυαστικό κύκλωμα το οποίο κάνει πράξεις ανάμεσα με 2 32bit εισόδους (A,B) ποια πράξη θα κάνει καθορίζεται απο την 4bit είσοδο Op το αποτέλεσμα θα έχει επίσης καθυστέρηση 10ns. Οι εξόδου αποτελούνται απο το Out το 32bit αποτέλεσμα της πράξης και τα Zero, Cout , Ovf bits τα οποία λαμβάνουν την τιμή '1' όταν αντιστοίχως το Out είναι 0 , υπάρχει carry out στην πρόσθεση αριθμών και τέλος όταν υπάρχει overflow στην πρόσθεση η αφαίρεση αριθμών.

Σχεδιάστηκε ένας σύγχρονος καταχωρητής 32bit με reset και write enable τα οποία αντιστοίχως μηδενίζουν την έξοδο dataout και επιτρέπουν να γραφούν δεδομένα στον register. Στην συνέχεια σχεδιάστηκε ένας Decoder 32 bits ο οποίος ανάλογα με το σήμα εισόδου 5bit "επιλέγει" μία από τις 32 εξόδους του θέτοντας το ανάλογο bit μέσα στην 32bit έξοδο ίσο με '1' ενώ τα άλλα ίσα με 0. Ακόμη σχεδιάστηκε ένας multiplexer 32bit ο οποίος ανάλογα με το 5bit σήμα εισόδου "επιλέγει" ως έξοδο μία εκ των 32x32bit είσοδο.Ο mux σχεδιάστηκε με την βοήθεια package το οποίο επιτρέπει να υπάρχει ως είσοδος ένα array 32 θέσεων με 32bit έκαστη. Εν τέλει ενώθηκαν τα παραπάνω components(32 registers, 2mux, 1 decoder) για τον σχεδιασμό του Register File ενός module το οποίο επιτρέπει την γραφή δεδομένων 32bit Din εφόσον το WrEn ισούται με '1' σε έναν από τους 31 καταχωρητές (καθώς ο καταχωρητής 0 είναι πάντα 0) αναλόγως με το Awr 5bit ενώ μπορεί να διαβάσει από 2 καταχωρητές ταυτόχρονα (Dout1 και Dout2 32bit) ανάλογα με τα Ard1 Ard2 5bit αλλά και να μηδενίσει όλους του καταχωρητές με την ενεργοποίηση του Rst. Αξίζει να σημειωθεί ότι ενώ η γραφή είναι σύγχρονη η ανάγνωση είναι ασύγχρονη και ότι κατά την δημιουργία του χρησιμοποιήθηκε for generate για την δημιουργία πολλαπλών registers.

ΦΑΣΗ 2

Μελετήθηκε η μνήμη RAM 2048x32bit που δίνεται έτοιμη και συντάχθηκε testbench για καλύτερη κατανόηση της λειτουργίας της.

Σχεδιάστηκαν 2 αθροιστές ένας που αθροίζει +4 στην 32bit είσοδο του και ένας που αθροίζει τις 2 32 bit εισόδους του σχεδιάστηκε ακόμη ένας multiplexer 2x32 bit εισόδων όμοιας λειτουργίας με αυτόν 32x32 εισόδων (δημιουργία νέου package) και ενώθηκαν με έναν register συμπληρώνοντας έτσι το IFSTAGE. Το IFSTAGE είναι το module το οποίο καθορίζει την τιμή που θα λάβει ο ProgramCounter (PC) Pc+4 η PC+4+immed. Αναλόγως με την τιμή του PC κάνει fetch και τις ανάλογες εντολές στην μνήμη.

Σχεδιάστηκε ένας mux 2x5 bit εισόδων όμοιας λειτουργίας με τους προηγούμενους 2 (δημιουργία νέου package) και ένα module ImmedSelect το οποίο αναλόγως με ένα σήμα εισόδου 2 bit κάνει την ανάλογη πράξη zerofill, sign extension , left shift zerofill , left shift sign extension στην είσοδο 16bit. Με την ένωση των παραπάνω components και ενός mux 2x32 bit εισόδων και ενός register file δημιουργήθηκε το DECSTAGE ενα module που αποκωδικοποιεί τις εντολές που έγιναν fetched απο το IFSTAGE. Ο mux 32x2 bit εισόδων επιλέγει ως έξοδο τα δεδομένα της ALU η της μνήμης αναλόγως με το RF_WrData_sel ο άλλος mux επιλέγει ποιο μέρος της εντολής θα πάει στο Ard2 του rf ενώ ο Immedselect μετατρέπει ένα άλλο μέρος της εντολής.

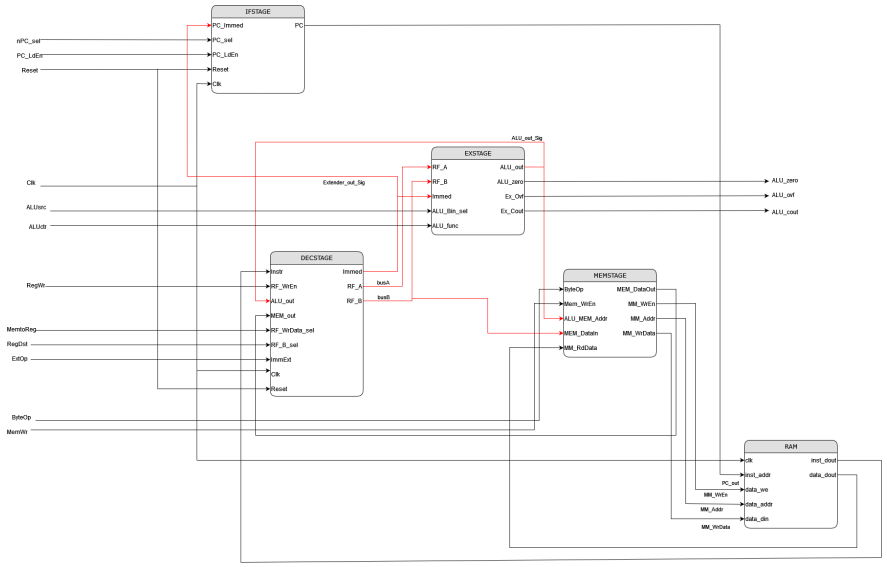
Συνδέθηκε η ALU με εναν mux 2x32 bit εισόδου και έτσι δημιουργήθηκε το EXSTAGE, ένα module το οποίο είναι υπεύθυνο για την εκτέλεση των εντολών που έχουν αποκωδικοποιηθεί από το DECSTAGE. Ο mux επιλέγει ανάμεσα στο immediate και έναν RF_B register ενώ η ALU δέχεται ως εισόδους την έξοδο του mux και μια έξοδο register RF_A.

Τέλος δημιουργήθηκε το MEMSTAGE ένα module το οποίο αλληλεπιδρά με την μνήμη κάνοντας load και store πληροφορία. Στην διεύθυνση μνήμης θα προστίθεται πάντα η τιμή 256 καθώς πριν από το 256 είναι χώρος μνήμης για εντολές ενώ αφαιρούνται τα τελευταία 2bit ως δείκτες θέσης μέσα στην διεύθυνση μνήμης. Μια θέση μνήμης αποτελείται από 32bit το sb αποθηκεύει ένα byte (8bit) στα τελευταία 8 bit της διεύθυνσης ομοίως το lb φορτώνει από τα τελευταία 8bit της διεύθυνσης τα δεδομένα σε έναν καταχωρητή. Αξίζει να σημειωθεί ότι ενώ για όλα τα υπόλοιπα modules το testbench δημιουργήθηκε βάση το ίδιο το module στο memstage δημιουργήθηκε ένα MEMTestingModule το οποίο συνέδεε το MEMSTAGE με την μνήμη και το testbench βασίστηκε πάνω σε αυτό.

ΦΑΣΗ 3

Συνενώθηκαν τα IFSTAGE, DECSTAGE, EXSTAGE, MEMSTAGE για την δημιουργία του module CONTROL και στο testbench του συνενώθηκε με την μνήμη για τον έλεγχο ορθής λειτουργίας όλων των components ως σύστημα. Οι συνδέσεις μεταξύ των compo-

nents έγινε όπως φαίνεται παρακάτω.



DATAPATH και RAM

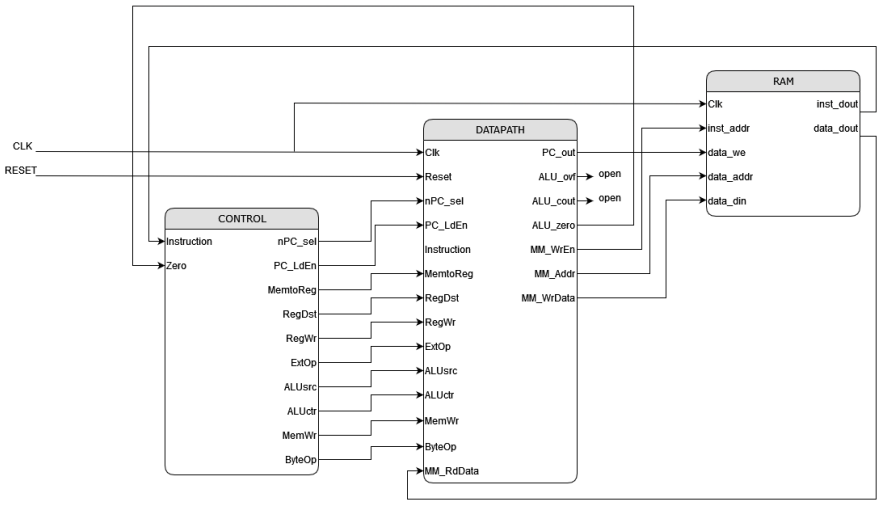
Σχόλια: Το datapath δεν εμπεριέχει την RAM στο σχήμα τα σήματα με μαύρο χρώμα θεωρούνται οι είσοδοι και έξοδοι του datapath ενώ με κόκκινο σημειώνοντα τα εσωτερικά σήματα μεταξύ component του datapath. Ακόμη για όλα τα component και για την RAM ισχύει ότι οι είσοδοι έχουν σημειωθεί στην δεξιά πλευρά του αντίστοιχου πλαισίου και οι έξοδοι αριστερά.

Σχεδιάστηκε στην συνεχεία το control το οποίο είναι το module που ελέγχει το datapath λαμβάνοντας 32bit Instructions απο την μνήμη και παράγει τα εξής σήματα $PC\_Sel, ALU\_Bin\_sel, PC\_LdEn, RF\_WrData\_sel, RF\_B\_Sel, RF\_WrEn, ALU\_func, MemWr, ByteOp, ImmExt$.

- Το PC_Sel παίρνει την τιμή 1 δηλαδή $4+Pc\_Immed$ μόνο όταν έρχονται εντολές όπου απαιτείται branch δηλαδή b και μόνο όταν ισχύει η συνθήκη του branch στις εντολές bne,beq.
- Το ALU_Bin_sel παίρνει την τιμή 0 μόνο στις εντολές που έχουν func και στις bne και beq καθώς ο έλεγχος για το αν γίνεται branch στις εντολές bne, beq γίνεται με την αφαίρεση των καταχωρητών που πρέπει να συγκριθούν και τον έλεγχο του Zero flag γιαυτό επιλέγεται 0 δηλαδή επιλογή του RF_B ως μια τιμή εισόδου της ALU
- Το PC_LdEn ισούται πάντα με 1.
- Το RF_WrData_sel ισούται με 1 μόνο στις περιπτώσεις lb και lw καθώς μονο τότε έρχονται δεδομένα από την μνήμη.

- Το RF_B_Sel ισούται με 0 μόνο για τις εντολές που έχουν εκτελούνται στην ALU λαμβάνοντας έτσι τον rd για διάβασμα και γράψιμο στο RF.
- Το RF_WrEn ισούται με 0 μόνο για τις εντολές b, beq, bne, sb, sw καθώς αυτές είναι οι μόνες εντολές για τις οποίες δεν χρειάζεται να γραφεί κάτι στον RF.
- Το ALU_func ισούται με "0001" στις περιπτώσεις εντολών bne και beq γιατί όπως εξηγήθηκε και παραπάνω για τον έλεγχο συνθήκης branch απαιτείται αφαίρεση μεταξύ των καταχωρητών, στις περιπτώσεις εντολών της ALU ισούται με τα 4 πρώτα bit του func καθώς είναι τα ίδια με τους κωδικούς εντολών στην ALU. Τέλος στις εντολές addi, nandi, ori απαιτείται η ALU να κάνει add nand και or αντιστοίχως οπότε το ALU_func λαμβάνει τις ανάλογες εντολές
- Το MemWr λαμβάνει τις τιμές 1 μόνος στις περιπτώσεις του sb και lb καθώς μόνο τότε απαιτείτε γραφή στην μνήμη.
- Το ByteOp λαμβάνει τιμές 0 για lw,sw και 1 για sb,lb
- Το ImmExt λαμβάνει τιμές "00" δηλαδή zero fill για nandi, ori "01" δηλαδή sign extension για εντολές li, addi, lw, sw ,lb ,sb ενω "10" δηλαδή zero fill after 16 shift to left για lui και "11" η sign extension and 2 shift to left για b, beq, bne. Η απόφαση για το ποία εντολή θα λάβει τι ImmExt έγινε σύμφωνα με την πράξη που γίνεται στο Imm στον πίνακα εντολών σελίδα 9 στην εκφώνηση.

Τέλος με την συνένωση του DATAPATH, CONTROL και μνήμης σύμφωνα με τον παρακάτω σχήμα δημιουργήθηκε το $PROC\_SC$ το ολοκληρωμένο module της πρώτης εργασίας.



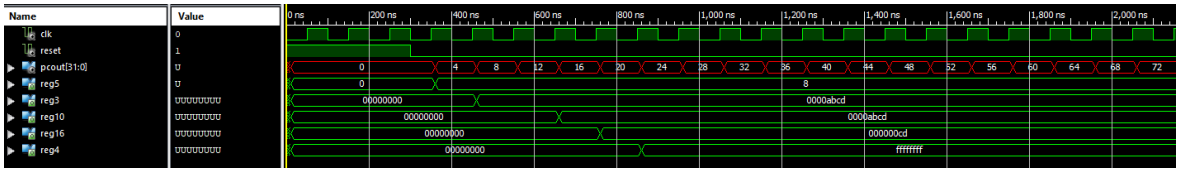
PROC_SC

Ισχύει όπως και στο παραπάνω σχήμα ότι οι είσοδοι έχουν σημειωθεί στην δεξιά πλευρά του αντίστοιχου πλαισίου και οι έξοδοι αριστερά ενώ όλα είναι εσωτερικά σήματα εκτός του CLK και RESET τα οποία είναι είσοδοι και τα ALU_ovf και ALU_cout τα οποία είναι open.

TESTING

Πραγματοποιήθηκαν 3 προγράμματα 2 τα οποία δινόταν και 1 το οποίο φτιάχθηκε ξεχωριστά όλα εξετάστηκαν στον $PROC\_SC$

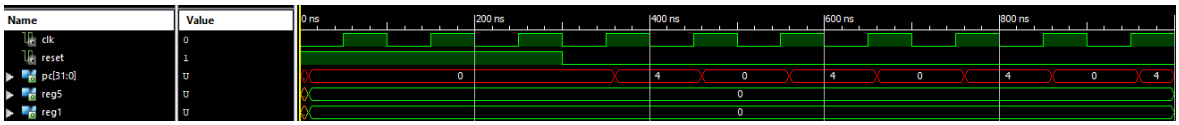
Πρόγραμμα αναφοράς 1



Πρόγραμμα 1 , PC με χρώμα κόκκινο , αρχείο rom3.data

Το reset είναι ενεργό για 3 κύκλους = 300ns 12ns μετά από ακμή ρολογιού (350ns) εκτελείται η πρώτη εντολή addi r5, r0, 8 και ο reg5 παίρνει τιμή 8, 12ns μετά από ακμή ρολογιού (450ns) εκτελείται η δεύτερη εντολή ori r3, r0, 0xABCD και ο reg3 παίρνει τιμή 0xABCD. Στην ακμή (550ns) εκτελείται η sw r3, 4(r0) όπου αποθηκεύεται στην θέση μνήμης 257 η 0xABCD ενώ 10ns μετά την ακμή (650ns) εκτελείται η lw r10, -4(r5) και φορτώνεται από την θέση μνήμης 257 η 0xABCD στον reg10 μετά 10ns μετά την ακμή (750ns) εκτελείται η lb r16, 4(r0) όπου φορτώνεται στον reg16 το πρώτο byte της διεύθυνσης μνήμης 257 δηλαδή το 0xCD. Τέλος 10ns μετά την ακμή εκτελείται η εντολή nand r4, r10, r16 όπου το r4 λαμβάνει την τιμή 0xFFFFFFFF παρατηρείται εν τέλει ότι ο pc register αυξάνεται κανονικά με +4 σε κάθε εντολή αφού δεν υπάρχει branch

Πρόγραμμα αναφοράς 2

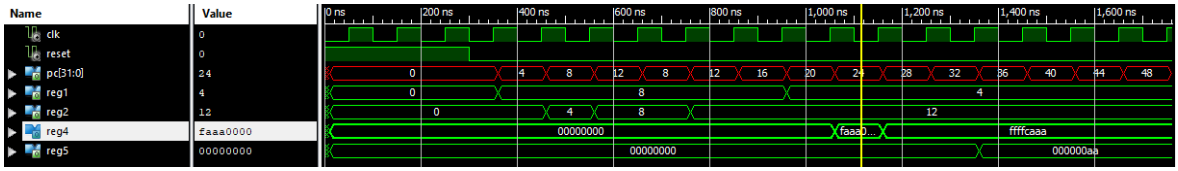


Πρόγραμμα 2 , PC με χρώμα κόκκινο, αρχείο romp1.data

Το reset είναι ενεργό για 3 κύκλους = 300ns 10ns μετά από ακμή (350ns) εκτελείται η bne r5, r5, 8 η οποία αποτυγχάνει αφού προφανώς r5=r5 έτσι ο pc την προσπερνά αυξάνοντας τιμή κατά 4 μετά 10ns μετά από ακμή (450ns) εκτελείται η b -2 η οποία πάει στην προηγούμενη εντολή και ο pc λαμβάνει πάλι την τιμή 0 έτσι δημιουργείται ενα loop

το οποίο δεν τελειώνει ποτέ αφού βλέπουμε εν τέλει ότι δεν εκτελείται η addi r1, r0, 1 καθώς το r1 έχει τιμή 0.

Πρόγραμμα αναφοράς 3



Πρόγραμμα 3 , PC με χρώμα κόκκινο, αρχείο rom10.data

Το reset είναι ενεργό για 3 κύκλους = 300ns 10ns μετά από ακμή (350ns) εκτελείται η addi r1, r0, 8 και ο reg1 παίρνει τιμή 8 10ns μετά από ακμή ρολογιού (450ns) εκτελείται η δεύτερη εντολή ori r2 ,r0, 4 όπου ο reg2 παίρνει την τιμή 4 , 10ns μετά από ακμή (550ns) εκτελείτε η εντολή addi r2 ,r2, 4 και ο reg2 παίρνει την τιμή 8 , 10ns μετά από ακμή (650ns) εκτελείτε η εντολή beq r1, r2 , -2 και αφού r1=r2 ο pc που έως τώρα αυξανόταν +4 για κάθε εντολή πλέον παίρνει την τιμή 8 από την παλιά 12 και ξανατρέχει η εντολή addi r2 ,r2, 4 10ns μετά από ακμή (750ns) όπου ο reg2 γίνεται 12. Πλέον δεν ισχύει r1=r2 οπότε δεν υπάρχει branch και ο pc από εδώ και πέρα αυξάνεται σταθερά κατά 4 έτσι γίνονται εν συνεχεία με καθυστέρηση 10ns εκάστη οι εντολές ror r1, r1 , lui r4, r0, 0xFAAA , addi r4, r0 ,0xCAAA με αποτέλεσμα οι καταχωρητές να έχουν τις τιμές ως εξής: reg1=4 , reg2=12 , reg4=0xFFFFCAAA. Στην συνέχεια (1250ns) αποθηκεύετε στην μνήμη στην διεύθυνση 257 κατά ακμή χωρίς καθυστέρηση το byte 0xAA από την sb r4 ,4(r0) και μετά από 10ns στην ακμή ο reg5 να παίρνει την τιμή 0xAA από την μνήμη από την εντολή lb r5 ,4(r0)

ΣΗΜΕΙΩΣΕΙΣ

Ο επεξεργαστής φαίνεται να λειτουργεί χωρίς πρόβλημα, μερικά testbench δημιουργήθηκαν από το doulos.com