# Οργάνωση Υπολογιστών – ΗΡΥ312
# Αναφορά 1ής εργασίας - Δημιουργία Επεξεργαστή ενός κύκλου

Ιάσονας Γεωργακάς
2017030021

Παναγιώτης Βασιλείου
2017030067

LAB31243750
Απρίλιος, 2020
Πολυτεχνείο Κρήτης

## 1 Εισαγωγή

Στην παρούσα εργασία σκοπός ήταν η υλοποίηση και επαλήθευση της λειτουργίας ενός επεξεργαστή ενός κύκλου χρησιμοποιώντας τη γλώσσα περιγραφής υλικού VHDL, εφαρμόζοντας τεχνική σχεδίασης bottom-up.

## 2 Τεχνολογία

Για την υλοποίηση της εργασίας χρησιμοποιήθηκε η σουίτα λογισμικού Xilinx ISE 13.7 και πιο συγκεκριμένα τα εργαλεία XST για σύνθεση και ISim για προσομοίωση. Η γλώσσα περιγραφής υλικού (HDL) πάνω στην οποία αναπτύχθηκε ο επεξεργαστής ήταν η VHDL-93.

## 3 Υλοποίηση

Η ανάπτυξη του επεξεργαστή έγινε σε τρία στάδια. Στο πρώτο στάδιο, υλοποιήσαμε την αριθμητική-λογική μονάδα (ALU), η οποία εκτελεί τις κύριες πράξεις στον επεξεργαστή, δηλαδή αριθμητικές και λογικές πράξεις. Συγκεκριμένα έχουν υλοποιηθεί οι εξής:

- Πρόσθεση
- Αφαίρεση
- And
- Or

- Not
- Nand
- Nor
- Shift right arithmetic by 1 bit
- Shift right logical by 1 bit
- Shift left logical by 1 bit
- Rotate left by 1 bit
- Rotate right by 1 bit

Η ALU έχει δύο 32-bit εισόδους A, B (οι τελεστέοι σε συμπλήρωμα ως προς δύο) και μία είσοδο 4-bit *Op* (ο κωδικός πράξης) η οποία προσδιορίζει την πράξη που εκτελείται και παράγει 4 εξόδους. Τα σήματα εξόδου της ALU είναι το αποτέλεσμα της εκάστοτε πράξης *Result* και τρία σήματα *Ovf*, *Zero*, *Cout* εκ των οποίων τα *Ovf* και *Zero* λειτουργούν ως flags. Το σήμα *Ovf* υποδεικνύει φαινόμενα υπερχείλισης στις πράξεις της πρόσθεσης και της αφαίρεσης, ενώ το σήμα *Zero* ενεργοποιείται όταν το αποτέλεσμα οποιασδήποτε πράξης είναι 0. Το *Cout* χρησιμοποιείται ως MSB για την απεικόνιση της σωστής πληροφορίας σε περιπτώσεις υπερχείλισης στην πρόσθεση ή στην αφαίρεση.

Στη συνέχεια δημιουργήσαμε 32 καταχωρητές των 32-bit. Οι καταχωρητές διαθέτουν τέσσερις εισόδους *d*, *WE*, *RST*, *CLK* και μία έξοδο *q*. Η έξοδος παίρνει την τιμή της εισόδου εφόσον ενεργοποιηθεί το σήμα ελέγχου εγγραφής *WE* (active high), το σήμα *RST* είναι ανενεργό (active low) και το *CLK* έχει rising edge. Ο καταχωρητής 0 είναι διαφορετικός από τους υπόλοιπους, γιατί έχει έξοδο πάντα την τιμή 0. Χρησιμοποιώντας ως building block τους 32-bit καταχωρητές, υλοποιήσαμε το αρχείο καταχωρητών (Register File). Το αρχείο καταχωρητών έχει δυνατότητα ανάγνωσης δύο καταχωρητών και εγγραφής ενός καταχωρητή προσδιορίζοντας τις επιθυμητές διευθύνσεις ανάγνωσης και εγγραφής. Για την εγγραφή στον καταχωρητή εγγραφής απαιτείται ο σωστός προσδιορισμός καταχωρητή εγγραφής και η ενεργοποίηση του ελέγχου εγγραφής *WE*. Το *RST* του Register File είναι βραυκυκλωμένο με τα *RST* όλων των καταχωρητών και επομένως η ενεργοποίησή του μηδενίζει τις τιμές όλων των καταχωρητών (εκτός του R0 που έιναι by default στο 0).

Στη συνέχεια πραγματοποίηθηκε ένα μικρό μέρος της σχεδίασης και η υλοποίηση των βασικών βαθμιδών (stages) του Datapath, σύμφωνα με τις προδιαγραφές της Αρχιτεκτονικής Συνόλου Εντολών (ISA) που μας δόθηκε, το οποίο υποστηρίζει 2 format εντολών. Το 1ο format υποστηρίζει τις R-type εντολές και έχει περιθώριο για επέκταση των εντολών shift, αφού/ μιας και το πεδίο Instruction[10:6] (γνωστό και ως shamt στο MIPS) δεν χρησιμοποιείται. Το 2ο format εντολών υποστηρίζει I-type αριθμητικές-λογικές εντολές, εντολές μνήμης και εντολές διακλάδωσης. Ο επεξεργαστής είναι αρχιτεκτονικής load/store και οι εντολές μνήμης

που υποστηρίζει είναι lw, sw, lb, sb. Οι εντολές διακλάδωσης είναι οι *branch if equal*, *branch if NOT equal* και *branch*, οι οποίες είναι PC-relative. Οι βαθμίδες του Datapath είναι η βαθμίδα ανάκλησης εντολών (Instruction Fetch), η βαθμίδα αποκωδικοποιήσης των εντολών (Decode), η βαθμίδα εκτέλεσης εντολών (Execution) και η βαθμίδα πρόσβασης μνήμης (Memory) οι οποίες εξυπηρετούν την εκτέλεση του συνόλου εντολών.

Το Instruction Fetch είναι το υποκύκλωμα που υπολογίζει τη διεύθυνση επόμενης εντολής. Σε αυτό περιέχεται ο 32-bit καταχωρητής PC (ο οποίος χρησιμοποιήθηκε και στο αρχείο καταχωρητών) και δύο 32-bit προσθετές. Ανάλογα με το σήμα ελέγχου της βαθμίδας επιλέγεται η αμέσως επόμενη διεύθυνση (PC + 4) ή το άθροισμα της αμέσως επόμενης διεύθυνσης με την είσοδο Immediate (PC + 4 + Immed).

Η βαθμίδα αποκωδικοποίησης (Decode stage) πραγματοποιεί την εγγραφή των δεδομένων στον κατάλληλο καταχωρητή, την επέκταση του Immediate με βάση το σήμα ελέγχου που λαμβάνει καθώς και την ανάγνωση των επιθυμητών καταχωρητών. Σε αυτήν περιέχεται το αρχείο καταχωρητών (RF), 3 πολυπλέκτες (MUXs) για την επιλογή καταχωρητών ανάγνωσης και εγγραφής και ένα υποκυλωμα που εκτελεί 4 είδη επέκτασης (Sign extension, Zerofilling, Sign extension and sll 2, sll 16). Οι καταχωρητές από τους οποίους λαμβάνονται τα δεδομένα επιλέγονται με τη χρήση δύο σημάτων ελέγχου (*RF_A_sel*, *RF_B_sel*). Συγκεκριμένα η τιμή στο σήμα *RF_A_sel* ανατίθεται 1 μόνο κατά την εκτέλεση των εντολών li και lui, ενώ έτσι ώστε η είσοδος RF_A της ALU να έχει τιμή 0 και με την χρήση της πράξης or μεταξύ του RF_A με το Immediate να φορτώσουμε το Immediate στον καταχωρητή προορισμού. Το σήμα *RF_B_sel* θέτεται ίσο με 1 στις εντολές sb, sw, beq και bne οι οποίες χρησιμοποιούν ως δεύτερο καταχωρητή ανάγνωσης τον καταχωρητή RF[rd]. Το σήμα *RF_Wr_Data_sel* θέτεται ίσο με 1 όταν θέλουμε να γράψουμε δεδομένα από τη μνήμη και 0 όταν τα δεδομένα έρχονται από την έξοδο της ALU. Το block diagram του Decode stage παρατίθεται στο Figure 1.

Η βαθμίδα εκτέλεσης εντολών (Execution stage) εκτελεί τις αριθμητικές και λογικές πράξεις και περιέχει την ALU και έναν πολυπλέκτη για την επιλογή μεταξύ καταχωρητή και Immediate ως δεύτερη είσοδο της ALU. Οι έξοδοι του Execution stage είναι το αποτέλεσμα της ALU και το flag ALU_zero.

Η βαθμίδα πρόσβασης μνήμης (Memory stage) λειτουργεί ως διεπαφή της *Ram* και του επεξεργαστή ώστε να πραγματοποιείται η μεταφορά δεδομένων. Ο ρόλος της βαθμίδας είναι μετατροπή της διεύθυνσης που έχει υπολογίσει η ALU στη σωστή διέυθυνση της *Ram*. Συγκεκριμένα, γίνεται μετατροπή της δίευθυνσης προσθέτοντας τη σταθερά $+400_{\text{hex}}$ για την ανάγνωση και εγγραφή δεδομένων μόνο από και προς το data segment. Επιπλέον, χρησιμοποιείται ένα σήμα ελέγχου για την επιλογή της ποσότητας μεταφοράς δεδόμενων ($ByteOp = 1$ για μεταφορά byte και $ByteOp = 0$ για μεταφορά λέξης).

Ο κώδικας υλοποίησης της *Ram* ήταν δοσμένος από το εργαστηριακό προσω-

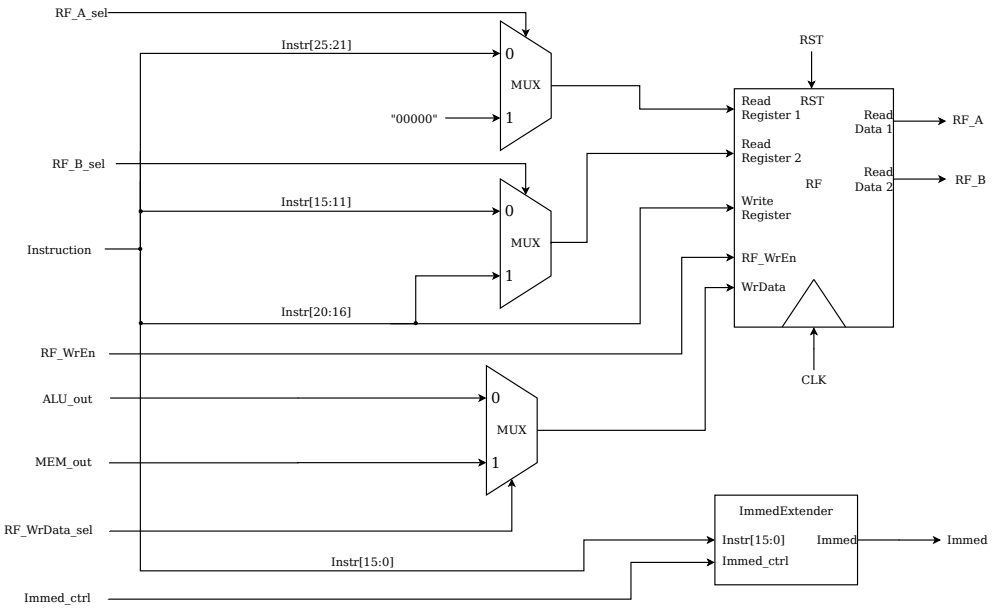


Figure 1: Decode stage

πικό και πραγματοποίησαμε την διασύνδεση με τα υπόλοιπα modules. Η Ram είναι 2048 θέσεων λέξεων και διαθέτει δύο ports για ανάγνωση και ένα port για εγγραφή. Η μνήμη χωρίζεται στο text segment (διευθύνσεις λέξεων 0 έως 255) και στο data semgment (διευθύνσεις λέξεων 256 έως 2047). Για την εγγραφή στο text segment γράφουμε τις τιμές που θέλουμε να φορτώσουμε σε ένα αρχείο και προσδιορίζουμε το αρχείο αυτό στον κώδικα της Ram. Για την ανάγνωση των εντολών από τη RAM απαιτείται η κατάλληλη διεύθυνση (στην είσοδο inst_addr) η οποία θα εμφανίσει το instruction της εντολής στην έξοδο inst_dout. Για την εγγραφγή στο data segemnt πρέπει να ενεργοποιηθεί το σήμα ελέγχου *Data_WrEn* (active high) και να προσδιοριστούν τα δεδομένα και η διεύθυνση εγγραφής. Για την ανάγνωση από το data segment πρέπει να διαβαστούν τα δεδομένα από την κατάλληλη διεύθυνση.

Στο υποκύκλωμα Datapath πραγματοποιείται η διασύνδεση των τεσσάρων βαθμίδων που αναφέρθηκαν παραπάνω. Δέχεται ως είσοδο σήματα ελέγχου από το Control και τις εξόδους δεδομένων της Ram. Στο Figure 2 παρατίθεται το block diagram του Datapath. Στην αριστερή πλευρά εμφανίζονται οι είσοδοι και στη δεξιά πλευρά οι έξοδοι.

Η μονάδα ελέγχου (Control Unit) αποτελείται από δύο ξεχωριστά κυκλώματα, τα Control και ALU Control. Το Control δέχεται ως είσοδο το Opcode της τρέχουσας εντολής και παράγει σήματα ελέγχου τα οποία καταλήγουν στο Datapath, εκτός του *ALU_Op* το οποίο το δέχεται ως είσοδο το ALU Control. Το

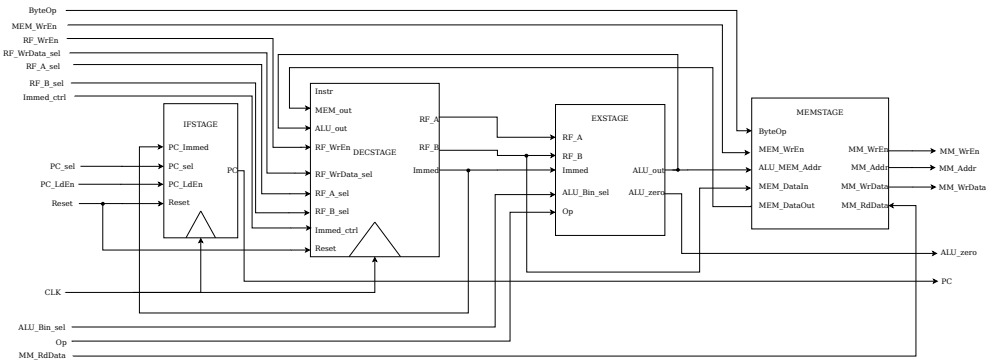


Figure 2: Datapath block diagram

ALU Control έχει ως δεύτερη είσοδο το func πεδίο της εντολής (Instruction[5:0]) και η έξοδός του δίνεται ως είσοδος στο Datapath. Επιπλέον, τα Control σήματα *Branch*, *BranchNotEq*, *Jump* και *ALU_zero* χρησιμοποιούν επιπλέον λογική (1 OR gate, 2 AND gates) για την παραγωγή του σήματος *PC_sel*. Ο επεξεργαστής (top level) έχει ένα RST το οποίο καταλήγει στα ακολουθιακά κυκλώματα του Datapath και ένα ρολόι (CLK) για τον συγχρονισμό Datapath και Ram. Το block diagram του top level παρατίθεται στο Figure 3.

# 4 Αποτελέσματα

Στο πρόγραμμα αναφοράς 1 εκτελέστηκαν μερικές αριθμητικές-λογικές εντολές και εντολές μνήμης. Στον 1ο κύκλο (0-100 ns) είχε ενεργοποιηθεί το *Reset* με αποτέλεσμα να μην αυξηθεί η τιμή του PC μέχρι τη χρονική στιγμή 160 ns, το οποίο είναι το άθροισμα της θετικής ακμής του ρολογιού (150 ns) και η χρονοκαθυστέρηση που προσδίδει το after (περίπου 10ns). Παρατηρούμε, ότι τόσο στους καταχωρητές όσο και στη μνήμη εκχωρούνται οι σωστές τιμές, με άνάλογη καθυστέρηση. Το simulation του επεξεργαστή για το πρόγραμμα αναφοράς 1 παρατίθεται στο Figure 4.

Στο πρόγραμμα αναφοράς 2 εκτελέστηκε ένα πρόγραμμα με infinite loop. Αυτό είναι εμφανές από την εναλλαγή της τιμής του PC, από 0 σε 4 και αντίστροφα. Ως αποτέλεσμα, η 3η εντολή του προγράμματος δεν εκτελείται ποτε και δεν γράφεται κάποια στον R1. Το simulation του επεξεργαστή για το πρόγραμμα αναφοράς 2 παρατίθεται στο Figure 5.

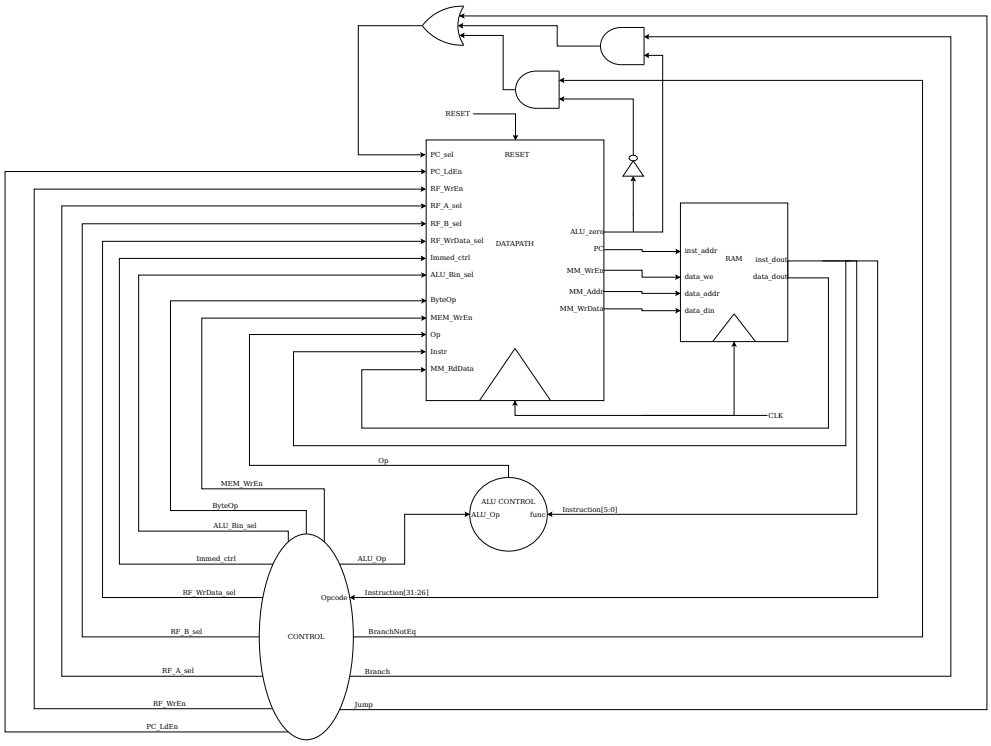


Figure 3: PROC SC - Top Level Diagram

# 5 Συμπέρασμα

Με την ολοκλήρωση της πρώτης εργασίας έγινε κατανοητή η σημασία του testing και verification ενός κυκλώματος με τη χρήση πολλαπλών testbench για κάθε υποκύκλωμα σε ένα project μεγάλης κλίμακας. Επίσης, παρατηρήθηκε ότι είναι αναγακίο το testbench να βασίζεται σε χαρακτηριστικές περιπτώσεις (συνήθεις και ακραίες περιπτώσεις). Για την προσομοίωση των χρονοκαθυστερήσεων που προσδίδει το υλικό, χρησιμοποιήθηκε η εντολή after. Κατανοήσαμε ότι οι χρονοκαθυστερήσεις σε ένα κύκλωμα διαμορφώνουν το critical path και επομένως επηρεάζουν την περίοδο ρολογιού που χρησιμοποιείται.

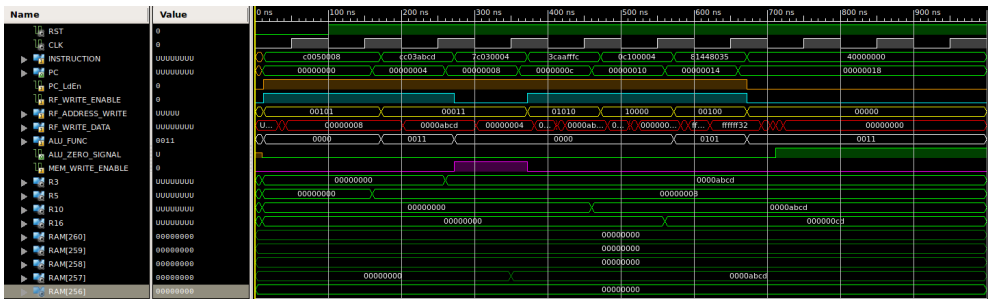


Figure 4: Report Program 1 - Simulation

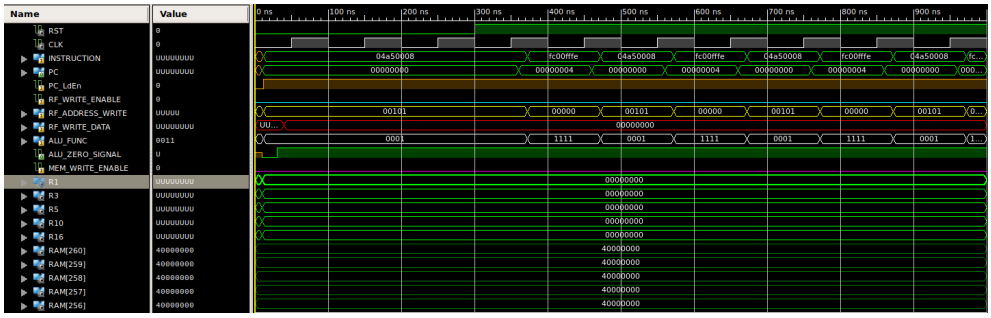


Figure 5: Report Program 2 - Simulation