# HARDWARE 1

## Εργασία Verilog

**Γιαννόπαπας Ματθαίος**

**10542**

**Email: mgiannopa@ece.auth.gr**

# Περιεχόμενα

# Εισαγωγή

Η εργασία αφορά την υλοποίηση βασικών κυκλωμάτων σε γλώσσα Verilog με σκοπό την εξοικείωση σε αυτήν. Ακολουθεί σχολιασμός τόσο στις ασκήσεις όσο και στην λογική που ακολουθήθηκε για την εκπόνηση τους.

# Άσκηση 1

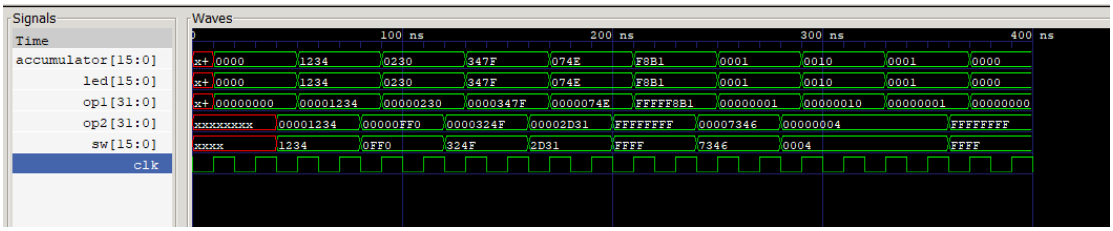
Στην πρώτη άσκηση καλούμαστε να δημιουργήσουμε μια Alu (Arithmetic Logic Unit), η οποία είναι υπεύθυνη για την εκτέλεση αριθμητικών λειτουργιών. Ο κώδικας βρίσκεται στο αρχείο alu.v.

Αρχικά έχουν οριστεί οι είσοδοι και οι έξοδοι του Module. Η μεταβλητή alu_op καθορίζει ποια πράξη θα εκτελεστεί. Με την βοήθεια παραμέτρων διαφοροποιούνται οι πράξεις μεταξύ τους και έπειτα με μια case ελέγχεται σε ποια τιμή αντιστοιχεί το alu_op, ώστε να εκτελεστεί η κατάλληλη πράξη. Οι πράξεις γίνονται όπως ορίζει το συντακτικό της γλώσσας και υπάρχει σαν σχόλιο τι εκτελεί η καθεμία ξεχωριστά συνεπώς δεν θα αναφερθεί περεταίρω στην αναφορά. Η χρήση ενός default μετά την συνθήκη, ορίζει σαν αποτέλεσμα το 0, καλύπτοντας το ενδεχόμενο λάθους στον καθορισμό των πράξεων.

# Άσκηση 2

Για την άσκηση 2 μας ζητείται να κατασκευάσουμε ένα κύκλωμα αριθμομηχανής, με την βοήθεια της ALU της προηγούμενης άσκησης. Ξεκινάμε, ορίζοντας τις θύρες μας και κάνοντας instantiate τόσο την alu όσο και το decoder(ο οποίος θα σχολιαστεί στην συνέχεια). Το πρώτο always block καθορίζει την τιμή του accumulator. Με το πάτημα το btnu ο accumulator μηδενίζεται, παρόλα αυτά σύγχρονα με το ρολόι, δηλαδή στην επόμενη θετική ακμή του. Ακόμη με το πάτημα του btnd ο accumulator ενημερώνεται με το αποτέλεσμα της alu και πάλι σύγχρονα με το ρολόι.

Στο επόμενο always block ενημερώνονται οι operands της alu με τις ζητούμενες τιμές με επέκταση πρόσημου σε αυτές. Η ενημέρωση αυτή γίνεται σε κάθε αλλαγή τόσου του accumulator όσο και του sw, όπως φαίνεται από το sensitivity list. Το αρχείο decoder.v υλοποίει την λογική πίσω από τα σχήματα που δίνονται. Τα ονόματα των ενδιάμεσων wires είναι όσο πιο σαφή γίνεται, αλλά είναι εκτενώς εξηγημένα και στα σχόλια του κώδικα. Στο αρχείο calctb.v δημιουργείται τόσο το αρχείο .vcd για την παραγωγή των κυματομορφών αλλά υπάρχει και ένα monitor που τυπώνει την τιμή του led σε κάθε test, για ακόμη καλύτερη κατανόηση.




Όπως φαίνεται και στην εικόνα αλλά και στο monitor, οι τιμές ήταν αυτές που αναμενόταν σύμφωνα με την εκφώνηση.

# Άσκηση 3

Η άσκηση 3 απαιτεί την δημιουργία ενός αρχείο καταχωρητών. Είναι αρκετά σημαντικό, καθώς αποθηκεύει τις τιμές που χρησιμοποιούνται από τον εκάστοτε επεξεργαστή. Αρχικά ορίζουμε τις θύρες, όπως μας δόθηκαν. Έπειτα μέσω ενός initial block γίνεται η αρχικοποίηση όλων των καταχωρητών στην τιμή 0, για την γρηγορότερη αρχικοποίηση τους χρησιμοποιήθηκε ένα for loop. Οι καταχωρητές μπορούν είτε να γράψουν είτε να διαβάσουν δεδομένα. Αυτό καθορίζεται μέσω του σήματος write, όταν είναι υψηλό οι τότε έχουμε εγγραφή δεδομένων, σε κάθε άλλη περίπτωση έχουμε διάβασμα αυτών. Τα παραπάνω υλοποιούνται σε ένα always block και πάντα σύγχρονα με το ρολόι.

# Άσκηση 4

Στην συγκεκριμένη άσκηση ξεκινάμε κάνοντας instantiate τις προηγούμενες υλοποιήσεις μας. Στη συνέχεια περνάμε στους καταχωρητές μας τα απαραίτητα bits από το instructions. Έπειτα ορίζουμε τις διάφορες τύπου εντολές της RISC-V και κάνουμε offset κατά ένα αριστερά το branch offset, όπως μας δίνεται στην εκφώνηση.

Στο always μπλοκ που ακολουθεί ανανεώνεται τιμή του PC, ανάλογα με την συνθήκη που ισχύει. Η ενημέρωση του PC γίνεται σε κάθε θετική ακμή του ρολογιού. Στο επόμενο block υλοποιείται η λογική ελέγχου για την διαδρομή δεδομένων, η οποία καθορίζει τα δεδομένα που πρέπει να εγγραφούν με βάση τις δοθείσες συνθήκες. Η εγγραφή εξαρτάται άμεσα από το σήμα MemToReg, αφού είναι αυτό που υποδηλώνει τι πρέπει να εγγραφεί. Τέλος πρέπει να καθορίσουμε ποιοι θα είναι οι όροι που θα εισαχθούν στην ALU. Ο πρώτος όρος προέρχεται πάντα από την θύρα readData1, ενώ ο δεύτερος μπορεί να προέρχεται είτε από την θύρα readData2 είτε από τα δεδομένα άμεσου προσήμου. Αξίζει να σημειωθεί πως η σειρά υλοποίησης των μπλοκ αυτών δεν επηρεάζει την άσκηση, καθώς «τρέχουν» ταυτόχρονα.

# Άσκηση 5

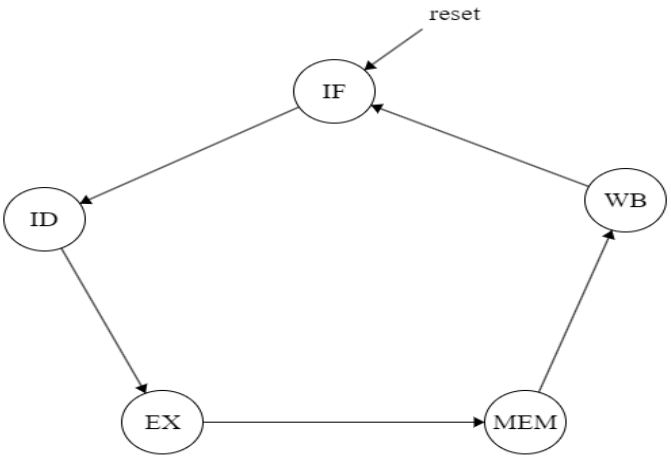
Στην άσκηση 5 κρίθηκε απαραίτητος ο ορισμός διαφόρων παραμέτρων, τόσο για την κατάσταση του FSM όσο και για την ευκολότερη αποκωδικοποίηση των εντολών.

Αρχικά γίνεται instantiate του module datapath. Ορίζουμε τα ανάλογα bits από του instructions σε opcode, funct3 και funct7 για την να γίνει πιο ευανάγνωστος και ξεκάθαρος ο κώδικας. Έπειτα υλοποιούμε το FSM, όπως μας δίνεται με τις αντίστοιχες καταστάσεις να διαδέχονται η μία την άλλη. Η κατάσταση IF (Instruction Fetch) επιλέχθηκε να είναι η κατάσταση σε περίπτωση reset. Έπειτα ξεκινά η διαδικασία αποκωδικοποίησης των εντολών, κατά την οποία χρησιμοποιείται τόσο το opcode, το funct3 όσο και το funct7. Για την αποκωδικοποίηση τους χρησιμοποιήθηκε το pdf που μας δόθηκε. Ανάλογα την τιμή των παραπάνω εκτελείται και διαφορετική λειτουργία, ο κώδικας περιέχει σχόλια που υποδεικνύουν τα bits και τις πράξεις που εκτελούν, για αυτό και θα γίνει εδώ περεταίρω αναφορά. Στο 4ο always block του κώδικα είναι η λογική πίσω από τον 2ο όρο της ALU, που αναφέρθηκε στην προηγούμενη άσκηση σε πιο απλοποιημένη μορφή. Στο τελευταίο κομμάτι υλοποιούνται τα σήματα της εκάστοτε κατάστασης του FSM, τα οποία λαμβάνουν συγκεκριμένες σύμφωνα με το στάδιο το οποίο βρίσκονται. Όλα τα παραπάνω γίνονται με την χρήση always blocks και σύγχρονα με το ρολόι.

Στο αρχείο top_proc_tb.v υπάρχει το αντίστοιχο testbench, γίνεται instantiate των ram, rom και multicycle και παράγεται το αντίστοιχο .vcd αρχείο, ώστε να λάβουμε τις κυματομορφές. Δεν ζητήθηκαν να δειχθούν κάποια συγκεκριμένα σήματα σε αυτές, συνεπώς επιλέχθηκαν οι θύρες της 5ης άσκησης.

Στις κυματομορφές επιλέχθηκαν δύο στιγμιότυπα, για την καλύτερη παρουσίαση των δεδομένων και ένα με την πλήρη παρουσίαση τους, από το οποίο δεν μπορεί να διακρίνει τις τιμές, αλλά δείχνει την λειτουργία του σε όλο το εύρος των τιμών του testbench.

Το σχηματικό του FSM μας είναι το παρακάτω:



Όλες οι εντολές θα περάσουν και από τα 5 στάδια, ανεξαρτήτως λειτουργίας και προέλευσης. Μία αναπαράσταση του FSM είναι και το σχήμα 8 της εκφώνησης. Τα στάδια διαδέχονται το ένα το άλλο και σε περίπτωση reset επιστρέφουμε στο Instruction Fetch. Αν υπάρξει reset, σε όποιο στάδιο και αν βρισκόμαστε θα πάμε στο IF, όμως για να μην επιβαρυνθεί το σχηματικό, επιλέχθηκε να μην γίνει ανάλογη σύνδεση από όλες τις καταστάσεις, αλλά να δειχθεί το reset με αυτό τον τρόπο που υποδηλώνει το παραπάνω.

Τα ονόματα των αρχείων κρατήθηκαν όπως υποδείκνυαν οι ασκήσεις(decoder etc), παρότι στο τέλος για την αναφορά υπήρχαν άλλες ονομασίες. Προτιμήθηκε ώστε να είναι κατανοητό πιο αρχείο επιτελεί την κάθε οδηγία της εκφώνησης.