Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστηριακές Ασκήσεις - Ψηφιακά Συστήματα HW-1

Σταύρος Σπυριδόπουλος 10845

Περιεχόμενα

1 Άσκηση 1

Σε αυτή την άκσηση ζητείται να υλοποιηθεί ένα συνδυαστικό κύκλωμα Αριθμητικής/Λογικής Μονάδας (ALU) το οποίο θα είναι υπεύθυνο για την εκτέλεση λειτουργειών μεταξύ δύο αριθμών 32-bit.
Αρχικά υλοποιήθηκε ένα `module alu` με εισόδους τους 2 αριθμούς και την λειτουργεία που θα εκτελεστεί και με εξόδους το αποτέλεσμα και τον καταχωρητή `zero`.
Στη συνέχεια με τη χρήση παραμέτρων `ALUOP_*` οριστικοποιήθηκαν οι διαφορετικοί 4-bit κωδικοί για κάθε πράξη.
Έπειτα, μέσα σε ένα `always block`, το οποίο ενεργοποιείται με κάθε αλλαγή στα σήματα εισόδου του `module`, γίνεται η υλοποίηση των αριθμητικών και λογικών πράξεων μέσω `cases` ώστε ο καταχωρητής εξόδου `result` να περιέχει το επιθυμητό αποτέλεσμα. Η προεπιλεγμένη τιμή του `result` είναι το 0.
Στις περιπτώσεις των λογικών πράξεων `op1<op` και `op1>>>op2[4:0]` υλοποίηθηκαν οι παρατηρήσεις της εκφώνησης όπως είναι εμφανές στο παρακάτω απόσπασμα κώδικα (1).
Τέλος, στην περίπτωση που το αποτέλεσμα της πράξης είναι 0, ο καταχωρητής εξόδου `zero` γίνεται 1 (`TRUE`).

```
1 ALUOP_SML : result = ($signed(op1) < $signed(op2)) ? 32'b1 : 32'b0;
2 ALUOP_ASR : result = $unsigned($signed(op1) >>> op2[4:0]);
```

Figure 1: Ειδικές περιπτώσεις λογικών πράξεων.

2 Άσκηση 2

Σε αυτή την άσκηση ζητείται να υλοποιηθεί μία αριθμομηχανή η οποία ανάλογα με τις εισόδος που θα δίνει ο χρήστης (πάτημα 5 πληκτρών) και με τη χρήση της `ALU` από την προηγούμενη άσκηση, θα αποθηκεύει το αποτέλεσμα στον καταχωρητή `accumulator`.

2.1 calc.v

Σε αυτό το αρχείο υλοποιείται το `module calc` με εισόδους τα κουμπιά του χρήστη και τον καταχωρητή `sw` και έξοδο τον καταχωρητή `led` που κατέχει την τιμή του `accumulator`.
Αρχικά γίνεται επέκταση προσήμου στις τιμές του `accumulator` και του `sw` ώστε να γίνουν 32-bit όπως οι είσοδοι της ήδη υπάρχουσας `ALU`.
Στη συνέχεια γίνεται η χρήση των `modules alu` (του αρχείου alu.v) και `alu_op_generator` (του αρχείου calc_enc.v) περνώντας τους τις ανανεωμένες 32-bit τιμές και τα κουμπιά του χρήστη.
Τέλος, μέσα σε ένα `always` block γίνεται η ανανέωση και το reset (σε περίπτωση πατήματος του κάτω κουμπιού) της τιμής του accumulator σε κάθε θετική ακμή του ρολογιού ώστε να ανανεώνεται σύγχρονα το αποτέλεσμα.

2.2 calc_enc.v

Σε αυτό το αρχείο υλοποιούνται τα λογικά κυκλώματα της εκφώνησης τα οποία συνδυάζουν τις εισόδους των κουμπιών και υπολογίζουν bit προς bit την λογική/αριθμητική πράξη (`alu_op`) που θα εκτελεστεί μεταξύ των δύο αριθμών.

2.3 Testbench

Αυτό το testbench προσομοιώνει και ελέγχει την λειτουργία του `calc module` εφαρμόζοντας πληθώρα αριθμητικών και λογικών πράξεων.
Εξετάζονται με σειρά οι περιπτώσεις της εκφώνησης:

- Τα κουμπιά εισόδου παίρνουν τιμές.
- Ο καταχωρητής sw παίρνει την προβλεπόμενη τιμή.
- Ενεργοποιείται το `btnd` και αρχίζει η πράξη.
- Τυπώνεται το επιθυμητό και εκτιμώμενο αποτέλεσμα επιτρέποντας τον έλεγχο από τον χρήστη.

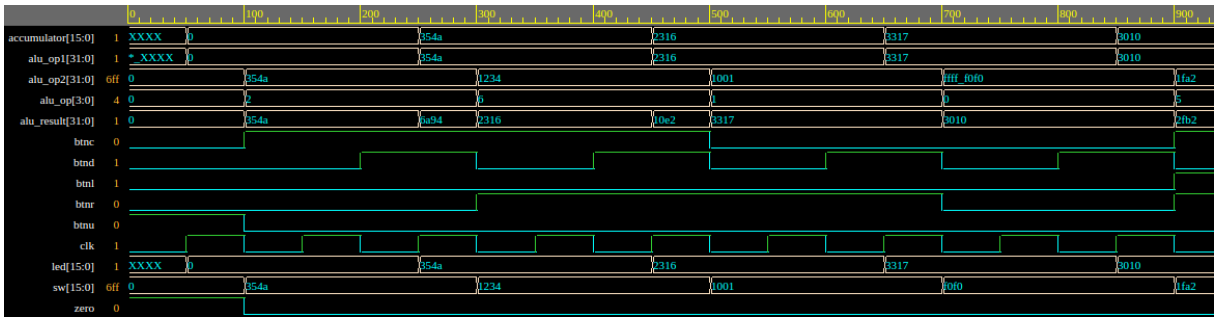

Figure 2: Σήματα Άσκησης 2 (0-900ps)

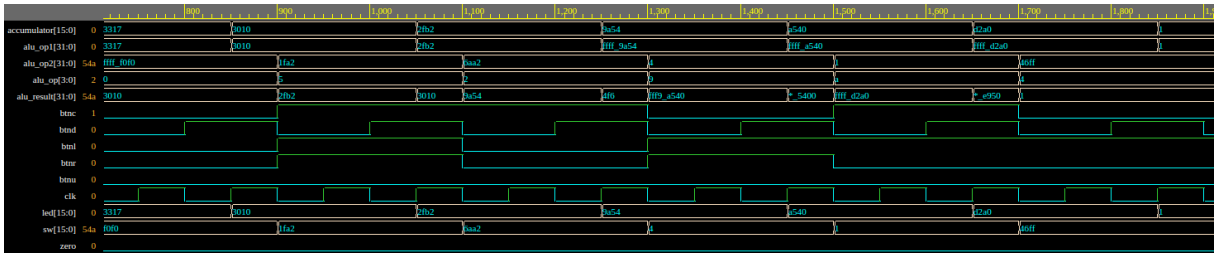

Figure 3: Σήματα Άσκησης 2 (900-1800ps)

3 Άσκηση 3

Σε αυτή την άσκηση υλοποιήθηκε το αρχείο καταχωρητών που θα χρησιμοποιηθεί και στην συνέχεια της εργασίας. Σε αυτό το αρχείο, το μέγεθος και το πλήθος των καταχωρητών εξαρτάται άμεσα από την παράμετρο `DATAWIDTH = 32`.
Ένα `module regfile` έχει οριστεί με εισόδους διευθύνσεις για τις θύρες ανάγνωσης (`readReg1, readReg2`) και την θύρα εγγραφής (`writeReg`) όπως και για τα δεδομένα που θα αναγνωστούν και θα εγγραφούν στις συγκεκριμένες θύρες (`readData1, readData2` και `writeData`).
Στην συνέχεια, όλοι οι καταχωρητές πέρνουν μηδενική τιμή και με την χρήση ενός `always` block γίνονται οι εγγραφές αν η είσοδος `write == 1`.
Στην περίπτωση που ο καταχωρητής εγγραφής έχει ίσια τιμή με κάποιον από τους 2 καταχωρητές ανάγνωσης, δίνεται προτεραιότητα στην εγγραφή των δεδομένων. Αυτό εύκολα υλοποιείται με την χρήση ενός `if` statement (4).

```
if(readReg1 != writeReg) begin
    readData1 = registers[readReg1];
end
if(readReg2 != writeReg) begin
    readData2 = registers[readReg2];
end
```

Figure 4: Προτεραιότητα εγγραφής σε εξαίρεση.

4 Άσκηση 4

Σε αυτή την άσκηση υλοποιείται η μονάδα datapath ενός RISC-V επεξεργαστή η οποία αποκωδικοποιεί και εκτελεί εντολές σύμφωνα με τα πρότυπα της συγκεκριμένης αρχιτεκτονικής.
Αρχικοποιείται ένα `module datapath` με παράμετρο τον αρχικό program counter, κύρια είσοδο μία εντολή 32 bits (`instr`) και εξόδους καταχωρητές που επιδιώκουν την ανάγνωση και εγγραφή στην μνήμη.
Στη συνέχεια γίνεται μία αποκωδικοποίηση της εντολής ανάλογα με το είδος της όπως φαίνεται παρακάτω (5).

```
wire [31:0] immI = {{20{instr[31]}}, instr[31:20]};
wire [31:0] immB = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
wire [31:0] immS = {{20{instr[31]}}, instr[31:25], instr[11:7]};
```

Figure 5: Αποκωδικοποίηση κάθε είδους εντολής.

- Στις εντολές τύπου I, η `immediate` τιμή προέρχεται από τα 12 πιο σημαντικά bits της εντολής (`instr[31:20]`) και επεκτείνεται με πρόσημο σε 32 bits.
- Στις εντολές τύπου S, η `immediate` τιμή σχηματίζεται από τη συνένωση των πιο σημαντικών `instr[31:25]` και λιγότερο σημαντικών `instr[11:7]` bits.
- Στις εντολές τύπου B, η `immediate` τιμή δημιουργείται από τα bits `instr[31]`, `instr[7]`, `instr[30:25]`, και `instr[11:8]`, με την προσθήκη ενός επιπλέον 0 στο τέλος για ευθυγράμμιση.

Έπειτα περνούνται όλες οι διευθύνσεις των `readReg1, readReg2` και `writeReg` στο `regifile` όπως και τα δεδομένα `readData1` και `op2` στην `ALU`.
Με στόχο να προωθούνται οι σωστές διατάξεις bit στις εκάστοτε κατηγορίες εντολών, γίνεται η χρήση ενός `always` block (6).

```
always @(*) begin
  case (instr[6:0])
    7'b0100011 : immediate = immS;  // Code for SW
    7'b1100011 : immediate = immB;  // Code for BEQ
    7'b0010011 : immediate = immI;  // Code for *I
    7'b0000011 : immediate = immI;  // Code for LW
  endcase
end
```

Figure 6: Είδος εντολής datapath.v

Για τον έλεγχο του Program Counter έχει υλοποιηθεί έα `always` block όπου:

- Σε περίπτωση που το σήμα επαναφοράς (`rst`) είναι `TRUE`, ο PC επανέρχεται στην αρχική του διεύθυνση (`INITIAL_PC`)
- Σε περίπτωση που το σήμα `loadPC` είναι `TRUE` υπάρχουν 2 τρόποι ενημέρωσης του PC:
  1. **Εντολή Branch**: Η τιμή του PC αυξάνεται κατά `Branch Offset`.
  2. **Επόμενη Διεύθυνση**: Η τιμή του PC αυξάνεται κατά 4.

Στη συνέχεια, σύμφωνα με το σήμα `ALUSrc` επιλέγεται ο δεύτερος τελεστής της ALU. Οι δύο δυνατές επιλογές είναι:

- Η δεύτερη θύρα δεδομένων ανάγνωσης του αρχείου καταχωρητών (`readData2`).
- Την formatted immediate type εντολή.

Τέλος, τα δεδομένα εξόδου για εγγραφή στη μνήμη (`dWriteData`) και εγγραφή πίσω στον καταχωρητή (`WriteBackData`) καθορίζονται ως εξής: Εάν το MemToReg είναι 1, η τιμή που θα εγγραφεί προέρχεται από το δεδομένο της μνήμης (`dReadData`). Εάν είναι 0, η τιμή που θα εγγραφεί προέρχεται από την έξοδο της ALU.

5 Άσκηση 5

Σε αυτή την άσκηση υλοποιείται η ροή δεδομένων μεταξύ των σταδίων ενός RISC-V επεξεργαστή και ένα testbench που επιβεβαιώνει τα αποτελέσματα της.

Αρχικά, καθορίζονται οι θύρες εισόδου και εξόδου, καθώς και η παράμετρος `INITIAL_PC`, η οποία χρησιμοποιείται για την αρχική ρύθμιση του `PC`.

Στη συνέχεια, το `module` συνδέεται με το datapath που υλοποιήθηκε στην προηγούμενη άσκηση, έτσι η παράμετρος `INITIAL_PC` περνάει στο `module`.

Ο κώδικας ορίζει και διαχειρίζεται τις καταστάσεις του pipeline για την οργάνωση της ροής δεδομένων και εντολών. Αυτή η διαδικασία υλοποιείται μέσω ενός *Finite State Machine* (FSM), το οποίο διαχειρίζεται τις φάσεις του pipeline: `IF` (Instruction Fetch), `ID` (Instruction Decode), `EX` (Execute), `MEM` (Memory Access), και `WB` (Write Back).

Επομένως, οι παράμετροι για διαφορετικούς τύπους εντολών καθορίζονται μέσω των πεδίων `opcode` και `funct3`, διακρίνοντας τις εντολές με βάση τα πρώτα 6 bits και τα bits 12-14. Οι εντολές κατηγοριοποιούνται ως μνήμης (`LW, SW`), διακλάδωσης (`BEQ`), ALU Immediate (`IMMEDIATE`), και Register-Register (`NON_IMMEDIATE`), επιτρέποντας την αποκωδικοποίηση του σήματος `ALUCtrl`.

Με ένα always block, που ενεργοποιείται σε κάθε `posedge` του ρολογιού (`clk`) ή του σήματος επαναφοράς (`rst`), αρχικοποιούνται όλες οι παράμετροι και τα σήματα στις αρχικές τους τιμές. Η κατάσταση του `FSM` ξεκινά από τη φάση `IF` και στη συνέχεια προχωρά διαδοχικά στις
`ID` → `EX` → `MEM` → `WB` → `IF`, ακολουθώντας ακολουθιακή λογική με non-blocking εντολές.

```
always @(posedge clk or posedge rst) begin
  if (rst) begin
    FSM <= IF;
    RegWrite <= 0;
    loadPC <= 0;
    ALUSrc <= 0;
    PCSrc <= 0;
    MemToReg <= 0;
    MemRead <= 0;
    MemWrite <= 0;
  end
  else begin
    case (FSM)
      IF:  FSM <= ID;
      ID:  FSM <= EX;
      EX:  FSM <= MEM;
      MEM: FSM <= WB;
      WB:  FSM <= IF;
    endcase
  end
end
```

Figure 7: Δομή εναλλαγής pipeline phases.

Σε άλλο `always` block, που εκτελείται με το ρολόι (`clk`), καθορίζονται οι φάσεις του pipeline ανάλογα με την τρέχουσα κατάσταση του `FSM`. Στη φάση `MEM`, ενεργοποιούνται τα `MemRead` (για `LW`) και `MemWrite` (για `SW`), ενώ στη φάση `WB` ενεργοποιούνται τα `loadPC`, `MemToReg` (για `LW`) και `RegWrite` (για όλες τις εντολές εκτός `SW` ή `BEQ`).

Επιπλέον, ο έλεγχος της ALU καθορίζεται σε ξεχωριστό `always` block. Αν το `opcode` είναι `NON_IMMEDIATE`, το `ALUCtrl` ορίζεται ανάλογα με το `funct3` και το bit 30 (instr[30]) της εντολής. Για `IMMEDIATE` εντολές, το `ALUCtrl` βασίζεται στο `funct3`. Ειδικά για `LW` και `SW`, το `ALUCtrl` τίθεται σε `ALUOP_ADD`, ενώ για `BEQ` σε `ALUOP_SUB`. Η διαφοροποίηση των `ARITHMETIC` και `SRL` γίνεται μέσω ενός `if` statement.

Τέλος, τα σήματα `ALUSrc` και `PCSrc` ενημερώνονται σε ακόμα ένα `always` block. Το `ALUSrc` είναι 1 για `LW`, `SW` ή `IMMEDIATE` εντολές, αλλιώς 0. Το `PCSrc` εξαρτάται από την τιμή του `Zero` για `BEQ`, ενώ για τις υπόλοιπες εντολές παραμένει 0.

5.1 Testbench

Για τον έλεγχο της λειτουργίας ενός RISC-V επεξεργαστή, δημιουργήθηκε το testbench ορίζοντας τα απαραίτητα modules και συνδέοντας τις εισόδους/εξόδους, όπως το `PC`, το `clk`, το `rst`, τη μνήμη εντολών `INSTRUCTION_MEMORY`) και τη μνήμη δεδομένων (`DATA_MEMORY`).
Τα πρώτα 9 bits του `PC` συνδέονται με την θύρα ανάγνωσης διευθύνσεων, ενώ το σήμα `dAddress` συνδέθηκε αντίστοιχα με τη μνήμη δεδομένων.
Το σήμα `clk` εναλλάσσεται κάθε $5ns$ μέσω ενός `always` block, ενώ το `rst` αρχικοποιείται σε 0 και απενεργοποιείται μετά από $15ns$ για την εκκίνηση του πυρήνα. Επιπλέον, μέσω των `$dumpfile` και `$dumpvars`, καταγράφηκαν όλες οι αλλαγές σημάτων σε αρχείο μορφής VCD για επαλήθευση.
Τέλος, ενεργοποιείται ο καταχωρητής `rst` και επαναφέρει τα σήματα στην αρχική τους κατάσταση, ενώ το πρόγραμμα τελειώνει μετά από για $1000ns$.

5.2 Σχηματικό Διάγραμμα FSM

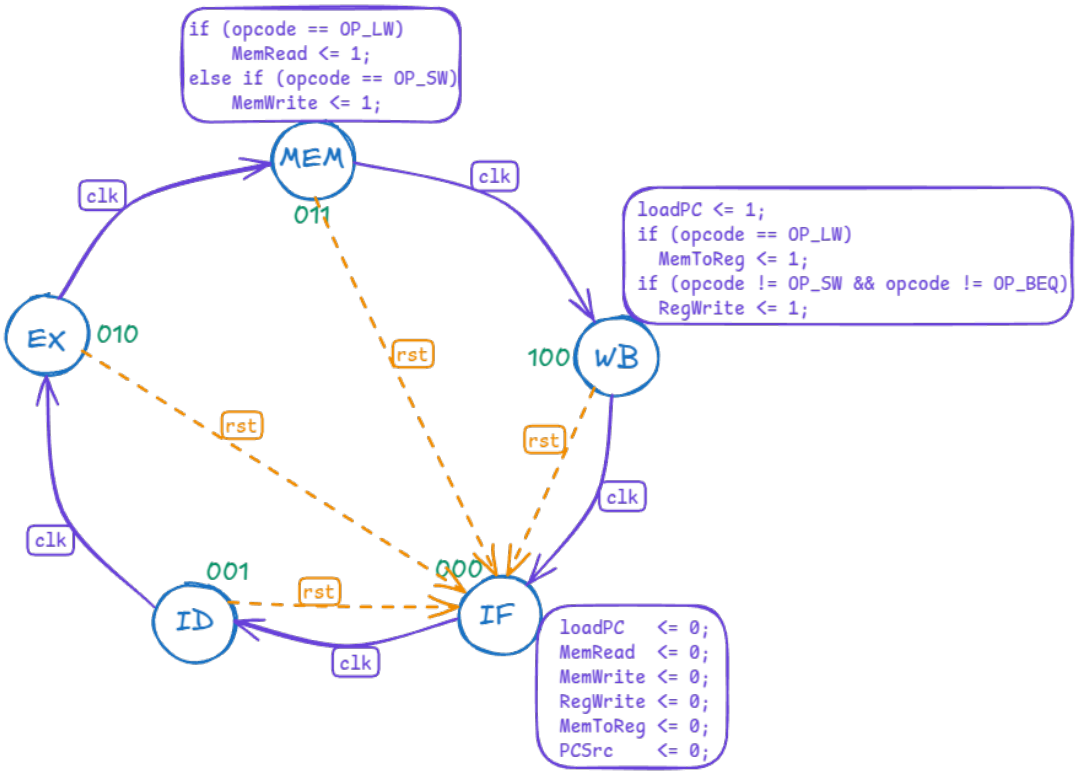

Figure 8: Σχηματικό Διάγραμμα από το FSM

5.3 Κυματομορφές Προσομοίωσης

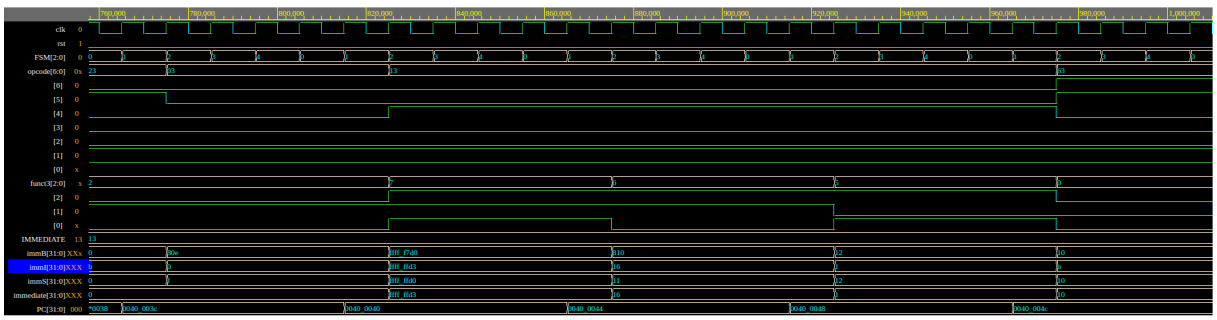

Figure 9: Κυματομορφές Προσομοίωσης

5.4 Εντολές Αρχείου rom_bytes.data

```
00000000011100000000000010010011    addi x1, x0, 7
00000001010100000000000100010011    addi x2, x0, 21
00000000001000001000000110110011    add x3, x1, x2
11111111011100000000001000010011    addi x4, x0, -9

11111110111100010000001010010011    addi x5, x2, -17
00000000010000101000001100110011    add x6, x5, x4
01000000001000011000001110110011    sub x7, x3, x2
00000000010100111001010000110011    sll x8, x7, x5

00000001000001000100010010110011    slt x9, x4, x8
00000000001001000100010100110011    xor x10, x8, x2
00000001000010101011101011110011    and x11, x10, x8
00000001001010111010110001100011    srl x12, x11, x9

00000000001101100110011010110011    or x13, x12, x3
01000001001001001010111100110011    sra x14, x4, x9
00000001011001010100000000100011    sw x11, 0(x5)
00000000000000010101001111000011    lw x15, 0(x5)

11111101001101000111100000010011    andi x16, x8, -45
00000001011010000110100010010011    ori x17, x16, 22
00000000000101101101100100010011    srli x18, x13, 1
00000001011011111000100001100011    beq x15, x11, 16

00000000000000000000000000000000    no instruction
00000000000000000000000000000000    no instruction
00000000000000000000000000000000    no instruction
00000000000000000000000000000000    no instruction

00000001111110010010010010010011    slti x9, x18, 15
00000011101001000100100110010011    xori x19, x8, 58
00000000000011000100110100010011    slli x20, x17, 1
01000000010011111101001010010011    srai x5, x15, 2
```

Figure 10: Εντολές Αρχείου rom_bytes.data

Όλες οι παραπάνω εντολές αποκωδικοποιήθηκαν με τη βοήθεια του online decoder : `https://luplab.gitlab.io/rvcodecjs/`