Αριστοτελειο Πανεπιστημιο Θεσσαλονικης
Πολυτεχνικη Σχολη
Τμημα Ηλεκτρολογων Μηχανικων & Μηχανικων Υπολογιστων

RV32I in Verilog

51 — Ψηφιακα Συστηματα HW σε Χαμηλα Επιπεδα Λογικης I

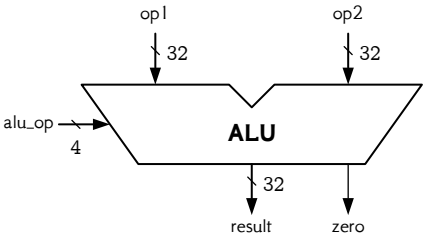
Καπετάνιος Αντώνιος [AEM 10417]
kapetaat@ece.auth.gr

Περίληψη

Στην παρούσα εργασία περιγράφεται σε Verilog ένας πυρήνας RISC-V περιορισμένων λειτουργιών βάσει του RV32I ISA. Στο πρώτο σκέλος περιγράφεται μία μονάδα αριθμητικών και λογικών πράξεων (ALU) η οποία, στο δεύτερο σκέλος, χρησιμοποιείται για την υλοποίηση μίας αριθμομηχανής. Στο τρίτο σκέλος, σχεδιάζεται το αρχείο καταχωρητών της αρχιτεκτονικής. Στο τέταρτο στάδιο, περιγράφεται το data path εντός του επεξεργαστή και τέλος, στο πέμπτο τμήμα, περιγράφεται η μονάδα ελέγχου και προσομοιώνεται η λειτουργία του επεξεργαστή. Η μεταγλώττιση του κώδικα πραγματοποιήθηκε χρήσει του Icarus Verilog[1] και τα waveforms παρήχθησαν με το λογισμικό GTKWave[2].

I ALU

Η μονάδα αριθμητικών και λογικών πράξεων (ALU) του σχήματος 1 έχει δύο εισόδους των 32 bit για τους τελεσταίους και μία είσοδο 4 bit για τον κωδικό της προς εκτέλεση πράξης. Το αποτέλεσμα της πράξης εμφανίζεται στην εξόδο των 32 bit. Τέλος, εάν το αποτέλεσμα της πράξης είναι μηδέν τότε ενεργοποιείται η έξοδος zero.



Σχήμα 1: Arithmetic logic unit (ALU). Οι είσοδοι των τελεσταίων είναι ακέραιοι αριθμοί.

Οι πράξεις που υποστηρίζει η μονάδα είναι οι εξής:

1. πρόσθεση: $op1 + op2$
2. αφαίρεση: $op1 - op2$
3. αριθμητική ολίσθηση δεξιά: $op1 \ggg op2[4:0]$
4. λογική σύζευξη: $op1 \wedge op2$
5. λογική διάζευξη: $op1 \vee op2$
6. λογική αποκλειστική διάζευξη: $op1 \oplus op2$
7. σύγκριση: $op1 < op2$
8. λογική ολίσθηση δεξιά: $op1 \gg op2[4:0]$
9. λογική ολίσθηση αριστερά: $op1 \ll op2[4:0]$.

Για την λογική πράξη της σύγκρισης πρέπει και οι δύο τελεσταίοι να είναι σε προσημανσμένη μορφή. Επιπλέον, για την αριθμητική ολίσθηση δεξιά θα πρέπει πρώτα ο τελεσταίος op1 να μετατραπεί σε προσημανσμένη μορφή κι έπειτα το αποτέλεσμα της ολίσθησης να μετατραπεί σε μη προσημανσμένη μορφή.

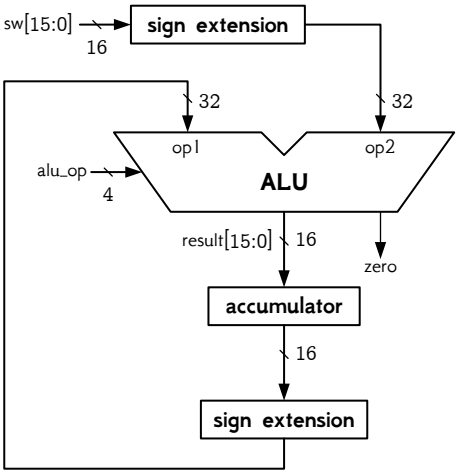
Τα παραπάνω περιγράφονται στο αρχείο `src/alu.v`. Ορίζεται ένα module με όνομα `alu` το οποίο έχει τις εισόδους op1, op2 και alu_op και τις εξόδους result και zero. Εντός του module δηλώνονται ως παράμετροι οι κωδικοί των πράξεων που υποστηρίζονται.

Το αποτέλεσμα της πράξης παράγεται από ένα πολυπλέκτη δύο 32 bit εισόδων και μίας 32 bit εξόδου. Το σήμα ελέγχου είναι ο κωδικός της πράξης. Μετά τον υπολογισμό του τελικού αποτελέσματος, εξετάζεται εάν είναι μηδέν και ενεργοποιείται η έξοδος zero αν η σύγκριση είναι αληθής.

[1] Github repository: `https://github.com/steveicarus/iverilog`
[2] Github repository: `https://github.com/gtkwave/gtkwave`

II Αριθμομηχανή

Η ALU της προηγούμενης ενότητας χρησιμοποιείται για την δημιουργία της αριθμομηχανής του σχήματος 2.



Σχήμα 2: Διάγραμμα ροής της αριθμομηχανής.

II.1 Module συσσωρευτή

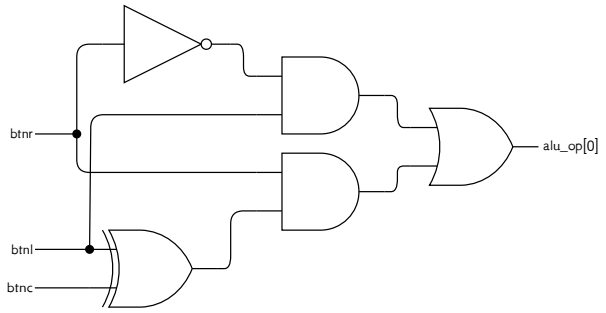
Τα 16 χαμηλότερα bit του αποτελέσματος της ALU περνάν ως είσοδος σε έναν συσσωρευτή των 16 bit. Ο συσσωρευτής λειτουργεί ως μνήμη της αριθμομηχανής και μηδενίζεται σύγχρονα με το πάτημα του άνω πλήκτρου και η τιμή του ενημερώνεται με κάθε πάτημα του κάτω πλήκτρου. Η έξοδός του συνδέεται με τα LED της αριθμομηχανής. Η υλοποίηση του συσσωρευτή περιγράφεται στο αρχείο `src/accum.v`. Το module `accum` έχει εισόδους το ρολόι, σήμα reset, σήμα για την ενημέρωση της τιμής του, τα δεδομένα

προς αποθήκευση και ως έξοδο έχει τα δεδομένα τα οποία ήταν ήδη αποθηκευμένα.

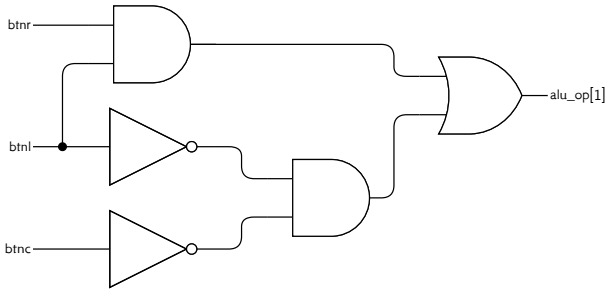
Εντός ενός `always` block με τις ανερχόμενες ακμές του ρολογιού στη λίστα ευαισθησίας του, εκτελείται ένα block εμφολευμένων `if`. Ο πρώτος έλεγχος αφορά το reset, το οποίο αν είναι ενεργό η έξοδος του συσσωρευτή γίνεται μηδέν. Εάν το σήμα reset είναι ανενεργό ελέγχεται το σήμα update. Αν το update είναι ενεργό η τιμή της εξόδου λαμβάνει την τιμή της εισόδου, διαφορετικά η έξοδος διατηρεί την τιμή της.

II.2 Module αποκωδικοποιητή

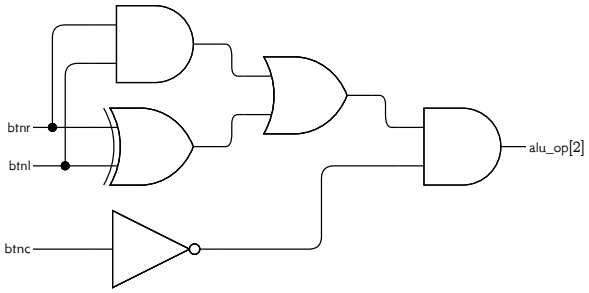
Η πράξη της ALU εισάγεται με συνδυασμό του αριστερού, δεξιού και κεντρικού πλήκτρου. Η αποδικοποίηση καθενός από τα τέσσερα bit του σήματος ελέγχου της ALU προκύπτει μέσω της λογικής των σχημάτων 3, 4, 5 και 6.



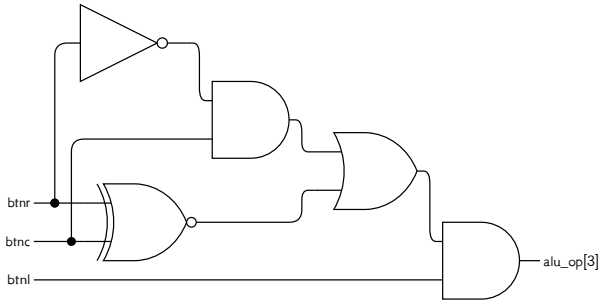
Σχήμα 3: LSB του σήματος ελέγχου της ALU.



Σχήμα 4: Bit 2 του σήματος ελέγχου της ALU.



Σχήμα 5: Bit 3 του σήματος ελέγχου της ALU.



Σχήμα 6: MSB του σήματος ελέγχου της ALU.

Οι μαθηματικές εκφράσεις που περιγράφουν τα παραπάνω σχήματα είναι

$$\text{alu\_op}[0] = (\neg\text{btnr} \wedge \text{btnl}) \vee (\text{btnr} \wedge (\text{btnl} \oplus \text{btnc})) \quad \text{(II.2.1)}$$

$$\text{alu\_op}[1] = (\text{btnr} \wedge \text{btnl}) \vee (\neg\text{btnl} \wedge \neg\text{btnc}) \quad \text{(II.2.2)}$$

$$\text{alu\_op}[2] = ((\text{btnr} \wedge \text{btnl}) \vee (\text{btnr} \oplus \text{btnl})) \wedge \neg\text{btnc} \quad \text{(II.2.3)}$$

$$\text{alu\_op}[3] = ((\neg\text{btnr} \wedge \text{btnc}) \vee \neg\,(\text{btnr} \oplus \text{btnc})) \wedge \text{btnl} \quad \text{(II.2.4)}$$

Στο αρχείο `src/decoder.v` ορίζεται ένα νέο module με όνομα `decoder` και εισόδους το κεντρικό, το αριστερό και το δεξί πλήκτρο. Η έξοδός του είναι τεσσάρων bit και αντιπροσωπεύει τον κωδικό της προς εκτέλεση πράξης της ALU.

Εντός ενός `always` block εφαρμόζονται οι παραπάνω σχέσεις χρήσει λογικών τελεστών της Verilog. Η λειτουργία, δηλαδή, του module περιγράφεται χρήσει structural Verilog.

II.3 Module αριθμομηχανής

Στο αρχείο `src/calc.v` ορίζεται ένα module με όνομα `calc` με εισόδους το ρολόι, πέντε πλήκτρα (κεντρικό, δεξί, αριστερό, άνω, κάτω), δεδομένα των 16 bit (δεύτερος τελεσταίος) και έξοδο σε LED.

Γίνεται instantiate του συσσωρευτή, της ALU και του αποκωδικοποιητή. Ως σήμα reset στον συσσωρευτή εφαρμόζεται το άνω πλήκτρο και ως update το κάτω πλήκτρο. Στην ALU, ως σήμα ελέγχου εφαρμόζεται η έξοδος του αποκωδικοποιητή.

Εντός ενός `always` block, τα 16 χαμηλότερα bit του αποτελέσματος της ALU συνδέονται στην είσοδο του συσσωρευτή. Η έξοδος του συσσωρευτή συνδέεται στα LED της αριθμομηχανής. Επιπλέον, γίνεται επέκταση προσήμου της τιμής των LED και το αποτέλεσμα συνδέεται στον πρώτο τελεσταίο της ALU. Ο δεύτερος τελεσταίος της ALU συνδέεται με τους διακόπτες δεδομένων μετά από επέκταση προσήμου.

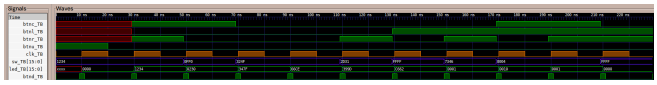
II.4 Testbench

Στο αρχείο `src/calc_TB.v`, αρχικά γίνεται instantiate η αριθμομηχανή. Έπειτα, σε ένα `initial` block μηδενίζεται το ρολόι, ενεργοποιείται το άνω πλήκτρο (reset) και στους διακόπτες δεδομένων δίνεται η αρχική τιμή $(1234)_{\text{HEX}}$. Μετά από 20ns το reset απενεργοποιείται.

Σε ένα `always` block χωρίς λίστα ευαισθησίας ανά 10ns η τιμή του ρολογιού συμπληρώνεται. Συνεπώς, η περίοδος είναι $T = 20\text{ns}$.

Σε ένα δεύτερο `always` block χωρίς λίστα ευαισθησίας ρυθμίζεται το κάτω πλήκτρο (update) ώστε να ενεργοποιείται 1ns πριν τη θετική ακμή του ρολογιού και να απενεργοποιείται 2ns μετά.

Εντός ενός δεύτερου `initial` block, μετά από 29ns (1ns πριν τη θετική ακμή του ρολογιού) ρυθμίζονται το αριστερό, κεντρικό και δεξί πλήκτρο για την πράξη της λογικής διάζευξης. Έπειτα, ανά 20ns ρυθμίζονται εκ νέου τα πλήκτρα και οι διακόπτες δεδομένων. Το χρονοδιάγραμμα φαίνεται στο διάγραμμα 1.



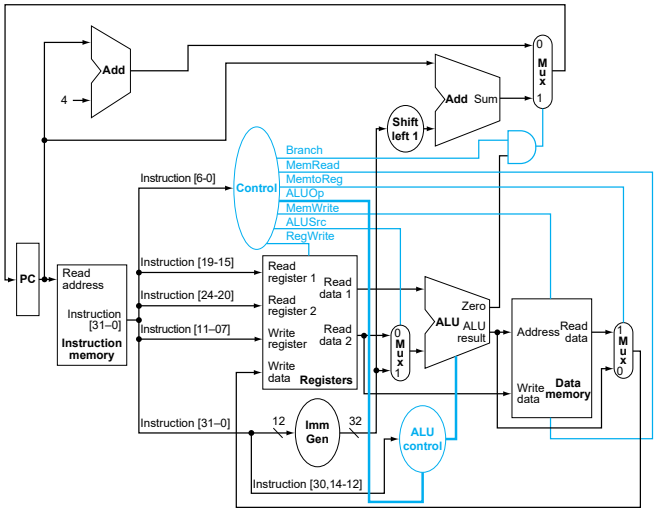
Διάγραμμα 1: Χρονοδιάγραμμα για το testbench της αριθμομηχανής. Επιβεβαιώνεται η ορθή λειτουργία της αριθμομηχανής.

III Αρχείο καταχωρητών

Στο αρχείο `src/regfile.v` δημιουργείται ένα νέο module με όνομα **`regfile`** το οποίο αναπαριστά το αρχείο καταχωρητών του επεξεργαστή. Το αρχείο καταχωρητών αποτελείται από 32 καταχωρητές των 32 bit. Οι είσοδοι του module είναι το ρολόι, η πρώτη και η δεύτερη διεύθυνση ανάγνωσης, η διεύθυνση εγγραφής, τα προς εγγραφή δεδομένα και το σήμα εγγραφής. Οι έξοδοι είναι τα δεδομένα των καταχωρητών των δύο διευθύνσεων ανάγνωσης.

Ορίζεται ένας πίνακας 32 **`reg`** το καθένα πλάτους 32 bit. Χρήσει ενός **`initial`** block και ενός επαναλήπτικού βρόχου **`for`** όλοι οι καταωρητές αρχικοποιούνται με την τιμή 0. Οι δύο έξοδοι του module ενημερώνονται μόλις αλλάξει η τιμή του προς ανάγνωση καταχωρητή. Τέλος, η εγγραφή πραγματοποιείται στις ανερχόμενες ακμές του ρολογιού εάν το σήμα εγγραφής είναι ενεργό. Διαφορετικά, τα δεδομένα των καταχωρητών παραμένουν αναλλοίωτα.

IV Data path



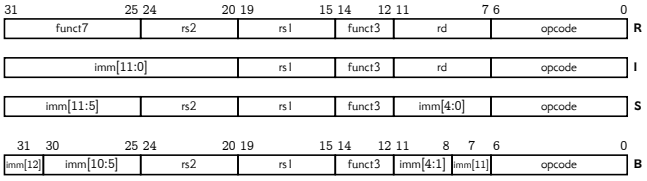
Σχήμα 7: RISC-V processor. Το σχηματικό είναι βασισμένο στο Σχήμα 4.17 του βιβλίου Computer Organization and Design[1].

Στην ενότητα αυτή, στο αρχείο `src/datapath.v`, υλοποιήθηκε η διαδρομή δεδομένων του επεξεργαστή του σχήματος 7. Οι είσοδοι του module **`datapath`** είναι το ρολόι, το σήμα reset, η προς εκτέλεση εντολή, το σήμα επιλογής του PC, το σήμα επιλογής του ALU source, το σήμα εγγραφής στο αρχείο καταχωρητών, το σήμα εγγραφής στο αρχείο καταχωρητών δεδομένων της κύριας μνήμης, ο κωδικός ελέγχου της ALU, το σήμα φόρτωσης της νέας διεύθυνσης του program counter και δεδομένα από την κύρια μνήμα. Όλα τα παραπάνω πλην του ρολογιού, του reset, της εντολής και του σήματος φόρτωσης του PC είναι σήματα που προέρχονται από το control unit. Οι έξοδοι του module είναι η νέα διεύθυνση του PC, το σήμα ένδειξης μηδενικού αποτελέσματος από την ALU, διεύθυνση της κύριας μνήμης, δεδομένα προς εγγραφή στην κύρια μνήμη και δεδομένα προς εγγραφή στο αρχείο καταχωρητών.

Οι εντολές που υποστηρίζονται από τον επεξεργαστή δίδονται στον πίνακα 1.

R-type	ADD	SLT	SLL	SRL
	AND	XOR	OR	SRA
	SUB			
I-type	ADDI	SLTI	SLLI	SRLI
	ANDI	XORI	ORI	SRAI
	LW			
S-type	SW			
B-type	BEQ			

Πίνακας 1: Υποσύνολο των εντολών RV32I που υποστηρίζονται από τον επεξεργαστή.



Σχήμα 8: R, I, S και B τύποι εντολών[2].

Αρχικά, στο module δηλώνονται οι παράμετροι των *opcodes* των υποστηριζόμενων εντολών και τα *funct3* πεδία της κάθε εντολής.[2]

Μέσω ενός **`initial`** block γίνεται η αρχικοποίηση του program counter. Έπειτα, με ένα **`always`** block το οποίο εκτελείται στις ανερχόμενες ακμές του ρολογιού υλοποιείται ο μηχανισμός του reset του program counter και ο πολυπλέκτης επιλογής της επόμενης τιμής του program counter. Συγκεκριμένα, εάν το σήμα reset είναι ενεργό η τιμή του PC επαναφέρεται στην αρχική. Διαφορετικά, εάν το σήμα φόρτωσης της νέας τιμής του PC είναι ενεργό ελέγχεται η τιμή του PCSrc (σχήμα 7). Εάν το PCSrc είναι ίσο με 1 τότε η νέα τιμή του PC είναι η τρέχουσα συν το branch offset, ειδάλλως είναι η τρέχουσα προσαυξημένη κατά 4.

Παρακάτω γίνεται instantiate το αρχείο καταχωρητών και υλοποιείται με ένα **`always`** block ο πολυπλέκτης μέσω του οποίου προκύπτει ο δεύτερος τελεσταίος της ALU. Ο πρώτος τελεσταίος είναι πάντα τα δεδομένα της πρώτης διεύθυνσης που δίνεται ως είσοδος στο αρχείο καταχωρητών. Εάν το σήμα ALUSrc είναι μηδέν, τότε ο δεύτερος τελεσταίος είναι τα δεδομένα της δεύτερης διεύθυνσης που δίνεται ως είσοδος στο αρχείο καταχωρητών. Διαφορετικά, ο δεύτερος τελεσταίος είναι το αποτέλεσμα του immediate generator του σχήματος 7.

Αμέσως μετά γίνεται instantiate η ALU και με ένα **`always`** block το οποίο εκτελείται όποτε αλλάζει η προς εκτέλεση εντολή, εξάγονται οι διευθύνσεις των καταχωρητών και παράγεται η σταθερή τιμή (immediate) βάσει του είδους της εντολής.

Για τις εντολές ολίσθησης τύπου I (**SLLI**, **SRLI** και **SRAI**) το immediate πλάτους 32 bit προκύπτει με zero extension της σταθεράς instr[24 : 20]. Για όλες τις υπόλοιπες εντολές τύπου I, συμπεριλαμβανομένης και της εντολής **LW**, το immediate πλάτους 32 bit προκύπτει με sign extension της σταθεράς instr[31 : 20].

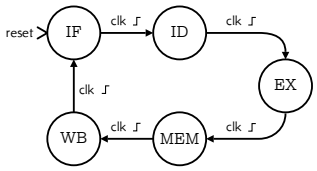
Για την εντολή **SW**. η οποία είναι τύπου S[2], η τιμή των 12 bit προκύπτει από τη συνένωση {instr[31 : 25], instr[11 : 7]} η οποία, στη συνέχεια, υφίσταται επέκταση προσήμου.

Τέλος, η εξαγωγή του immediate για τις εντολές τύπου B είναι πιο περίπλοκη. Το immediate πλάτους 12 bit προκύπτει από τη συνένωση $\{\text{instr}[31], \text{instr}[7], \text{instr}[30:25], \text{instr}[11:8]\}$. Η τιμή αυτή υφίσταται επέκταση προσήμου και στη νέα τιμή εκτελείται ολίσθηση προς τα αριστερά κατά 1 bit, το οποίο αριθμητικά ισούται με διπλασιασμό της τιμής. Η τελική τιμή είναι το branch offset.

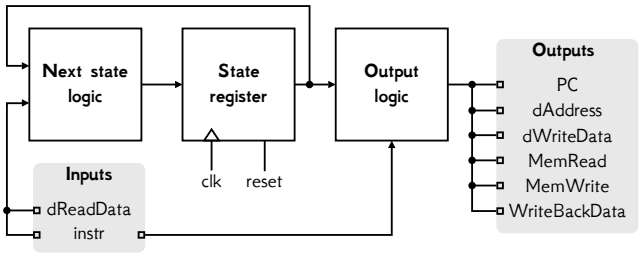
Τέλος, υλοποιείται ο πολυπλέκτης επιλογής των δεδομένων εγγραφής στο αρχείο καταχωρητών. Η διεύθυνση της κύριας μνήμης στην οποία θα εγγραφούν δεδομένα παίρνει την τιμή του αποτελέσματος της ALU. Τα δεδομένα προς εγγραφή στην κύρια μνήμη είναι τα δεδομένα του δεύτερου καταχωρητή εισόδου στο αρχείο καταχωρητών. Εάν το σήμα MemToReg είναι ενεργό, τότε τα δεδομένα προς εγγραφή στο αρχείο καταχωρητών προέρχονται από την κύρια μνήμη (εντολή `LW`). Διαφορετικά, τα δεδομένα προς εγγραφή στο αρχείο καταχωρητών προέρχονται από το αποτέλεσμα της ALU.

V Control Unit

Στην παρούσα ενότητα υλοποιείται η μονάδα ελέγχου του σχήματος 7 ως μηχανή πεπερασμένων καταστάσεων. Οι καταστάσεις της μηχανής είναι `IF` (instruction fetch), `ID` (instruction decoding), `EX` (execution), `MEM` (memory access) και `WB` (write back). Το διάγραμμα καταστάσεων φαίνεται στο διάγραμμα 2. Το διάγραμμα ροής του FSM φαίνεται στο σχήμα 9. Εφ' όσον η είσοδος επηρεάζει την έξοδο, το FSM είναι τύπου Mealy.



Διάγραμμα 2: Διάγραμμα καταστάσεων. Το reset επαναφέρει τη μηχανή στην κατάσταση IF. Η μετάβαση στην επόμενη κατάσταση εκτελείται σε κάθε ανερχόμενη ακμή του ρολογιού.



Σχήμα 9: Διάγραμμα ροής του FSM.

Στο αρχείο `src/multicycle.v` δημιουργείται module με όνομα `multicycle` και εισόδους το ρολόι, σήμα reset, τα 32 bit της προς εκτέλεση εντολής και δεδομένα από την κύρια μνήμη. Οι έξοδοι του module είναι η διεύθυνση του PC, η διεύθυνση εγγραφής δεδομένων στην κύρια μνήμη, τα δεδομένα προς εγγραφή στην κύρια μνήμη,το σήμα ανάγνωσης από την κύρια μνήμη, το σήμα εγγραφής στην κύρια μνήμη και τα δεδομένα προς εγγραφή στο αρχείο καταχωρητών.

Οι παράμετροι που δηλώνονται εντός του module είναι τα *opcodes* και τα *funct3* πεδία των υποστηριζόμενων εντολών, και οι κωδικοί των πράξεων για την ALU όπως έχουν δηλωθεί και στο `src/alu.v`.

Αφότου γίνει instantiate του data path, ξεκινάει η περιγραφή της λογικής του FSM. Για τις πέντε καταστάσεις της μηχανής χρησιμοποιείται one-hot κωδικοποίηση.

Με ένα `always` block περιγράφεται η μνήμη των καταστάσεων. Το block εκτελείται στις ανερχόμενες ακμές του ρολογιού και έχει σύγχρονο reset το οποίο θέτει ως τρέχουσα κατάσταση την `IF`. Διαφορετικά, η τρέχουσα κατάσταση παίρνει την τιμή της επόμενης κατάστασης.

Ο προσδιορισμός της επόμενης κατάστασης γίνεται χρήσει ενός `always` block με την τρέχουσα κατάσταση στη λίστα ευαισθησίας. Η διαδοχή των καταστάσεων είναι `IF` → `ID`, `ID` → `EX`, `EX` → `MEM`, `MEM` → `WB`, `WB` → `IF`. Η μετάβαση από τη μία κατάσταση στην επόμενη εξαρτάται μονάχα από την ταυτότητα της τρέχουσας κατάστασης.

Οι έξοδοι του FSM σε κάθε κατάσταση είναι ως εξής:

- `IF`: τα σήματα RegWrite, MemToReg, PCSrc και loadPC μηδενίζονται.
- `ID`: στο στάδιο αυτό ρυθμίζεται το σήμα ALUCtrl βάσει του opcode της εντολής.
  - εάν το opcode της εντολής είναι STORE το ALUCtrl ρυθμίζεται στην πράξη της πρόσθεσης.
  - Εάν το opcode είναι LOAD, το ALUCtrl ρυθμίζεται και πάλι στην πράξη της πρόσθεσης.
  - εάν το opcode είναι BRANCH, το ALUCtrl ρυθμίζεται στην αφαίρεση. Δεν χρειάζεται να ελεγχθεί το πεδίο funct3 για opcode BRANCH καθότι στην προκειμένη υποστήριζεται μόνο η εντολή `BEQ`.
  - για opcode OP_IMM το οποίο αντιστοιχεί στις εντολές τύπου I (πλην της `LW`) το ALUCtrl ρυθμίζεται βάση μόνον του funct3. Εξαίρεση αποτελούν οι εντολές ολίσθησης προς τα δεξιά. Το είδος της δεξιάς ολίσθησης, αριθμητική ή λογική, προσδιορίζεται από το bit 30. Ειδικότερα, εάν το bit 30 είναι μηδέν, η ολίσθηση είναι λογική. Διαφορετικά, εάν το bit 30 είναι μονάδα, η ολίσθηση είναι αριθμητική.
  - εάν το opcode είναι OP, το οποίο αντιστοιχεί στις εντολές τύπου R, η ρύθμιση του ALUCtrl εξαρτάται από το funct7 και το funct3. Εάν το funct7 είναι $(0000000)_{\text{BIN}}$, το ALUCtrl ρυθμίζεται απευθείας βάση του funct3. Εάν το funct7 είναι $(01000000)_{\text{BIN}}$ και το πεδίο funct3 είναι $(000)_{\text{BIN}}$ τότε η πράξη της ALU θα είναι αφαίρεση. Εάν το funct3 είναι $(101)_{\text{BIN}}$ η πράξη της ALU θα είναι αριθμητική ολίσθηση προς τα δεξιά. Ο διαχωρισμός ως προς το funct7 είναι απαραίτητος διότι οι εντολές `SUB` και `ADD` έχουν το ίδιο funct7. Ομοίως και οι εντολές `SRA` και `SRL` έχουν το ίδιο funct7.

  Επιπλέον, ρυθμίζεται το σήμα επιλογής του δεύτερου τελεσταίου της ALU. Για εντολές με opcode LOAD, STORE ή OP_IMM το ALUSrc παίρνει την τιμή 1 και ο δεύτερος τελεσταίος της ALU θα είναι ο immediate πλάτους 32 bit. Για όλα τα υπόλοιπα opcodes το ALUSrc απενεργοποιείται και ο δεύτερος τελεσταίος είναι τα δεδομένα του της δεύτερης διεύθυνσης που δίνεται ως είσοδος στο αρχείο καταχωρητών.
- `EX`: δεν τροποποιούνται οι έξοδοι του FSM σε αυτό το στάδιο.
- `MEM`: τα σχετικά με την κύρια μνήμη σήματα ρυθμίζονται βάσει του opcode της εντολής.

  - LOAD: το σήμα ανάγνωσης από τη μνήμη ενεργοποιείται και το σήμα εγγραφής στην μνήμη απενεργοποιείται.
  - STORE: το σήμα ανάγνωσης από τη μνήμη απενεργοποιείται και ενεργοποιείται το σήμα εγγραφής στη μνήμη.
  - BRANCH: απενεργοποιούνται τα σήματα εγγραφής και ανάγνωσης από τη μνήμη. Επιπλέον, ρυθμίζεται το σήμα PCSrc το οποίο ελέγχει την επόμενη τιμή του PC. Εάν το αποτέλεσμα της ALU είναι μηδέν, τότε ακολουθείται το branch και το PCSrc σήμα γίνεται 1. Διαφορετικά, το σήμα PCSrc γίνεται 0.
  - Η προκαθορισμένη συμπεριφορά είναι τα σήματα ανάγνωσης και εγγραφής στη μνήμη να είναι απενεργοποιημένα.

- `WB`: σε αυτό το στάδιο ενεργοποιείται το σήμα loadPC για την φόρτωση της επόμενης εντολής, απενεργοποιούνται τά σήματα ανάγνωσης και εγγραφής στην κύρια μνήμα και ρυθμίζονται τα σήματα εγγραφής στο αρχείο καταχωρητών βάση των opcodes.
  - LOAD: το σήμα εγγραφής στο αρχείο καταχωρητών και το σήμα εγγραφής δεδομένων που προέρχονται από τη μνήμη ενεργοποιούνται.
  - STORE: το σήμα εγγραφής στο αρχείο καταχωρητών και το σήμα εγγραφής δεδομένων που προέρχονται από τη μνήμη απενεργοποιούνται.
  - Λοιπά opcodes: το σήμα εγγραφής στο αρχείο καταχωρητών ενεργοποιείται και το σήμα εγγραφής δεδομένων που προέρχονται από τη μνήμη απενεργοποιείται.

V.1 Testbench

Στο αρχείο `src/top_proc_TB.v` γίνεται instantiate του module της μνήμης εντολών, του module της μνήμης δεδομένων και του module του control unit. Σε ένα `initial` block ενεργοποιείται το ρολόι και το σήμα reset το οποίο μένει ενεργό και 30ns. Εντός ενός `always` block ανά 10ns συμπληρώνεται το σήμα του ρολογιού. Επομένως, η περίοδος του ρολογιού είναι $T = 20$ns. Τέλος, η προσομοίωση ολοκληρώνεται μετά από 100μs.

Το αρχείο `sc/rom_bytes.data` περιέχει τα δεδομένα της μνήμης εντολών. Προκειμένου να γίνει πιο εύκολος ο έλεγχος του χρονοδιαγράμματος οι εντολές, μέσω ενός python script (`util/instructions.py`) που συνοδεύει την παρούσα αναφορά, αποκωδικοποιήθηκαν σε γλώσσα assembly και δίνονται στο listing 1. Τα immediates έχουν αποκωδικοποιηθεί σε προσημανσμένη δεκαδική μορφή.

Listing 1: Περιεχόμενα μνήμης εντολών σε μορφή γλώσσας assembly. Τα πεδία των καταχωρητών εμφανίζονται με τη σειρά του σχήματος 8

```
addi x1, x0, 4        #0x0000
addi x2, x0, 1        #0x0004
addi x3, x0, 3        #0x0008
addi x4, x0, 7        #0x000C
addi x5, x0, -2       #0x0010
add x6, x3, x3        #0x0014
sub x7, x6, x5        #0x0018
sll x8, x3, x2        #0x001C
slt x9, x8, x4        #0x0020
xor x10, x1, x7       #0x0024
srl x11, x10, x9      #0x0028
and x12, x3, x6       #0x002C
or x13, x10, x12      #0x0030
sra x14, x5, x2       #0x0034
sw x10, 0(x2)         #0x0038
lw x15, 0(x2)         #0x003C
andi x16, x13, 7      #0x0040
ori x17, x13, 4       #0x0044
srli x18, x4, 3       #0x0048
beq x6, x8, 6         #0x004C
None                  #0x0050
None                  #0x0054
None                  #0x0058
None                  #0x005C
None                  #0x0060
None                  #0x0064
None                  #0x0068
None                  #0x006C
None                  #0x0070
None                  #0x0074
None                  #0x0078
xori x19, x11, 12     #0x007C
slli x20, x19, 1      #0x0080
srai x21, x5, 1026    #0x0084
slti x22, x20, 28     #0x0088
None                  #0x008C
```

Στο χρονοδιάγραμμα 3 τα σήματα immediate_32, op1, op2 και result εκτυπώνονται με προσημανσμένη μορφή στο δεκαδικό σύστημα. Το σήμα του ρολογιού εκτυπώνεται πολλές φορές για ευκολία ανάγνωσης των κυματομορφών.

Διάγραμμα 3: Χρονοδιάγραμμα για τον επεξεργαστή του σχήματος 7.

Στο **πρώτο εκ των τεσσάρων διαγραμμάτων**, η πρώτη εντολή αποθηκεύει την τιμή 4 στον καταχωρητή `x1` προσθέτοντας το

4 στο περιεχόμενο του x0 το οποίο είναι μηδέν. Ο πρώτος τελεσταίος είναι 0 και ο δεύτερος έχει την τιμή του immediate, 4. Το αποτέλεσμα είναι 4 και το wrideData είναι επίσης 4. Οι επόμενες τέσσερις εντολές ακολουθούν την ίδια λογική. Η δεύτερη, κατά σειρά εκτέλεσης, εντολή γράφει την τιμή 1 στον καταχωρητή x2. Το op1 είναι 0, το op2 είναι 1, το result είναι 1 και το writeData είναι επίσης 1. Η τρίτη, κατά σειρά εκτέλεσης, εντολή γράφει την τιμή 3 στον καταχωρητή x3. Το op1 είναι 0, το op2 είναι 3, το result είναι 1 και το writeData είναι επίσης 3. Η τέταρτη, κατά σειρά εκτέλεσης, εντολή γράφει την τιμή 7 στον καταχωρητή x4. Το op1 είναι 0, το op2 είναι 7, το result είναι 1 και το writeData είναι επίσης 7. Η πέμπτη, κατά σειρά εκτέλεσης, εντολή γράφει την τιμή -2 στον καταχωρητή x5. Το op1 είναι 0, το op2 είναι -2, το result είναι -2 και το writeData είναι επίσης $(-2)_{\text{DEC}} = (4294967294)_{\text{HEX}}$. Η έκτη εντολή προσθέτει το περιεχόμενο του καταχωρήτη x3 στο περιοχόμενο του καταχωρητή x3 και αποθηκεύει το αποτέλεσμα στον καταχωρητή x6. Όντως, τα rs1 και rs2 έχουν την τιμή 03, τα op1 και op2 έχουν τιμή 3, το rd έχει την τιμή 6 και το result και το writeData έχουν την τιμή 6. Στην επόμενη εντολή, από το περιεχόμενο του καταχωρητή x6 αφαιρείται το περιοχόμενο του x5 και το αποτέλεσμα αποθηκεύεται στον καταχωρητή x7. Πράγματι, το rs1 είναι 6, το rs2 είναι 5 και το rd είναι 7. Το αποτέλεσμα είναι 8 και το writeData είναι επίσης 8, αφού op1 είναι 6 και op2 είναι -2. Στην επόμενη εντολή, το περιεχόμενο του καταχωρητή x3 ολισθαίνει κατά 1 bit προς τα αριστερά και το αποτέλεσμα αποθηκεύεται στον καταχωρητή x8. Το rs1 είναι 3, το rs2 είναι 2 και το rd είναι 8. Το op1 είναι 3 και το op2 είναι 1. Το αποτέλεσμα είναι 6, αφού η ολίσθηση αριστερά κατά 1 bit ισοδυναμεί με διπλασιασμό της τιμής και το writeData είναι επίσης 6. Στην επόμενη εντολή, το περιεχόμενο του καταχωρητή x8 συγκρίνεται με το περιεχόμενο του καταχωρητή x4. Επειδή το x8 είναι μικρότερο του x4, το αποτέλεσμα της σύγκρισης είναι 1 και το writeData είναι επίσης 1 το οποίο σύμφωνα με την εντολή αποθηκεύεται στον x7 και επιβεβαιώνεται αφού rd είναι 7. Στην δέκατη εντολή, το αποτέλεσμα της αποκλειστικής διάζευξης μεταξύ των bit των x1 και x7 αποθηκεύεται στον καταχωρητή x10. Το op1 είναι $4 = (0100)_{\text{BIN}}$ και το op2 είναι $8 = (1000)_{\text{BIN}}$. Το result είναι $12 = (1100)_{\text{BIN}}$ και το rd είναι $10 = (0\text{A})_{\text{HEX}}$.

Στο **δεύτερο εκ των τεσσάρων διαγραμμάτων** η πρώτη εντολή που εκτελείται (PC$= (00400028)_{\text{HEX}}$) είναι δεξιά ολίσθηση του x10 x10[4 : 0], δηλαδή κατά 1 bit. Το αποτέλεσμα είναι 6, αφού η δεξιά ολίσθηση κατά 1 bit ισοδυναμεί με υποδιπλασιασμό και το writeData είναι επίσης 6. Στην επόμενη εντολή πραγματοποιείται λογική σύζευξη των περιοχομένων των x3 και x6 και το αποτέλεσμα αποθηκέυεται στον x12. Είναι op1$= 6 = (110)_{\text{BIN}}$ και op2$= 6 = (011)_{\text{BIN}}$. Το result σήμα έχει την τιμή $2 = (010)_{\text{BIN}}$ και είναι σωστή. Στην επόμενη εντολή, εκτελείται λογική διάζευξη των περιεχομένων των x10 και x12. Το op1 είναι $12 = (1100)_{\text{BIN}}$ και το op2 είναι $2 = (0010)_{\text{BIN}}$ (εξαιτίας της προηγούμενης εντολής). Το αποτέλεσμα θα πρέπει να είναι $14 = (1110)_{\text{BIN}}$ το οποίο επιβεβαιώνεται από το χρονοδιάγραμμα. Αμέσως μετά, το περιεχόμενο το x5 υφίσταται αριθμητική ολίσθηση προς τα δεξιά κατά x2[4 : 0] και το αποτέλεσμα αποθηκεύεται στον x14. Το op1 είναι $-2 = (1 \ldots 10)_{\text{BIN,signed}}$ και το op2 1. Το αποτέλεσμα θα πρέπει να είναι -1 και πράγματι ισχύει.

Έπειτα, δοκιμάζονται οι εντολές προσπέλασης της μνήμης. Για PC$= (00400038)_{\text{HEX}}$ αποθηκεύεται στη μνήμη στη διεύθυνση που δείχνει ο καταχωρητής x2 η τιμή του καταχωρητή x10. Το σήμα MemWrite γίνεται ενεργό για έναν κύκλο και η τιμή που γράφεται στη μνήμη θα πρέπει να είναι 12. Όντως, το dWriteData είναι $12 = (0C)_{\text{HEX}}$. Αμέσως μετά, η τιμή που γράφθηκε στη μνήμη διαβάζεται και αποθηκεύται στον καταχωρητή x15. Τα σήματα MemToReg και MemRead ενεργοποιούνται το καθένα για ένα κύκλο και συγκεκριμένα, το MemRead ενεργοποιείται στον κύκλο πριν το MemToReg. Η τιμή που γράφεται στο αρχείο καταχωρητών είναι WriteData$= 12$ όπως ήταν και το αναμενόμενο.

Η επόμενη εντολή εκτελεί λογική σύζευξη των bit του καταχωρητή x13 με την τιμή $7 = (00111)_{\text{BIN}}$ και το αποτέλεσμα αποθηκεύεται στον καταχωρητή x16. Η τιμή του x13 (op1) είναι $14 = (01110)_{\text{BIN}}$. Το WriteData έχει την σωστή τιμή $6 = (00110)_{\text{BIN}}$. Η επόμενη εντολή εκτελεί λογική διάζευξη των bit του καταχωρητή x13 με την τιμή $4 = (00100)_{\text{BIN}}$ και το αποτέλεσμα αποθηκεύεται στον καταχωρητή x17. Η τιμή του x13 (op1) είναι $14 = (01110)_{\text{BIN}}$. Το WriteData έχει την σωστή τιμή $14 = (01110)_{\text{BIN}}$. Αμέσως μετά, εκτελείται δεξιά λογική ολίσθηση κατά 3 στον καταχωρητή x4 και το αποτέλεσμα αποθηκεύεται στον x18. Είναι op1$= 7 = (0111)_{\text{BIN}}$ και op2$= 3$. Το αποτέλεσμα είναι σωστά 0 και ενεργοποιείται το σήμα Zero της ALU. Τελευταία εντολή στο δεύτερο διάγραμμα είναι branch on equal. Συγκρίνεται η τιμή του x6 με την τιμή του x8. Επειδή είναι ίσες, το αποτέλεσμα της σύγκρισης είναι 1 και το Zero σήμα παραμένει ενεργό. Άρα το νέο PC θα υπολογιστεί με βάση το branch offset.

Στο **τρίτο εκ των τεσσάρων διαγραμμάτων** η πρώτη εντολή που εκτελείται είναι η εντολή στη διεύθυνση $(0040050)_{\text{HEX}}$ αφού η προηγούμενη εντολή ήταν branch και το Zero της ALU ήταν ενεργό. Η εντολή στην τρέχουσα διεύθυνση δεν εκτελεί κάποια πράξη. Δεν ισοδυναμεί αυστηρά με την `NOP` στο επίπεδο της κωδικοποίησης αλλά το αποτέλεσμα είναι ακριβώς το ίδιο. Η επόμενη όχι `NOP` εντολή εκτελείται στη διεύθυνση $(004007C)_{\text{HEX}}$. Επόμενη εντολή είναι αποκλειστική διάζευξη με immediate. Τα op1 και op2 είναι σωστά και το αποτέλεσμα είναι $10 = (01010)_{\text{BIN}}$. Η επόμενη εντολή, ολισθαίνει αριστερά το περιοχόμενο του x19 κατά 1 bit. Το αποτέλεσμα θα πρέπει να είναι το διπλάσιο της προηγούμενης τιμής του x19 το οποίο επιβεβαιώνεται. Η προτελευταία εντολή είναι αριθμητική ολίσθηση προς τα δεξιά του καταχωρήτή x5 (op1$= -2 = (1 \ldots 10)_{\text{BIN}}$) κατά 2. Το αποτέλσμα θα είναι $(1 \ldots 1)_{\text{BIN}} = -1$. Απομένει μία εντολή σύγκρισης με immediate. Γίνεται η σύγκριση $28 <$ x20. Από την παραπρογούμενη εντολή η τιμή του x20 είναι 20. Άρα το αποτέλεσμα είναι μηδέν.

Η ορθή λειτουργία του επεξεργαστή έχει επαληθευθεί βάσει των αποτελεσμάτων του χρονοδιαγράμματος 3.

VI Παρατηρήσεις

VI.1 Αρχείο καταχωρητών

Σχετικά με το αρχείο καταχωρητών, ο πρώτος καταχωρητής, x0, στην αρχιτεκτονική RISC είναι βραχυκυκλωμένος στη γείωση. Κατα αυτόν τον τρόπο διατηρεί πάντα σταθερή τιμή μηδέν. Εφόσον κάτι τέτοιο δεν αναφέρεται στις προδιαγραφές της εκφώνησης δεν υλοποιείται. Στην περίπτωση που θα απαιτούνταν η υλοποίηση της προδιαγραφής αυτής θα μπορούσε να γίνει με δύο τρόπους. Είτε με μία διαρκή ανάθεση της τιμής μηδέν στον καταχωρητή x0, είτε με έναν έλεγχο της διεύθυνσης εγγραφής. Εάν η διεύθυνση εγγραφής είναι ο καταχωρητής x0, τότε η εγγραφή αγνοείται.

Listing 2: Τροποποίηση του αρχείου καταχωρητών ώστε ο x0 να είναι βραχυκυκλωμένος στη γείωση.

```
66  always @(posedge clk) begin
67    if(write) begin
68      if(writeReg!=0)
69        x_reg[writeReg]<=writeData;
70    end
71  end
```

VI.2 Compilation

Για τη μεταγλώττιση του source code και την παραγωγή των dump files με τις τιμές για την προβολή των χρονοδιαγραμμάτων έχει προστεθεί το αρχείο `src/makefile`. Το target `all` κάνει compile το testbench της αριθμομηχανής και το testbench του επεξεργστή. Το target `calc` μεταγλωττίζει μόνο το testbench της αριθμομηχανής και το target `top` μεταγλωττίζει μόνο το testbench του επεξεργαστή. Υπάρχουν δύο επιπλέον targets για τη διαγραφή των αναπαράξιμων αρχείων.

Αναφορές

[1] David A. Patterson και John L. Hennessy. *Computer Organization and Design - The Hardware/Software Interface*. 5th. 2014. ISBN: 978-0-12-407726-3.

[2] Andrew Waterman και Krste Asanović. *The RISC-V Instruction Set Manual*. Τόμ. 1. 2019. URL: `https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf`.