Ψηφιακά Συστήματα HW-1

ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ

ΑΡΙΣΤΟΤΕΛΗΣ ΤΣΕΚΟΥΡΑΣ – ΔΙΠΛ. ΗΛ. ΜΗΧ. ΚΑΙ ΜΗΧ. ΥΠΟΛ.
ΒΑΣΙΛΗΣ ΠΑΥΛΙΔΗΣ – ΑΝ. ΚΑΘΗΓΗΤΗΣ, ΤΗΜΜΥ
ΑΠΘ | ΕΡΓΑΣΤΉΡΙΟ ΗΛΕΚΤΡΟΝΙΚΗΣ

Περιεχόμενα

Άσκηση 1

Μια αριθμητική/λογική μονάδα (Arithmetic Logic Unit (ALU)) είναι ένα σημαντικό στοιχείο κάθε συστήματος επεξεργαστή RISC-V. Αυτή η μονάδα είναι υπεύθυνη για την εκτέλεση αριθμητικών λειτουργιών, όπως πρόσθεση και αφαίρεση, καθώς και λογικών λειτουργιών, όπως AND/OR. Σε αυτή την άσκηση θα δημιουργήσετε μια ALU την οποία θα χρησιμοποιήσετε για τον επεξεργαστή RISC-V σε μία μεταγενέστερη άσκηση. Θα χρησιμοποιήσετε επίσης την ALU για να υλοποιήσετε μια απλή "αριθμομηχανή".

Η ALU σας θα σχεδιαστεί για να υλοποιεί τις ακόλουθες πράξεις: προσημασμένη πρόσθεση, προσημασμένη αφαίρεση, λογικό AND, λογικό OR, λογικό XOR, σύγκριση "Μικρότερο από" και τρεις διαφορετικές πράξεις ολίσθησης.

Ξεκινήστε την άσκηση δημιουργώντας ένα νέο αρχείο Verilog με όνομα **alu.v**. Ορίστε ένα νέο module με όνομα “alu” και προσθέστε τις ακόλουθες θύρες (τα ονόματα των θυρών ΠΡΕΠΕΙ να είναι ακριβή όπως ορίζονται στον πίνακα):

Όνομα θύρας	**Κατεύθυνση**	**Πλάτος [αρ. bit]**	**Σκοπός**
op1	Είσοδος	32	Τελεστής 1 σε συμπλήρωμα ως προς 2
op2	Είσοδος	32	Τελεστής 2 σε συμπλήρωμα ως προς 2
alu_op	Είσοδος	4	Δείχνει ποια λειτουργία πρέπει να εκτελεστεί
zero	Έξοδος	1	Δείχνει πότε το αποτέλεσμα της ALU είναι μηδέν
result	Έξοδος	32	Αποτέλεσμα

Η ALU που πρόκειται να δημιουργήσετε είναι μια "32-bit" ALU που σημαίνει ότι οι είσοδοι έχουν πλάτος 32-bit, και παράγει μια έξοδο πλάτους 32-bit. Η ALU θα εκτελέσει μία από τις διαφορετικές πράξεις μεταξύ των τελεστών 'op1' και 'op2' με βάση την τιμή του σήματος εισόδου 'alu_op'.

Αυτό το κύκλωμα είναι ένα "συνδυαστικό" κύκλωμα που σημαίνει ότι το αποτέλεσμα της εξόδου του κυκλώματος εξαρτάται μόνο από τις εισόδους ('op1', 'op2' και 'alu_op') και δεν υπάρχει μνήμη για την αποθήκευση της προηγούμενης κατάστασης (δεν υπάρχουν καταχωρητές και συνεπώς δεν υπάρχει ανάγκη για είσοδο σήματος ρολογιού ή επαναφοράς). Η λειτουργία που εκτελεί η ALU βασίζεται στο σήμα εισόδου 'alu_op' τεσσάρων bit, όπως ορίζεται στον ακόλουθο πίνακα:

alu_op	**Πράξη**	**Αποτέλεσμα**
0000	Λογική AND	op1 & op2
0001	Λογική OR	op1 | op2
0010	Πρόσθεση	op1 + op2
0110	Αφαίρεση	op1 - op2

0111	Μικρότερο από	op1 < op2
1000	Λογική ολίσθηση δεξιά κατά op2 bits	op1 >> op2[4:0]
1001	Λογική ολίσθηση αριστερά κατά op2 bits	op1 << op2[4:0]
1010	Αριθμητική ολίσθηση δεξιά κατά op2 bits	op1 >>> op2[4:0]
1101	Λογική XOR	op1 ^ op2

Προσοχή: Η πράξη "Μικρότερο από" πρέπει να γίνει πάνω σε προσημασμένους αριθμούς. Επίσης για την αριθμητική ολίσθηση, το op1 θα πρέπει να μετατραπεί σε προσημασμένο αριθμό και το αποτέλεσμα μετά ξανά σε μη προσημασμένο.

Πολυπλέκτης ALU

Το κύκλωμα ALU πρέπει να περιλαμβάνει έναν πολυπλέκτη για να επιλέγει ποια από τις λειτουργίες θα εκτελέσει. Μπορείτε να καθορίσετε τις σταθερές χρησιμοποιώντας την οδηγία parameter στη Verilog ως εξής:

```
parameter[3:0] ALUOP_SUB = 4'b0110;
```

Θα πρέπει να δημιουργήσετε μια σταθερά για κάθε μία από αυτές τις εννιά εντολές λειτουργίας της ALU. Προσθέστε όλες τις παραμέτρους μέσα στο αρχείο **alu.v**.

Άσκηση 2

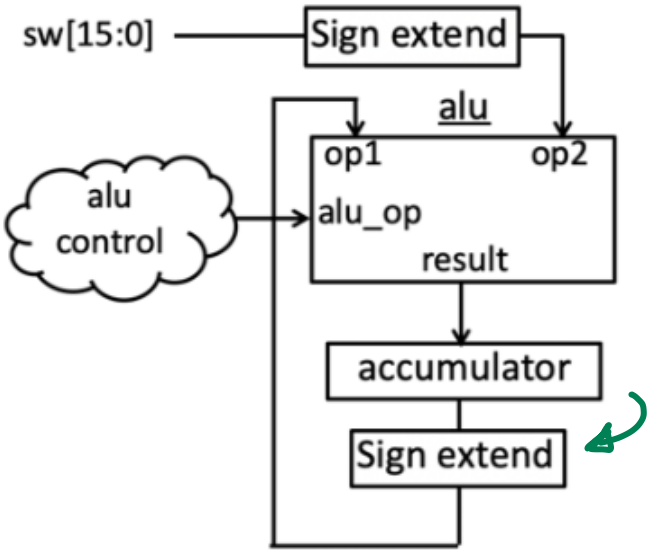
Για την άσκηση αυτή, θα σχεδιάσετε ένα κύκλωμα αριθμομηχανής που χρησιμοποιεί την ALU που δημιουργήσατε στην προηγούμενη άσκηση. Αυτό το κύκλωμα θα διατηρεί μια τρέχουσα τιμή της σε ένα συσσωρευτή 16-bit καταχωρητή και θα επιτρέπει στο χρήστη να ενημερώνει την τιμή υλοποιώντας οποιαδήποτε από τις αριθμητικές και λογικές συναρτήσεις που παρέχει η ALU σας.

Ξεκινήστε δημιουργώντας ένα νέο αρχείο Verilog με όνομα **calc.v**. Ορίστε μια νέα μονάδα με όνομα calc και προσθέστε τις ακόλουθες θύρες (τα ονόματα των θυρών ΠΡΕΠΕΙ να είναι ακριβή για να ταιριάζουν με το testbench):

Όνομα θύρας	**Κατεύθυνση**	**Πλάτος [αρ. bit]**	**Σκοπός**
clk	Είσοδος	1	Ρολόι
btnc	Είσοδος	1	Κεντρικό πλήκτρο
btnl	Είσοδος	1	Αριστερό πλήκτρο
btnu	Είσοδος	1	Πάνω πλήκτρο
btnr	Είσοδος	1	Δεξί πλήκτρο
btnd	Είσοδος	1	Κάτω πλήκτρο

sw	Είσοδος	16	Διακόπτες για την εισαγωγή δεδομένων
led	Έξοδος	16	LED για την έξοδο του συσσωρευτή

Τα δύο βασικά στοιχεία του κυκλώματος είναι ένας συσσωρευτής 16-bit για να κρατά την τρέχουσα τιμή της αριθμομηχανής σας και η ALU που δημιουργήσατε στην προηγούμενη άσκηση. Η σχέση της ALU και του συσσωρευτή πρέπει να σχεδιαστεί όπως φαίνεται στο ακόλουθο σχήμα:



Σχ. 1: Διάγραμμα ροής της αριθμομηχανής.

Δημιουργήστε έναν καταχωρητή 16 bit (που ονομάζεται accumulator) για να κρατάει τα περιεχόμενα της τρέχουσας τιμής της αριθμομηχανής. Σχεδιάστε τον καταχωρητή σας ως εξής:

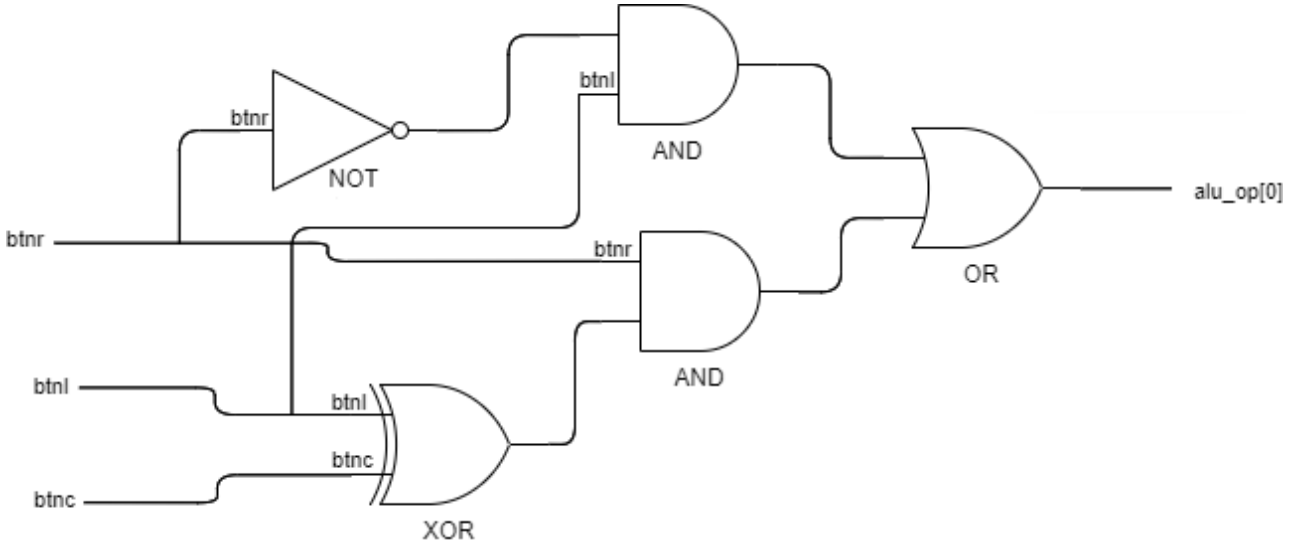
- Ο accumulator θα πρέπει να συνδεθεί με την είσοδο του ρολογιού.
- Ο accumulator θα πρέπει να μηδενίζεται σύγχρονα με το πάτημα του **btnu.**
- Η είσοδος του accumulator θα πρέπει να είναι τα 16 χαμηλότερα bit της εξόδου αποτελέσματος 32 bit της ALU.
- Ο accumulator θα πρέπει να ενημερώνεται κάθε φορά που πατιέται το "down" (**btnd**).
- Συνδέστε την τιμή του accumulator με τις εξόδους LED.

Συνδέστε τις εισόδους στην ALU ως εξής:

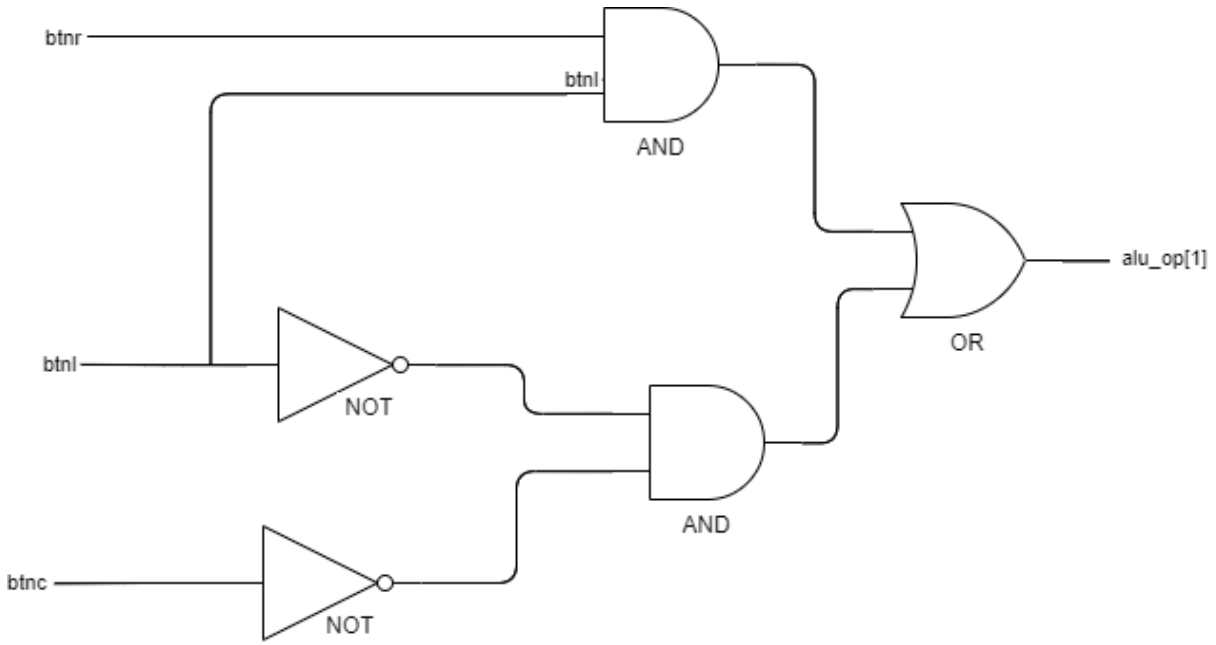
- Δημιουργήστε ένα σήμα 32-bit που είναι μια έκδοση με επέκταση προσήμου του accumulator 16-bit. Συνδέστε αυτό το σήμα στην είσοδο '**op1**' της ALU.
- Δημιουργήστε ένα σήμα 32-bit που είναι μια έκδοση με επέκταση προσήμου των εισόδων του διακόπτη 16-bit. Συνδέστε αυτό το σήμα στην είσοδο '**op2**' της ALU.
- Συνδέστε τα χαμηλότερα 16 bit της εξόδου "**result**" στην είσοδο του accumulator (όπως αναφέρθηκε παραπάνω).
- Δημιουργήστε ένα νέο σήμα για το '**alu_op**' καθώς και τη λογική για αυτό το σήμα όπως αναλύεται παρακάτω (Σχήματα 2-5).

Μπορείτε να δημιουργήσετε ένα σήμα 32-bit με επέκταση προσήμου χρησιμοποιώντας τον τελεστή concatenation της Verilog (δηλ., επαναλάβετε το κορυφαίο bit του σήματος που επεκτείνετε με πρόσημο όσες φορές χρειάζεται).

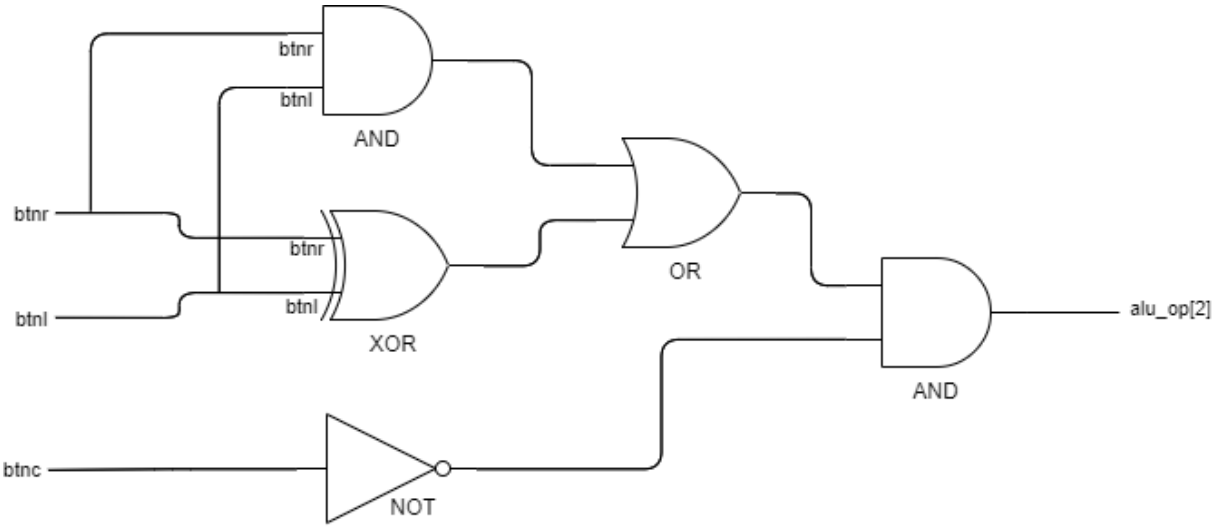
Το σήμα '**alu_op**' καθορίζει ποια πράξη της ALU θα εκτελεστεί. Θα καθορίσετε ποια λειτουργία θα εκτελέσετε με βάση την τιμή των τριών πλήκτρων: **btnl, btnc** και **btnr**. Θα πρέπει να δημιουργήσετε το συνδυαστικό κύκλωμα των Σχημάτων 2-5 που να παράγει το κατάλληλο σήμα '**alu_op**' με βάση την τιμή αυτών των τριών πλήκτρων. Υλοποιήστε το κύκλωμα **decoder.v** σε structural verilog και συνδέστε το στο κύκλωμα αριθμομηχανής **calc.v**.



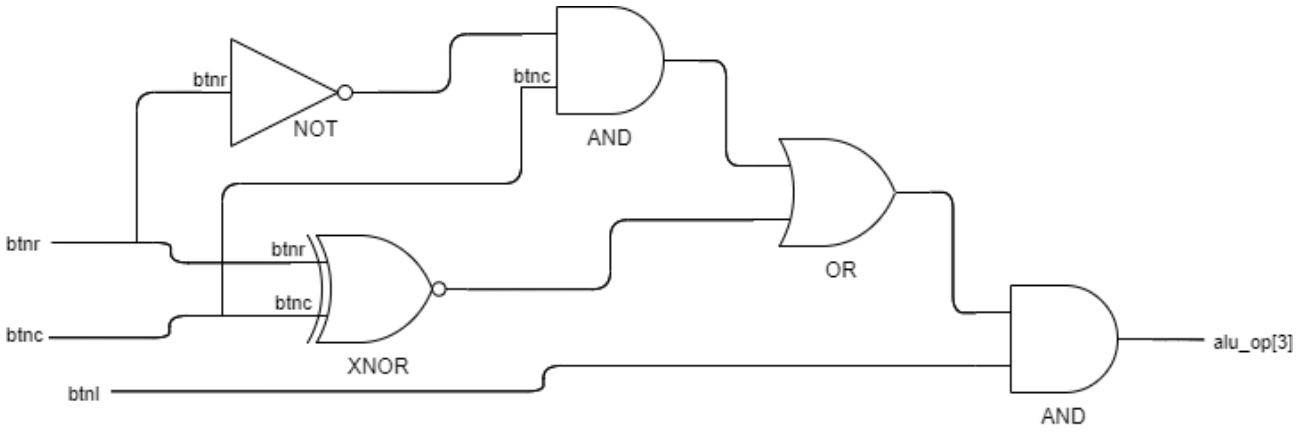
Σχ. 2: Παραγωγή του alu_op[0] μέσω των btnr, btnl, btnc.



Σχ. 3: Παραγωγή του alu_op[1] μέσω των btnr, btnl, btnc.



Σχ. 4: Παραγωγή του alu_op[2] μέσω των btnr, btnl, btnc.



Σχ. 5: Παραγωγή του alu_op[3] μέσω των btnr, btnl, btnc.

Testbench

Δημιουργήστε ένα testbench το οποίο θα ελέγχει την ορθή λειτουργία της αριθμομηχανής, καθώς και την ορθή λειτουργία της ALU. Ελέγξτε την αριθμομηχανή με τη σειρά για τις τιμές:

btnl, btnc, btnr (input)	**Previous value (acc.)**	**Switches (input)**	**Function in ALU**	**Expected Result**
(btnu for reset)	xxxx	xxxx	Reset	0x0
0,1,1	0x0	0x1234	OR	0x1234
0,1,0	0x1234	0x0ff0	AND	0x0230
0,0,0	0x0230	0x324f	ADD	0X347f
0,0,1	0x347f	0x2d31	SUB	0x074e
1,0,0	0x074e	0xffff	XOR	0xf8b1

1,0,1	0xf8b1	0x7346	Less Than	0x0001
1,1,0	0x0001	0x0004	Shift Left Logical	0x0010
1,1,1	0x0010	0x0004	Shift Right Arithmetic	0x0001
1,0,1	0x0001	0xffff	Less Than	0x0000

Προσοχή: Η αριθμομηχανή δεν θα υποστηρίζει λογική ολίσθηση προς τα δεξιά.

Άσκηση 3

Το αρχείο καταχωρητών είναι άλλο ένα σημαντικό στοιχείο κάθε συστήματος επεξεργαστή. Το αρχείο καταχωρητών αποθηκεύει τις τιμές των καταχωρητών που χρησιμοποιούνται από τον επεξεργαστή RISC-V. Σχεδόν όλες οι εντολές διαβάζουν ή/και γράφουν στο αρχείο καταχωρητών. Σε αυτή την άσκηση, θα δημιουργήσετε ένα αρχείο καταχωρητών το οποίο θα συμπεριλάβετε στον επεξεργαστή σας. Ξεκινήστε αυτή την άσκηση δημιουργώντας ένα νέο αρχείο Verilog με όνομα **regfile.v** και προσθέστε τις ακόλουθες θύρες σε αυτή τη μονάδα:

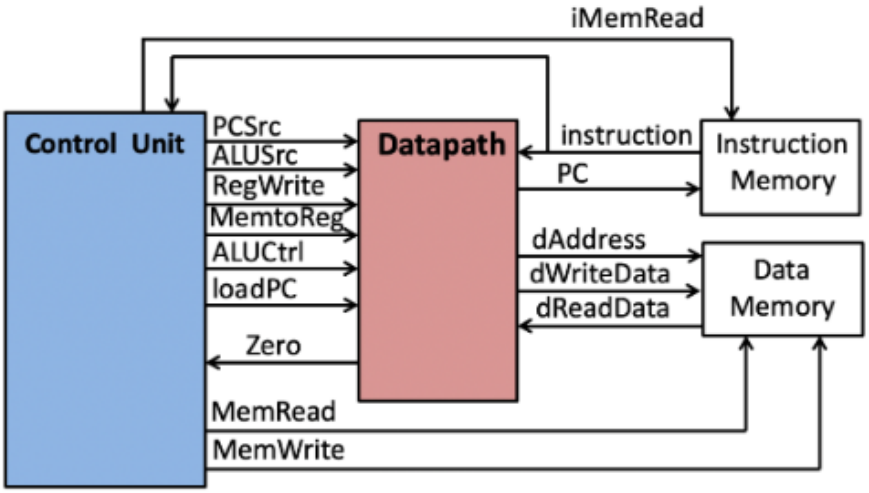
Όνομα θύρας	**Κατεύθυνση**	**Πλάτος [αρ. bit]**	**Σκοπός**
clk	Είσοδος	1	Ρολόι
readReg1	Είσοδος	5	Διεύθυνση για τη θύρα ανάγνωσης 1
readReg2	Είσοδος	5	Διεύθυνση για τη θύρα ανάγνωσης 2
writeReg	Είσοδος	5	Διεύθυνση για θύρα εγγραφής
writeData	Είσοδος	32	Δεδομένα προς εγγραφή
write	Είσοδος	1	Σήμα ελέγχου που υποδεικνύει εγγραφή
readData1	Έξοδος	32	Δεδομένα ανάγνωσης από τη θύρα 1
readData2	Έξοδος	32	Δεδομένα ανάγνωσης από τη θύρα 2

Υλοποιήστε ένα αρχείο καταχωρητών 32×32-bit. Αρχικοποιήστε τους 32 καταχωρητές με μηδενικά μέσω ενός initial block και μιας for. Ύστερα μέσω ενός always block διαβάστε τις τιμές των καταχωρητών από κάθε έξοδο (readData1, readData2) και ανάλογα με το σήμα write εγγράψτε τα δεδομένα από το writeData στην αντίστοιχη διεύθυνση που έχετε ως είσοδο. Προσέξτε την περίπτωση που η διεύθυνση εγγραφής είναι ίδια με κάποια από τις διευθύνσεις ανάγνωσης.

Άσκηση 4

Ένα διάγραμμα υψηλού επιπέδου του επεξεργαστή RISC-V που θα σχεδιάσετε φαίνεται στο Σχήμα 6. Το διάγραμμα αυτό περιέχει τα ακόλουθα τέσσερα στοιχεία:

- ☐ Μνήμη εντολών: Μια διευθυνσιοδοτούμενη μνήμη που περιέχει τις εντολές που υλοποιούν το πρόγραμμα που εκτελείται (το αρχείο .data που σας δίνεται, καθώς και το rom.v). Τα δεδομένα διαβάζονται μόνο από αυτή τη μνήμη.
- ☐ Μνήμη δεδομένων: Μια διευθυνσιοδοτούμενη μνήμη που περιέχει τα δεδομένα του χρήστη (το αρχείο ram.v). Περιέχει τη στοίβα και όλα τα δεδομένα χρήστη που χρησιμοποιούνται από το πρόγραμμα. Τα δεδομένα διαβάζονται και γράφονται σε αυτή τη μνήμη.
- ☐ Διαδρομή δεδομένων: Οι εσωτερικές λειτουργικές μονάδες, οι καταχωρητές και οι πολυπλέκτες που χρησιμοποιούνται για την υλοποίηση μεμονωμένων εντολών.
- ☐ Μονάδα ελέγχου: Ελέγχει τη λειτουργία της διαδρομής δεδομένων.



Σχ. 6: Διάγραμμα του επεξεργαστή. Στα πλαίσια της Άσκησης 4 θα σχεδιαστεί το datapath.

Θα σχεδιάσετε τη μονάδα διαδρομής δεδομένων σε αυτή την άσκηση και τη μονάδα ελέγχου στην επόμενη. Δεν θα σχεδιάσετε τη μνήμη εντολών και τη μνήμη δεδομένων (σας δίνονται).

Το μονοπάτι δεδομένων σας θα μπορεί να υποστηρίζει τις ακόλουθες εντολές:

Register-Register: ADD, SUB, AND, OR, XOR, SLT, SLL, SRL, SRA

ALU Immediate: ADDI, ANDI, ORI, XORI, SLTI, SLLI, SRLI, SRAI

Μνήμη: LW, SW

Διακλάδωση: BEQ

Προσοχή: Επειδή το κύκλωμά σας θα αποκωδικοποιεί εντολές RISC-V, είναι σημαντικό να κατανοήσετε πως να αποκωδικοποιείτε και να ερμηνεύετε σωστά αυτές τις δυαδικές εντολές. Θα βρείτε χρήσιμο το εγχειρίδιο των εντολών RISC-V κατά την εργασία σας πάνω σε αυτή και την επόμενη άσκηση. Ο πίνακας του συνόλου εντολών βρίσκεται στη σελίδα 130 (pdf σελίδα 148). Δώστε βαρύτητα στα R, I, S, B types.

Η υλοποίησή σας θα πρέπει να περιλαμβάνει την ALU που δημιουργήσατε στην Άσκηση 1 και το αρχείο καταχωρητών που δημιουργήσατε στην Άσκηση 3. Ξεκινήστε την άσκησή σας δημιουργώντας ένα αρχείο Verilog με όνομα **datapath.v** με τις θύρες που συνοψίζονται στον

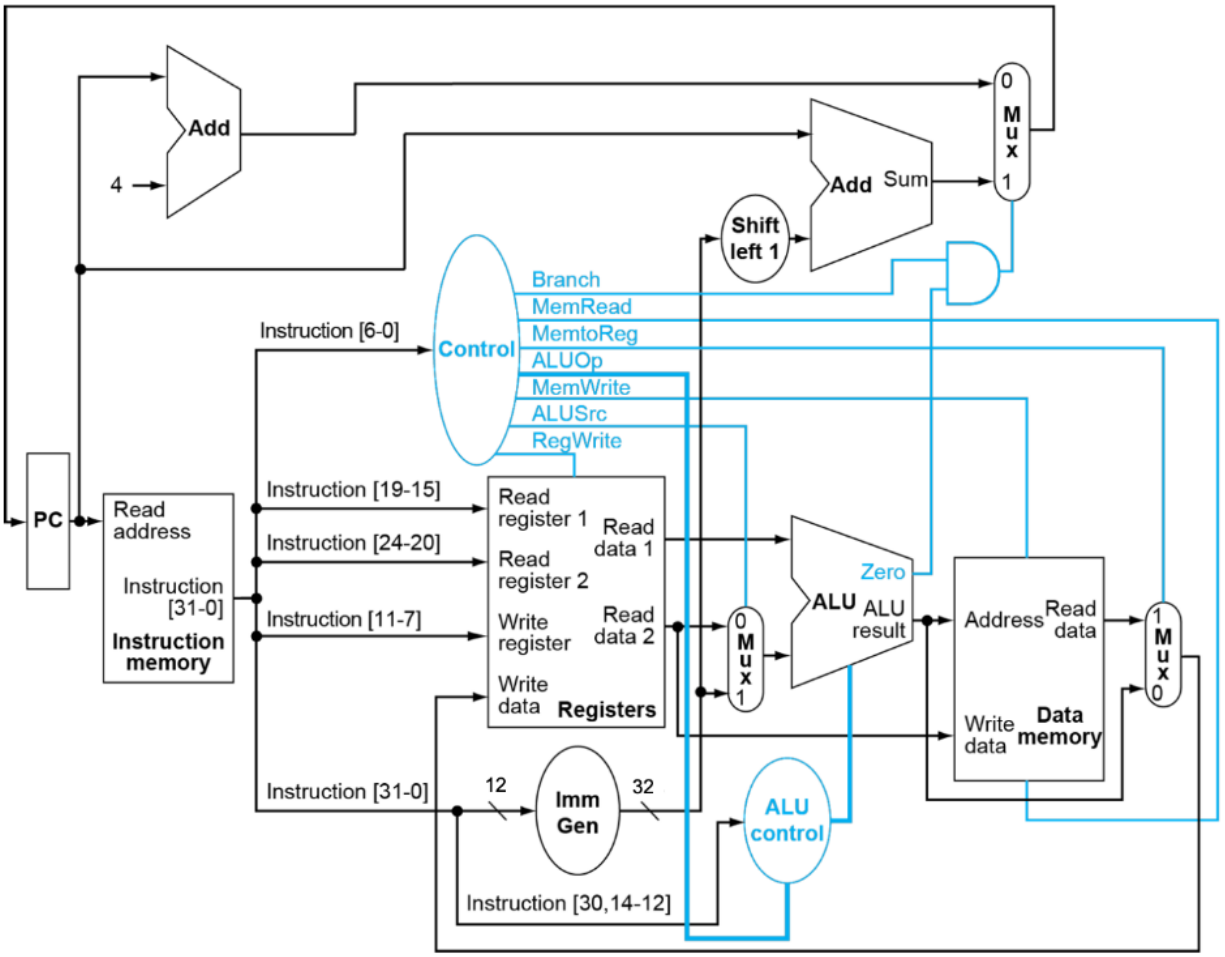
παρακάτω πίνακα μαζί με έναν πίνακα που δείχνει την παράμετρο που πρέπει να συμπεριλάβετε στο αρχείο Verilog σας.

Παράμετρος	**Πλάτος [αρ. bit]**	**Προεπιλεγμένη τιμή**
INITIAL_PC	32	0x00400000

Όνομα Θύρας	**Κατεύθυνση**	**Πλάτος [αρ. bit]**	**Σκοπός**
clk	Είσοδος	1	Ρολόι
rst	Είσοδος	1	Σύγχρονο Reset
instr	Είσοδος	32	Δεδομένα εντολών από τη μνήμη εντολών
PCSrc	Είσοδος	1	Πηγή του PC
ALUSrc	Είσοδος	1	Πηγή του 2ου τελεστή της ALU
RegWrite	Είσοδος	1	Εγγραφή δεδομένων στους καταχωρητές
MemToReg	Είσοδος	1	Πολυπλέκτης εισόδου στους καταχωρητές
ALUCtrl	Είσοδος	4	Δείχνει ποια λειτουργία πρέπει να εκτελέσει η ALU
loadPC	Είσοδος	1	Ενημέρωση του PC με μια νέα τιμή
PC	Έξοδος	32	Program Counter
Zero	Έξοδος	1	Ένδειξη μηδενισμού ALU
dAddress	Έξοδος	32	Διεύθυνση για δεδομένα Μνήμη
dWriteData	Έξοδος	32	Δεδομένα προς εγγραφή στη μνήμη δεδομένων
dReadData	Είσοδος	32	Δεδομένα προς ανάγνωση από τη μνήμη δεδομένων
WriteBackData	Έξοδος	32	WriteBack δεδομένα που επιστρέφουν στους καταχωρητές

Ο σκοπός του μονοπατιού δεδομένων είναι να λαμβάνει μια εντολή (instr), να αποκωδικοποιεί την εντολή (αναγνωρίζοντας τον τύπο εντολής και τα πεδία εντολής) για να ρυθμίζει κατάλληλα τους τελεστές της ALU (για παράδειγμα, τα σήματα των τελεστών της ALU θα είναι διαφορετικά από μια εντολή ADD και μια εντολή SW) και στη συνέχεια να τροφοδοτεί τα σήματα ελέγχου (τα οποία δίνονται στις θύρες εισόδου) όπου είναι απαραίτητο.

Η υπόλοιπη περιγραφή αυτής της άσκησης θα σας καθοδηγήσει στη διαδικασία σχεδίασης της διαδρομής δεδομένων σε Verilog. Λάβετε υπόψη τα ακόλουθα πέντε τμήματα μαζί με το Σχήμα 7 που σας δίνεται.



Σχ. 7: Διάγραμμα υλοποίησης για το datapath (κάποια σήματα δεν εμφανίζονται μέσα στο διάγραμμα). Τα μπλε τμήματα του διαγράμματος αποτελούν κομμάτι της επόμενης άσκησης, συνεπώς δε χρειάζεται να τα υλοποιήσετε για αυτήν.

Program Counter (PC)

Το πρώτο στοιχείο της διαδρομής δεδομένων είναι ο μετρητής προγράμματος (Program Counter) ή PC. Ο PC είναι ένας καταχωρητής που περιέχει την τιμή της διεύθυνσης μνήμης εντολών. Δείχνει τη θέση στη μνήμη εντολών όπου βρίσκεται η τρέχουσα εντολή που εκτελείται.

Σημειώστε ότι το αρχείο Verilog σας πρέπει να υποστηρίζει την παράμετρο "**INITIAL_PC**". Αυτή η παράμετρος χρησιμοποιείται για τον καθορισμό της αρχικής τιμής του PC όταν η διαδρομή δεδομένων επαναφέρεται με το σύγχρονο σήμα rst. Η προεπιλεγμένη τιμή είναι "0x00400000".

Το σήμα ελέγχου εισόδου **loadPC** χρησιμοποιείται για να υποδείξει ότι το PC πρέπει να ενημερωθεί στην επόμενη ακμή ρολογιού. Όταν αυτό το σήμα είναι υψηλό, το PC θα πρέπει να ενημερωθεί με μία από τις δύο από δύο διαφορετικές τιμές:

- PC+4 (όταν το πρόγραμμα προχωρά στην επόμενη εντολή στη μνήμη), και

- PC + branch_offset (όταν το πρόγραμμα πραγματοποιεί διακλάδωση).

Η απόφαση για το ποια από αυτές τις δύο επιλογές φορτώνεται στο PC υπαγορεύεται από το σήμα ελέγχου **PCSrc**. Οι λεπτομέρειες της μετατόπισης διακλάδωσης θα περιγραφούν παρακάτω.

Register File

Μέσα στη διαδρομή δεδομένων βρίσκεται το αρχείο καταχωρητών που περιέχει τους 32 καταχωρητές του επεξεργαστή. Προσθέστε το αρχείο καταχωρητών από την Άσκηση 3 στο μονοπάτι δεδομένων σας. Θα πρέπει να δημιουργήσετε λογική αποκωδικοποίησης που να καθορίζει τις τιμές στις θύρες διευθύνσεων του αρχείου καταχωρητών (δηλαδή, **readReg1, readReg2,** και **writeReg**) από την εντολή που είναι έτοιμη από τη μνήμη εντολών (δηλαδή, την είσοδο **instr**). Το σήμα εγγραφής καταχωρητή θα πρέπει να συνδεθεί στην είσοδο ελέγχου **RegWrite** του ανώτερου επιπέδου (θα ελέγχεται από τη μονάδα ελέγχου που ολοκληρώνεται στο επόμενο εργαστήριο).

Immediate Generation

Για τις εντολές τύπου I (άμεσες), η πηγή δεδομένων που εισέρχεται στη δεύτερη θύρα ανάγνωσης της ALU είναι τα 12 MSB της λέξης εντολής (**instr[31:20]**), η οποία έχει επεκταθεί με πρόσημο σε μια τιμή των 32 bit. Δημιουργήστε ένα νέο σήμα στο μονοπάτι δεδομένων σας που δημιουργεί την τιμή 32-bit με επέκταση προσήμου για τα άμεσα δεδομένα προς χρήση από την ALU.

Θα χρειαστεί επίσης να δημιουργήσετε μια άμεση τιμή και για τις εντολές τύπου S (store) από την εντολή που διαβάζεται από τη μνήμη εντολών. Η άμεση τιμή για εντολές τύπου S είναι διαφορετική από την άμεση τιμή για εντολές τύπου I (δείτε pdf σελίδα 148) και συνεπώς θα χρειαστεί ξεχωριστό σήμα. Το ίδιο πρέπει να ακολουθηθεί για τις εντολές τύπου B (BEQ).

ALU

Το επόμενο σημαντικό στοιχείο της διαδρομής δεδομένων είναι η αριθμητική λογική μονάδα (ALU). Θα πρέπει να χρησιμοποιήσετε την ALU από την Άσκηση 1 στο κύκλωμα της διαδρομής δεδομένων σας. Η είσοδος στον πρώτο τελεστή της ALU (**op1**) είναι η έξοδος από τη θύρα **readData1** από το αρχείο καταχωρητών. Ο δεύτερος τελεστής στην ALU μπορεί να προέρχεται από μία από τις δύο πηγές: από τη δεύτερη θύρα δεδομένων ανάγνωσης του αρχείου καταχωρητών (**readData2**) ή από τα δεδομένα άμεσου προσήμου extended που δημιουργούνται από την εντολή όπως περιγράφεται παραπάνω στο τμήμα “Immediate Generation”.

Το σήμα ελέγχου **ALUSrc** υπαγορεύει ποια είσοδος θα χρησιμοποιηθεί. Δημιουργήστε έναν πολυπλέκτη που επιλέγει μεταξύ αυτών των δύο σημάτων και οδηγεί την είσοδο **op2** της ALU σας. Συνδέστε το σήμα **ALUCtrl** του κορυφαίου επιπέδου στην είσοδο **alu_op**. Όπως περιγράφηκε στην Άσκηση 1, το σήμα **Zero** χρησιμοποιείται για να δείχνει πότε το αποτέλεσμα της ALU είναι μηδέν. Αυτό το σήμα χρησιμοποιείται για να αποφασιστεί και πότε γίνονται οι διακλαδώσεις, όπως φαίνεται και στο Σχήμα 7.

Branch Target

Η λογική της διαδρομής δεδομένων πρέπει να καθορίσει τη διεύθυνση στόχου διακλάδωσης που χρησιμοποιείται όταν γίνεται μια διακλάδωση (BEQ).

Ο στόχος μιας διακλάδωσης υπολογίζεται με την πρόσθεση της τρέχουσας τιμής του PC στο "branch offset" που υπολογίζεται με την αποκωδικοποίηση της εντολής instr (τύπος εντολής B). Σημειώστε ότι το branch offset αναφέρεται σε μισές λέξεις, πράγμα που σημαίνει ότι το offset πρέπει να μετατοπιστεί αριστερά κατά 1 (δεν μπορεί να κωδικοποιήσει μονές διευθύνσεις). Υλοποιήστε τον κώδικα Verilog που απαιτείται για τον υπολογισμό του branch offset. Επίσης, δημιουργήστε το σήμα της διακλάδωσης προσθέτοντας αυτό το branch offset στο τρέχον PC. Αυτός ο στόχος διακλάδωσης φορτώνεται στο PC όπως περιγράφθηκε στο τμήμα “Program Counter”.

Write Back

Το τελευταίο κομμάτι του μονοπατιού δεδομένων είναι η λογική της “εγγραφής προς τα πίσω” που καθορίζει ποια τιμή εγγράφεται στο αρχείο καταχωρητών. Η τιμή που εγγράφεται στο αρχείο καταχωρητών είναι μία από τις εξής δύο τιμές: το αποτέλεσμα της ALU για τις συμβατικές αριθμητικές και λογικές εντολές ή το αποτέλεσμα της ανάγνωσης μνήμης (**dReadData**). Το σήμα ελέγχου **MemtoReg** υπαγορεύει ποιο από αυτά τα δύο σήματα χρησιμοποιείται από το αρχείο καταχωρητών. Δημιουργήστε έναν πολυπλέκτη που επιλέγει μεταξύ αυτών των δύο σημάτων και συνδέστε την έξοδο του πολυπλέκτη στην είσοδο **writeData** του αρχείου καταχωρητών. Τα δεδομένα που γράφονται στους καταχωρητές πρέπει επίσης να συνδεθούν στην έξοδο **WriteBackData** του κυκλώματος.

Άσκηση 5

Για αυτή την άσκηση, θα δημιουργήσετε έναν ελεγκτή πολλαπλών κύκλων που εκτελεί κάθε εντολή σε πέντε κύκλους ρολογιού. Ξεκινήστε την άσκησή σας δημιουργώντας ένα αρχείο Verilog με όνομα **multicycle.v**. Οι θύρες που πρέπει να έχει το module είναι οι ακόλουθες:

Παράμετρος	**Πλάτος [αρ. bit]**	**Προεπιλεγμένη τιμή**
INITIAL_PC	32	0x00400000

Όνομα Θύρας	**Κατεύθυνση**	**Πλάτος [αρ. bit]**	**Σκοπός**
clk	Είσοδος	1	Ρολόι
rst	Είσοδος	1	Σύγχρονο Reset
instr	Είσοδος	32	Δεδομένα εντολών από τη μνήμη εντολών
dReadData	Είσοδος	32	Ανάγνωση δεδομένων από τη μνήμη δεδομένων
PC	Έξοδος	32	Program Counter
dAddress	Έξοδος	32	Διεύθυνση για δεδομένα μνήμης
dWriteData	Έξοδος	32	Δεδομένα προς εγγραφή στη μνήμη δεδομένων
MemRead	Έξοδος	1	Σήμα ελέγχου που υποδεικνύει ανάγνωση μνήμης
MemWrite	Έξοδος	1	Σήμα ελέγχου που υποδεικνύει εγγραφή στη μνήμη

WriteBackData	Έξοδος	32	Δεδομένα που εγγράφονται σε καταχωρητές (για αποσφαλμάτωση)

Όπως και στην προηγούμενη άσκηση, το αρχείο verilog multicycle.v θα πρέπει να έχει μια παράμετρο για το INITIAL_PC. Τα παρακάτω βήματα θα σας καθοδηγήσουν στη διαδικασία δημιουργίας της μονάδας ελέγχου.

Datapath

Το πρώτο βήμα είναι να αρχικοποιήσετε το κύκλωμα διαδρομής δεδομένων που δημιουργήσατε στη προηγούμενη άσκηση. Όταν αρχικοποιήσετε το datapath σας, θα πρέπει να περάσετε την παράμετρο **INITIAL_PC** της μονάδας multicycle στη μονάδα datapath σας. Αυτή η παράμετρος υποδεικνύει την αρχική τιμή της εντολής που θα εκτελεστεί.

Συνδέστε τις ακόλουθες θύρες του datapath στην αντίστοιχη θύρα του multicycle module:

- PC
- instr
- dAddress
- dReadData
- dWriteData

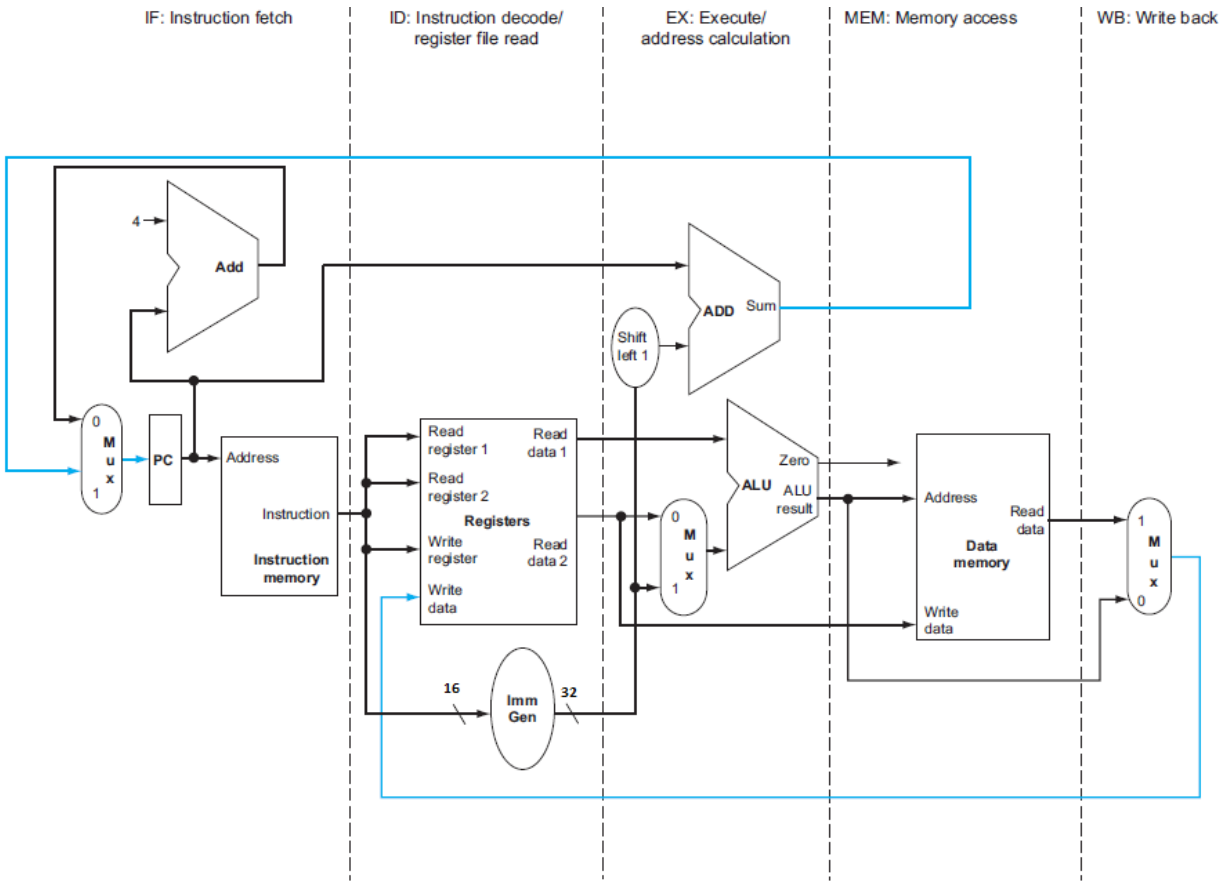
Η υπόλοιπη λογική εντός αυτής της μονάδας θα παράγει τα σήματα ελέγχου που ελέγχουν τη μονάδα datapath και θα περιγραφεί λεπτομερέστερα παρακάτω.

FSM

Το επόμενο βήμα για τη μονάδα ελέγχου είναι η δημιουργία της μηχανής πέντε καταστάσεων που διατρέχει με τη σειρά τα ακόλουθα πέντε βήματα εκτέλεσης μιας εντολής:

Κατάσταση	**Σκοπός**
IF	Instruction Fetch: Παροχή του PC στη μνήμη εντολών
ID	Instruction Decode: Αποκωδικοποίηση της ληφθείσας εντολής και έναρξη πρόσβασης στους καταχωρητές
EX	Execute: Εκτέλεση της λειτουργίας στην ALU
MEM	Memory: Εκτέλεση πρόσβασης στη μνήμη (για lw/sw)
WB	Write Back: Εγγραφή νέων δεδομένων στους καταχωρητές

Ο πρωταρχικός σκοπός αυτής της άσκησης είναι να υλοποιήσετε τη μηχανή κατάστασης και να δημιουργήσετε τα σήματα ελέγχου για τη μονάδα διαδρομής δεδομένων (το μπλε τμήμα του Σχήματος 7). Το Σχήμα 8 καθώς και τα παρακάτω τμήματα θα σας βοηθήσουν να κατανοήσετε τη λογική που χρειάζεστε για τη δημιουργία των σημάτων ελέγχου.



Σχ. 8: Διαχωρισμός του διαγράμματος του datapath σε κάθε μια από τις 5 καταστάσεις.

ALUCtrl

Το '**ALUCtrl**' είναι ένα σήμα ελέγχου τεσσάρων bit που υπαγορεύει τη λειτουργία της ALU (ALU Control του Σχήματος 7). Η παραγωγή του σήματος '**ALUCtrl**' απαιτεί αποκωδικοποίηση πολλαπλών επιπέδων που περιλαμβάνει το πεδίο '**opcode**' μιας εντολής (**instr**) καθώς και τα bit '**funct3**' και '**funct7**'. Η αποκωδικοποίηση θα πρέπει να ορίσει τη λειτουργία της ALU με βάση την εντολή ως εξής:

LW/SW: πρόσθεση

BEQ: αφαίρεση

Υπόλοιπες λειτουργίες ALU: λειτουργία με βάση τα funct3 και funct7 (συμπεριλαμβανομένων των εντολών μετατόπισης)

Παράξτε το σήμα '**ALUCtrl**' με βάση τα δεδομένα παραπάνω καθώς και με βάση της σελίδας 148 του pdf. Λάβετε υπόψη και τον πίνακα της Άσκησης 1.

ALUSrc

Όταν '**ALUSrc**'== 0, η έξοδος του καταχωρητή ανάγνωσης 2 χρησιμοποιείται ως είσοδος στον τελεστή 2 της ALU. Όταν '**ALUSrc**'== 1, άμεσα δεδομένα (immediate data) χρησιμοποιούνται για την είσοδο στον τελεστή 2 της ALU. Άμεσα δεδομένα απαιτούνται μόνο για τις εντολές 'load', 'store' και 'ALU Immediate'. Μπορείτε να προσδιορίσετε την τιμή του '**ALUSrc**' απευθείας από τα

bits του opcode της εντολής. Όπως και πριν χρησιμοποιήστε τα δεδομένα της σελίδας 148. Αυτό το σήμα δεν εξαρτάται από την κατάσταση του FSM.

MemRead και MemWrite

Δύο σήματα ελέγχουν τη διεπαφή μνήμης δεδομένων: '**MemRead**' και '**MemWrite**'. Το σήμα '**MemRead**' τίθεται για τις εντολές 'load' και το σήμα '**MemWrite**' για τις εντολές 'store'. Σε αντίθεση με τα προηγούμενα σήματα, τα σήματα αυτά πρέπει να τίθενται μόνο κατά τη διάρκεια του σταδίου '**MEM**' της μηχανής FSM.

MemtoReg και RegWrite

Το σήμα '**RegWrite**' χρησιμοποιείται για τον έλεγχο της εγγραφής τιμών στο αρχείο καταχωρητών. Για αυτή την ακολουθία εκτέλεσης πέντε κύκλων, το '**RegWrite**' πρέπει να τίθεται σε '1' μόνο κατά τη διάρκεια της κατάστασης '**WB**' της μηχανής κατάστασης εντολών. Ορισμένες εντολές δεν γράφουν πίσω στο αρχείο καταχωρητών, οπότε αυτό το σήμα θα παραμείνει χαμηλό για ολόκληρη την ακολουθία εντολών πέντε κύκλων ρολογιού.

Το σήμα '**MemtoReg**' βασίζεται στην τρέχουσα εντολή και τίθεται σε '1' μόνο κατά την εκτέλεση μιας εντολής 'load'. Αυτό το σήμα δεν εξαρτάται από την τρέχουσα κατάσταση (δεν έχει σημασία τι είναι αυτό το σήμα κατά τη διάρκεια όλων των καταστάσεων εκτός από την κατάσταση '**WB**').

loadPC και PCSrc

Το σήμα '**loadPC**' χρησιμοποιείται για την ενημέρωση του PC με μια νέα τιμή. Το σήμα '**loadPC**' πρέπει να τίθεται σε '1' κατά τη διάρκεια της κατάστασης '**WriteBack**' κάθε εντολής (φορτώνοντας έτσι μια νέα τιμή PC πριν από την είσοδο στην επόμενη κατάσταση **IF**).

Το σήμα '**PCSrc**' χρησιμοποιείται για να υποδείξει ποια τιμή πρέπει να φορτωθεί στον PC: είτε PC+4 για κανονικές εντολές είτε PC+branch_offset για πράξεις που λαμβάνονται με διακλάδωση. Το σήμα αυτό βασίζεται τόσο στην τρέχουσα εντολή όσο και στην τιμή του σήματος '**Zero**' (δείτε Σχήμα 7). Αυτό το σήμα πρέπει να τίθεται σε '1' όταν υπάρχουν οι ακόλουθες συνθήκες: (1) η τρέχουσα εντολή είναι μια λειτουργία "**BEQ**" και (2) το **Zero** είναι ίσο με '1'.

Testbench

Δημιουργήστε ένα testbench που θα ελέγχει όλες τις εντολές που υποστηρίζονται, όπως παρουσιάστηκαν στη προηγούμενη άσκηση και όπως φαίνονται και παρακάτω:

Register-Register: ADD, SUB, AND, OR, XOR, SLT, SLL, SRL, SRA

ALU Immediate: ADDI, ANDI, ORI, XORI, SLTI, SLLI, SRLI, SRAI

Μνήμη: LW, SW

Διακλάδωση: BEQ

Χρησιμοποιήστε τα αρχεία μνημών που σας δίνονται για τη μνήμη εντολών και για τη μνήμη δεδομένων, καθώς και το αρχείο με την αρχικοποίηση των εντολών (.data) και εκτελέστε προσομοιώσεις.

Παραδοτέα Ασκήσεων 1-5

Για τις παραπάνω ασκήσεις θα πρέπει να υποβάλετε μία αναφορά η οποία να περιέχει τα ακόλουθα και με την εξής σειρά:

1) Το αρχείο **alu.v** της άσκησης 1.
2) Τα αρχεία **calc.v, calc_enc.v, calc_tb.v** της άσκησης 2.
3) Το αρχείο **regfile.v** της άσκησης 3.
4) Το αρχείο **datapath.v** της άσκησης 4.
5) Τα αρχεία **top_proc.v** και **top_proc_tb.v** της άσκησης 5.

Στην αναφορά σας συμπεριλάβετε σχηματικό διάγραμμα από το FSM της Άσκησης 5 καθώς και κυματομορφές προσομοίωσης από τις Ασκήσεις 2 και 5. Επίσης προσθέστε σύντομες αναφορές στη διαδικασία που ακολουθήσατε για σχεδίαση και υλοποίηση κάθε module από κάθε άσκηση.

Η αναφορά πρέπει να υποβληθεί ως 1 αρχείο pdf όπου θα αναφέρεται το όνομά σας και το ΑΕΜ σας. Σε διαφορετική περίπτωση δε θα δίνεται κανένας βαθμός στην εργασία.

Βαθμολόγηση: Η εργασία είναι ατομική και υποχρεωτική, βαθμολογείται με άριστα το 30 και ισοδυναμεί με το 30% του συνολικού βαθμού του μαθήματος (δηλ. μέχρι 3 βαθμούς). Μη υποβολή ή εκπρόθεσμη υποβολή της εργασίας ισοδυναμεί με απώλεια τριών βαθμών στο συνολικό βαθμό. Επίσης, η βαθμολόγηση γίνεται με βάση τη συνολική εικόνα της αναφοράς.

Η προθεσμία υποβολής της εργασίας είναι: **21 Ιανουαρίου 2024 στις 23:59**, μόνο μέσω της πλατφόρμας elearning.

Αναφορές

ECEN 323: Computer Organization Home • ECEn 323: Computer Organization. Available at: https://ecen323wiki.groups.et.byu.net/ (Accessed: 12 November 2023).