

Ασκηση 8 [8.5]

a) Τιτανικας Αλυθειας Main Ctrl

OPCODE	aluSrc	rfSrc	mRd	mWr	rfWr	brOp	aluOp
0000011	1	1	1	0	1	0	000
0100011	1	x	0	1	0	0	000
1100011	0	x	0	0	0	1	001
0010011	1	0	0	0	1	0	010
0110011	0	0	0	0	1	0	110
xxxxx00	x	0	0	0	0	0	XXX

b) Τιτανικας Αλυθειας brctrl

brOp	zero	sign	funct3	PCSrc
0	x	x	xxx	0
1	1	0	0x0	1
1	0	x	0x0	0
1	0	1	1x0	1
1	x	0	1x0	0
1	0	x	0x1	1
1	1	x	0x1	0
1	x	0	1x1	1
1	x	1	1x1	0

Ασκηση 8.7 (Αναλυτική οι χρόνοι)

- a) t=500ps
- b) t=500+200=700ps
- c) t=500+300=800ps
- d) t=500+300+400=1200ps
- e) t₁=400ps, t₂=500+400+200=1100ps
- f) Αν εξουπερ το τρέις jump κτλ θα είναι
 $t = 500 + 300 + 400 = 1200ps$
 Διαρροετικά
 $t = 1300 ps$
- g) t=500+300+400+500=1700ps
- h) t=500+300+400+500=1700ps
- i) t=500+300+400+500+200=1900ps

Ασκηση 8.7

- a. 500 ps
- b. 700ps
- c. 800ps
- d. 1200ps
- e. 4000ps και 1100ps $\frac{PC+4}{PC+2}$ imm
- f. 1200ps i 1100ps $\Rightarrow f = 5.26316 \cdot 10^8 \cdot 10^{-6} \Rightarrow f = 5.26316 \cdot 10^2 \Rightarrow f = 526,316 MHz$
- g. 1700ps
- h. 1700ps
- i. 1900ps

Η χειρότερη (αργότερη) από τις λειτουργίες που έχουμε είναι η store (η αδιαλιπώντας), η οποία διαρκεί 1900ps. Ήσ εκ ταύτου ο κύριος των πολογιών για αυτόν των επεξεργαστών θα είναι διάρκεια 1.9 ns. Επομένως η συνάντηση των πολογιών για αυτόν των

ΕΠΕΞΕΡΓΑΣΤΩΝ θα είναι: $f = \frac{1}{T} \Rightarrow f = \frac{1 \cdot 10^9}{1.9} \Rightarrow f = 5.26316 \cdot 10^8$

AV ή ALU έχει καθυστέρηση 4400ps τότε η πιο αργή λειτουργία

Θα έχει διάρκεια 1900+4400=5900ps, δηλαδή $T = 5.9 ps$

$f = \frac{10^9}{5.9} \Rightarrow f = 1.69492 \cdot 10^8 \Rightarrow f = 1.69492 \cdot 10^2 \Rightarrow f = 168,492 MHz$

To παρόν έχει ξίνει περίπου 3000ps πιο αργό.

aluOp	function3	function7	aluMd
000	XXX	XXXXXXX	00X0
001	XXX	XXXXXXX	00X1
010	110	XXXXXXX	010X
010	111	XXXXXXX	011X
010	X0x	XXXXXXX	00X0
110	110	XXXXXXX	010X
110	111	XXXXXXX	011X
110	000	0100000	00X1
110	000	0000010	1XXX
110	000	X0XXXX0X	00X0
110	000	X1XXXX1X	00X0

Άριθμος 8.6

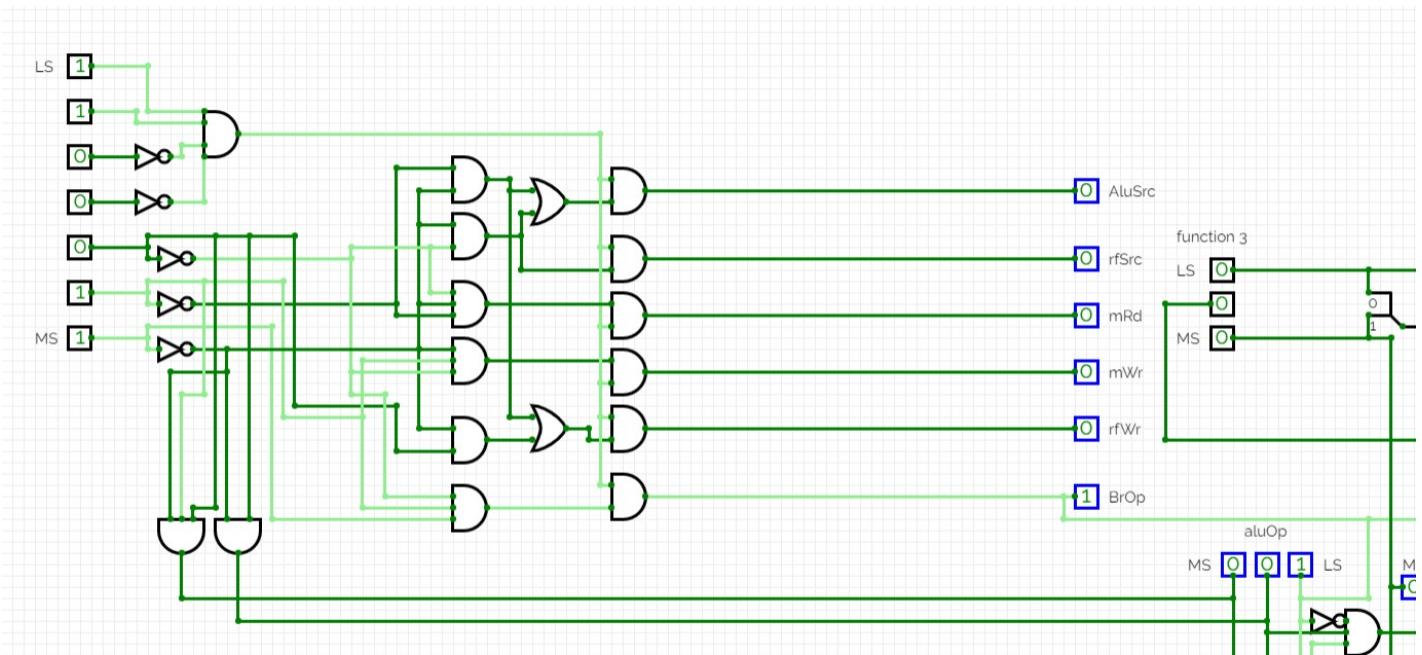
Παρατηρούμε ότι το PC+4 βρίσκεται στο "σύρτι" των αριθμητών κωντά
τον PC+4. Για να μεταφέρουμε το PC+4 στον καταχωριτή προαιρούμε, αρκει να
αυξήσουμε τον πολυπλέκτη των rfSrc από 1 bit σε 2 (εάν γίνεται επομένως
επιλογή rfSrc από 1 bit σε 2) και να συνδέσουμε το "σύρτι" PC+4 σε μια
θέση του πολυπλέκτη (εστιώντας 00 και 01 παίρνει το αποτέλεσμα από την ALU,
ο συνδυασμός 10 από το data και ο 11 το PC+4). Η διεύρυνση μνήμης στην
αυτή καινή σύρτι (μεταξύ μεταξύ εντολή jalr) προκύπτει ακριβώς με το ίδιο
τρόπο όπως η lw, αρκει τώρα να μη διαβάζουμε από την DMem και το
αποτέλεσμα να το σημάνει στο πολυπλέκτη pcSrc (ο οποίος θα εγγυηθεί
από 1 bit σε 2bit). Η εντολή jalr πρέπει να ακινητοποιεί το brOp σημάνοντας
το οποίο με την βοήθεια του funct3 πρέπει να καταταβαίνει ότι είναι
jalr εντολή και να στρέψει πάντα το πολυπλέκτη στη θέση του απότελεσμάτων
της ALU, ενώ παρατητώντας rfSrc να σημάνει το PC+4 στον rd. Πλέον θα
είναι εντελείται και η jal, η οποία έχει στην brOp δεν έχει "τοποθετεί" το
πολυπλέκτη στη θέση του απότελεσμάτων της ALU, αλλιώς θα ήταν $x_2 + PC$.

Ασκηση 8.6 (Συνέχεια)

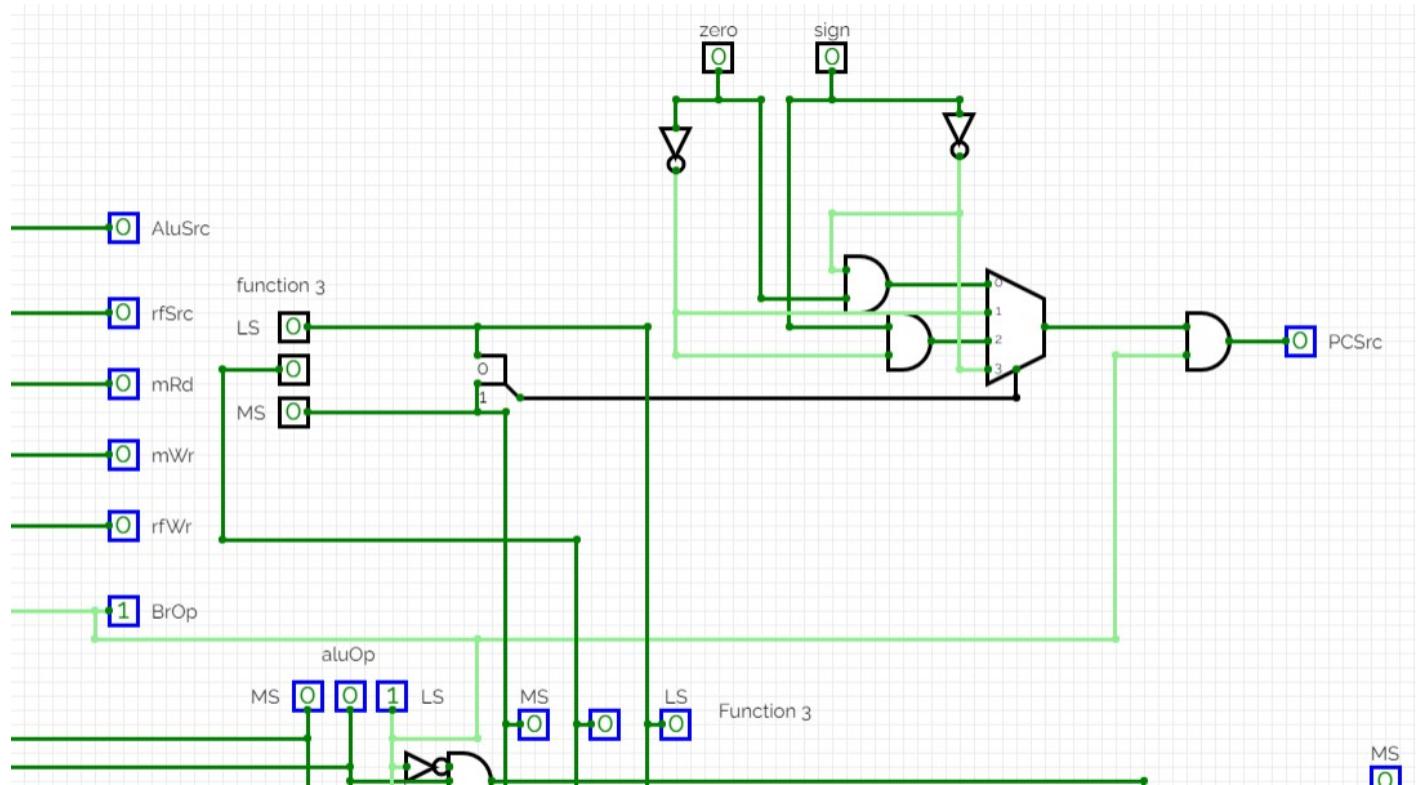
Αναδυτικότερα όταν έχουμε την εντολή j l τότε το σήμα rfSrc πρέπει να επιλέγεται το PC+4 (Θέση 11). Ο pcSrc πρέπει να επιλέγεται το $PC + 2 \times 1mm$. Τέλος πρέπει το σήμα rfWr να είναι 1 ώστε να γράφεται το PC+4 από την rd ματακωρύχη.

Όταν έχουμε $jalr$ πρέπει το pcSrc να επιλέγεται το αποτέλεσμα της ALU, ενώ το rfSrc να επιλέγεται το PC+4 και αρχικά $rfWr = 1$ να εφραγθεί στο αρχείο ματακωρύχων (το PC+4). Τέλος η aluSrc=1 πρέπει να επιλέγεται το 1mm και να έχει προσθέτην signed (με την κατάλληλη aluMd),

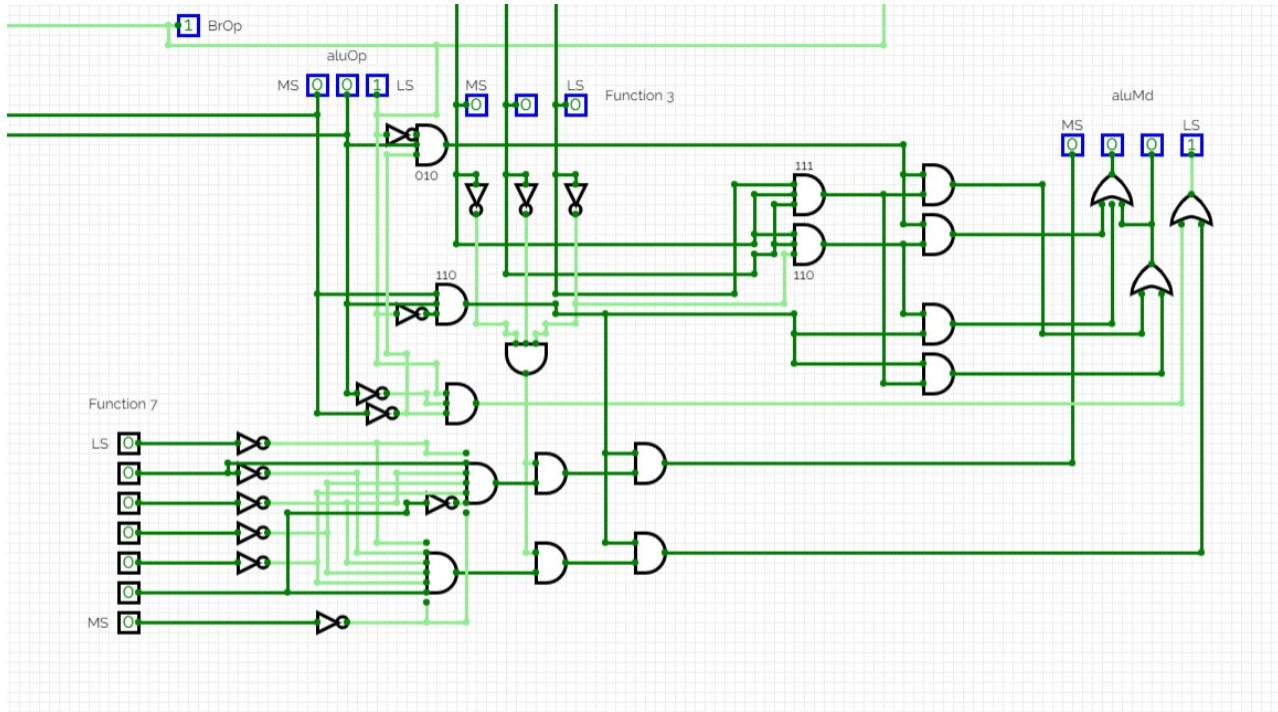
Κύκλωμα Main Ctrl



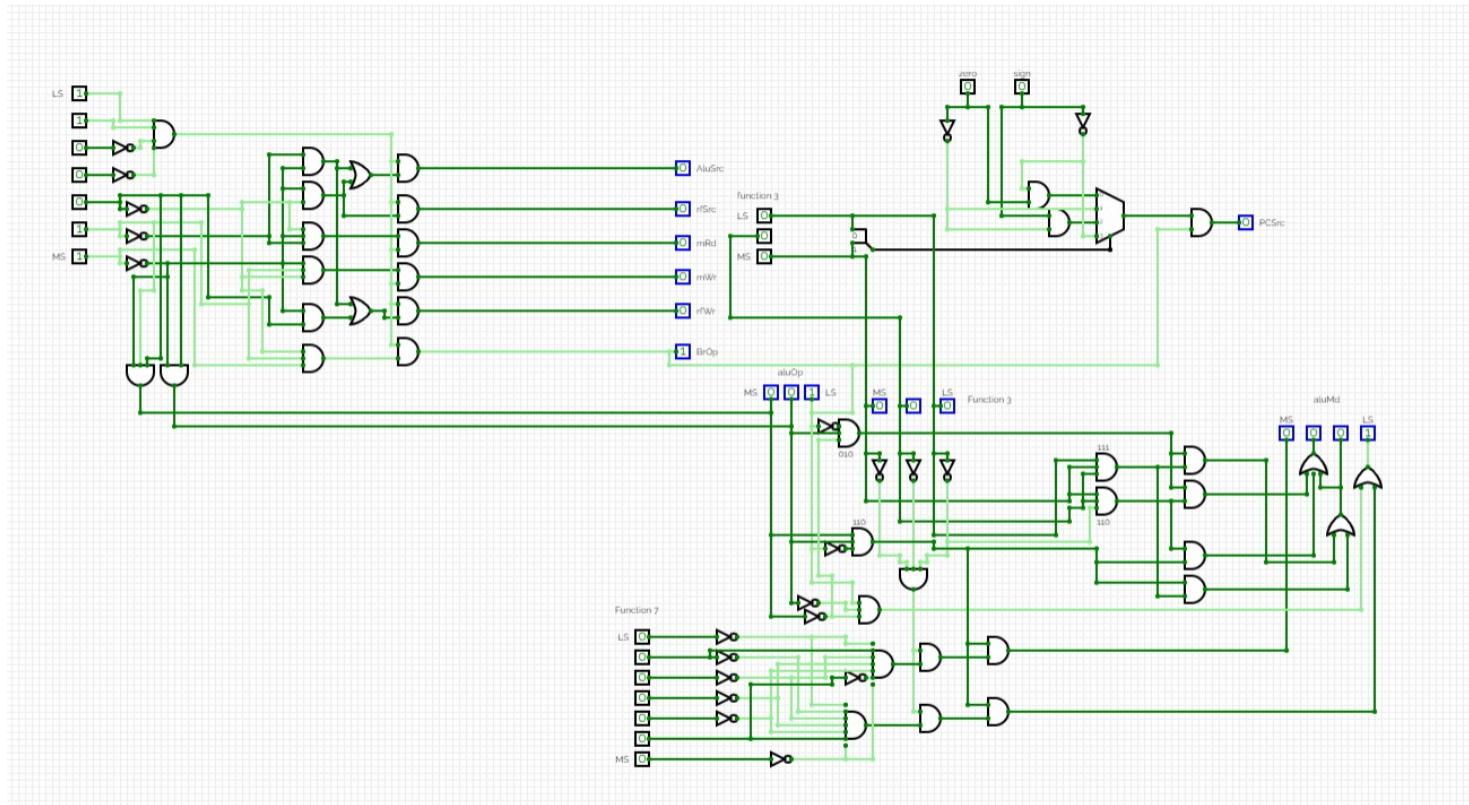
Κύκλωμα brctrl



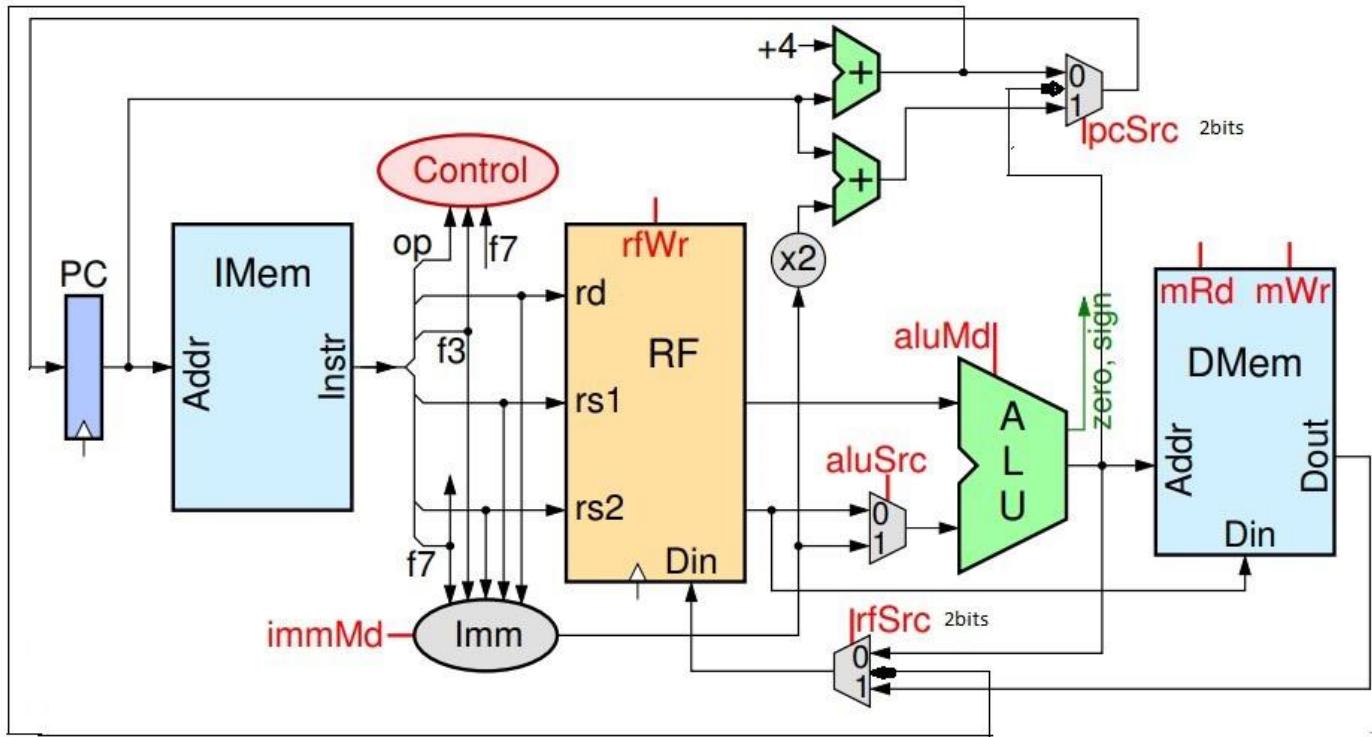
Κύκλωμα aluctrl



Ολόκληρο το κύκλωμα



8.6) Αλλαγές datapath για υποστήριξη εντολών jal και jalr



1) Τα στάδια της pipeline του επεξεργαστή είναι 5.

Instruction fetch) Το πρώτο στάδιο είναι στην Instruction Memory στον μέσω το PC (καταχώριση). Επιλέγεται η εντολή και από αυτή προκύπτει να εκτελεσθεί (παρατήτης παλογίζεται και $PC + 4$)

Instruction decode) Στο δεύτερο στάδιο, το οποίο βρίσκεται αρέσκια μετά το IMem (δηλαδή στο decoder και στους Registers), γίνεται η αποδικτικότητα της εντολής από decoder και παρατήτης διαβάζεται από $rs1, rs2, rd$ και λιγότερος αν χρησιμοποιούνται οι οξι.

Execute iii) Στο τρίτο στάδιο γίνονται οι πράξεις μεταξύ register και μεταξύ register και σταθερών αριθμητών στην ALU. Βρίσκεται στην ALU το στάδιο αυτό, και αναδοτική σε αυτό το οποίο γίνονται οι "υπολογισμοί".

Memory Access iv) Στο τέταρτο στάδιο, εκτελούνται: κυρίως εντολές load και store, αλλα στο στάδιο αυτό διαβάζονται τα δεδομένα (της διεύθυνσης που μας δίνεται) από την DMem και γεμίζονται σε αυτή την διεύθυνση ακόλουθα (store).

Write back v) Στο πέμπτο στάδιο, αποθηκεύονται (αν χρειάζεται πάντα) τα αποτελέσματα στο καταχώριστη rd . Απορρίζεται πολλά δεδομένα θα χρειάζονται (αν χρειάζεται εφερεψη) στην RF (rd αυτοκεντρικό). Τα δεδομένα μπορεί να είναι είτε αποτέλεσμα μιας αριθμητικής πράξης στην ALU ή αποτέλεσμα από την DMem (Τ.Ι. load).

2). Οι βαθμίδες οι οποίες είναι απαραίτητες για όλες τις εντολές είναι i), ii), iii)

Στην i) βαθμίδα υπολογίζεται η διεύθυνση της εντολής που προκύπτει και εκτελεσθεί και έπειτα στην ii) βαθμίδα αποδικτικούται το opcode της εντολής είναι παραπάντα διαβάζεται το λιμν, η διεύθυνση $rs1, rs2$ και rd , ανεξαρτήτως αν τα χρησιμοποιούνται ή όχι. Τέλος, η τελευταία απαραίτητη βαθμίδα για όλες τις εντολές είναι η 'ALU' (iii) βαθμίδα), τόσο για υπολογισμό αποτελεσμάτων πράξην, διεύθυνσεων κτλ.

3) Η βαθμίδα που δεν μπει κατί χρήση για τις εντολές αριθμητικών πράξην είναι η iv). Στις αριθμητικές πράξεις το αποτέλεσμα της ALU δεν εκφράζεται διεύθυνση της DMem και επομένως δεν προσφέρει κατί το 4ο στάδιο (εκτός από το ότι πρέπει τα αριθμ. we και rd να είναι σύνορα για να μη διαβιβασθεί η γεμίζοντας παίκτη). Άλλο την άλλη πλευρά, η βαθμίδα που δεν μπει κατί χρήση για τις εντολές store είναι η v). Στις εντολές store αποτελείται η αποθήκευση των δεδομένων ενας register στην DMem. Επομένως σταν φτάνουν στην 4η βαθμίδα εχουμε "απολατική τελειωση της αποτί πας". Πλα' όλα αυτά το στάδιο \Rightarrow πρέπει να εξασφαλίσει ότι δεν θα γίνει κατόπιν εγγερψη από RF κατά την διάρκεια της store.

4)

a) .text

register to register

sub x11, x2, x4

add x13, x14, x15

add x17, x20, x21

b) # register and immediate

addi x10, x0, 23

andi x12, x16, 0

addi x25, x26, -67

c) # load and store commands

sw x18, 0(x3)

lw x19 8(x15)

5)

.text

register to register (dependent)

sub x11, x2, x4

add x15, x11, x14

add x17, x11, x15

6) Before Instruction Scheduling

.data

ints:

.word 5

.word 455

.word -6

.word 23

.word 7

.text

la s0,ints #load address

d=a+b;

lw x10, 0(s0) #a

lw x11, 4(s0) #b

add x10, x10, x11

sw x10, 12(s0) #d

e=a+c;

lw x10, 0(s0) #a

lw x11, 8(s0) #c

add x10, x10, x11

sw x10, 16(s0) #e

Συνέχεια δίπλα →

6) After Instruction Scheduling

.data

ints:

- word 5
- word 455
- word -6
- word 23
- word 7

.text

la \$0,ints #load address

first we load all our int variables

lw x10,0(\$0)#a

lw x11,4(\$0)#b

lw x12,8(\$0)#c

d=a+b;

add x13,x10,x11

sw x13,12(\$0)#d

e=a+c;

add x13,x10,x12

sw x13,16(\$0)#e

Y.R.

Αντι για την εγκώδι "word k", οποια δεσμεύει 1 word και τοποθετεί το k ws πλειστόπερα του, μπορούμε να κρινούμε ποια bytes της εγκώδης είναι zero μ (όπου μ αριθμός bytes της δεσμεύει να δεσμεύεται), ή ότια δεσμεύει μ bytes και τα αρχικάταξαι με 0. Αν επιφέρεις χρησιμότητα το .zero το "πρόγραμμα" θα είχε την έγκινη μορφή

.data

ints:

- zero 20

.text

la \$0,ints

addi x5,x0,5

sw x5,0(\$0)

addi x5,x0,455

sw x5,4(\$0)

addi x5,x0,-6

sw x5,8(\$0)

addi x5,x0,23

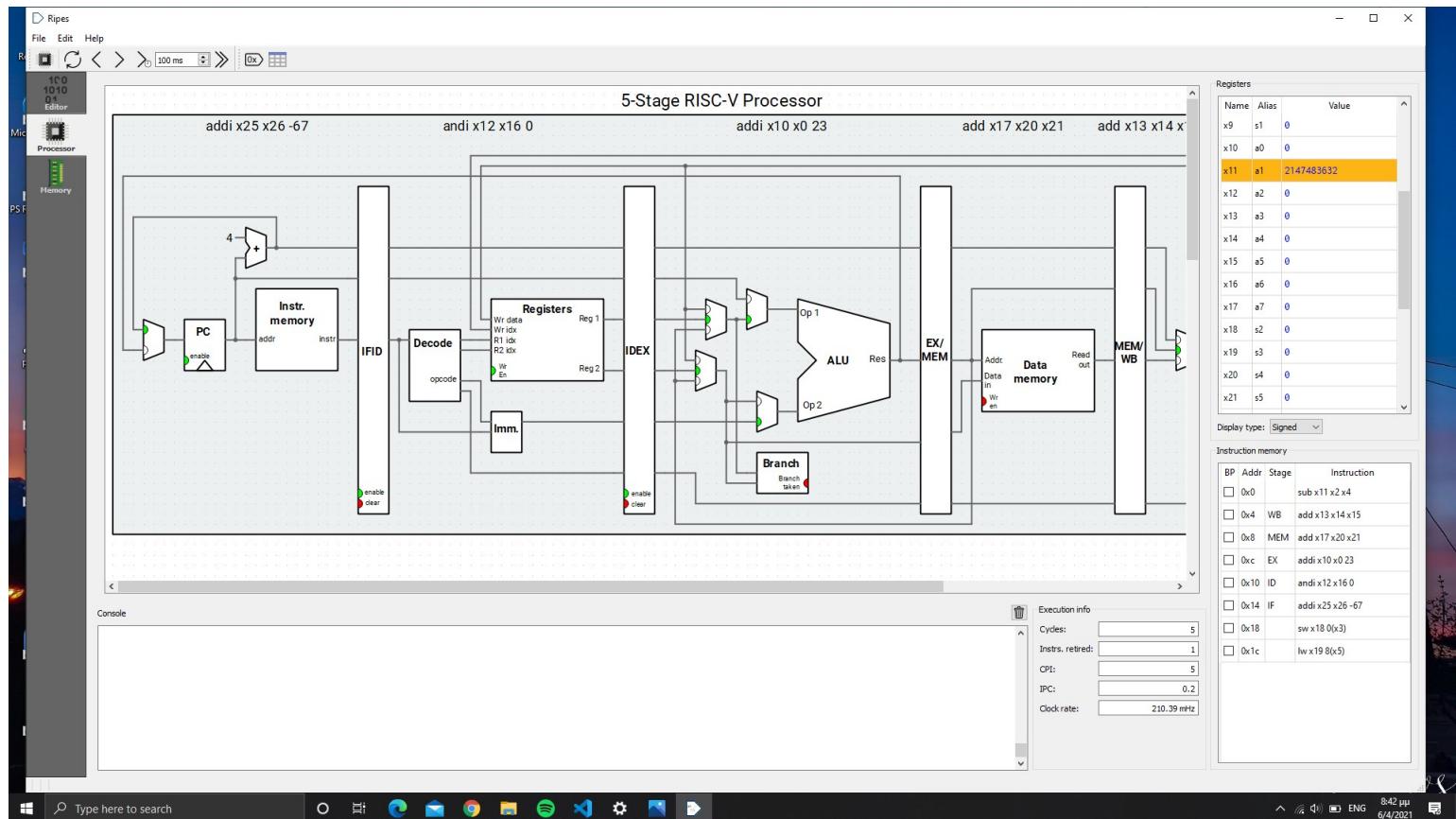
sw x5,12(\$0)

addi x5,x0,7

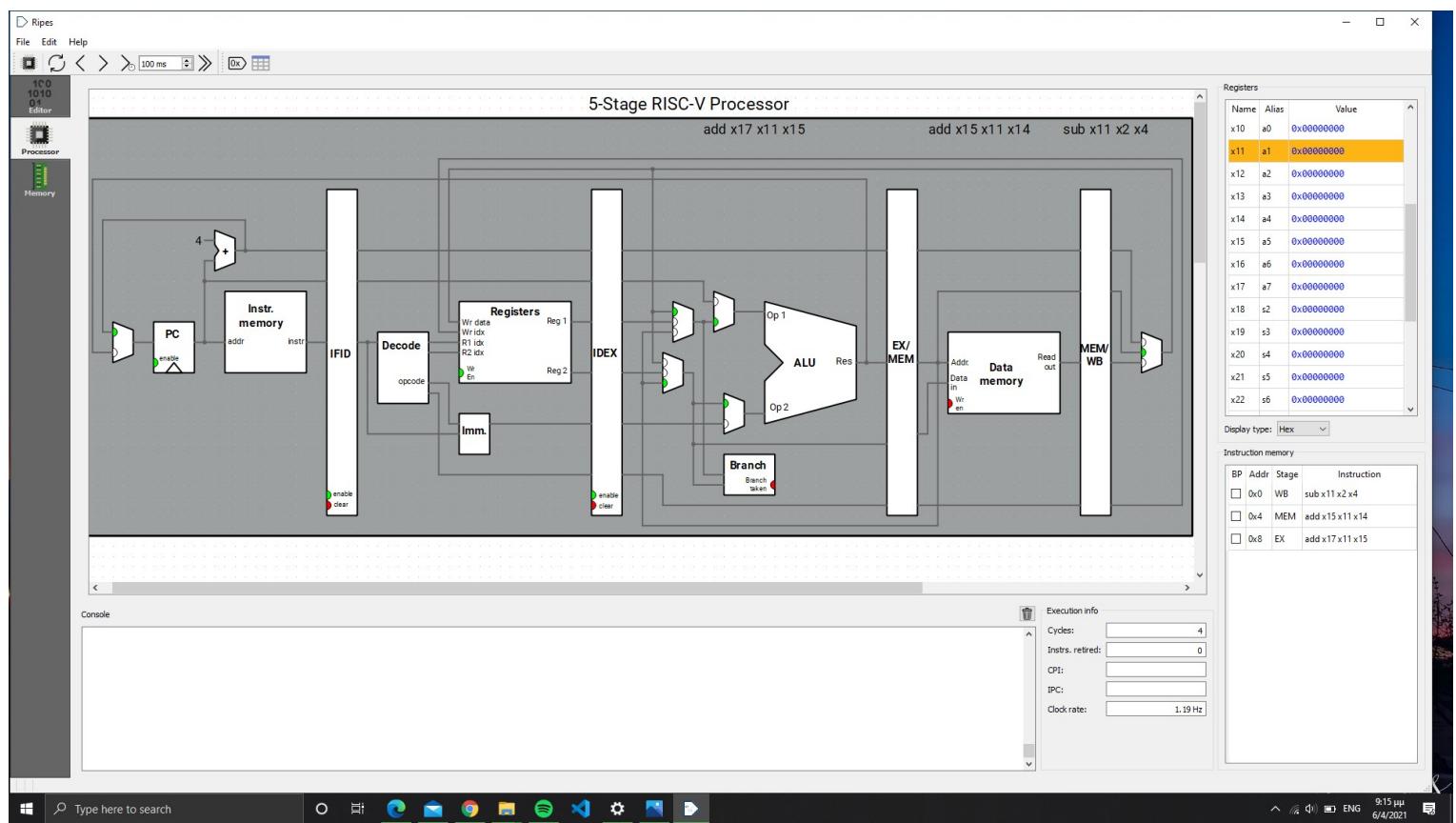
sw x5,16(\$0)

Προκατέβουν να απαρίγων αυτή την διαδικασία, προτίμως να κρινούμε ποια bytes της εγκώδης .word

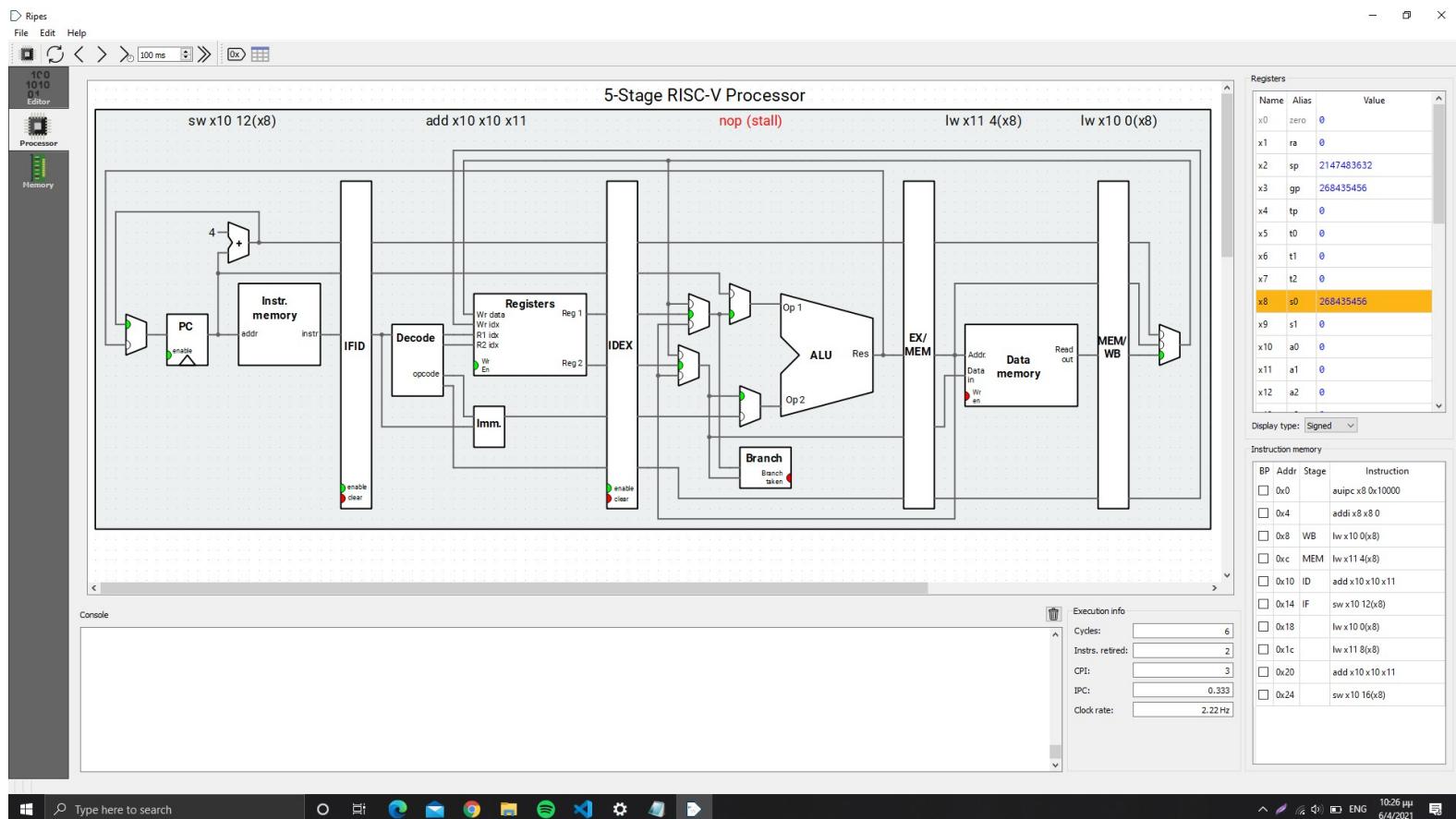
9.9.4



9.9.5



Before instruction scheduling



After instruction scheduling

