

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ Οργάνωση Υπολογιστών

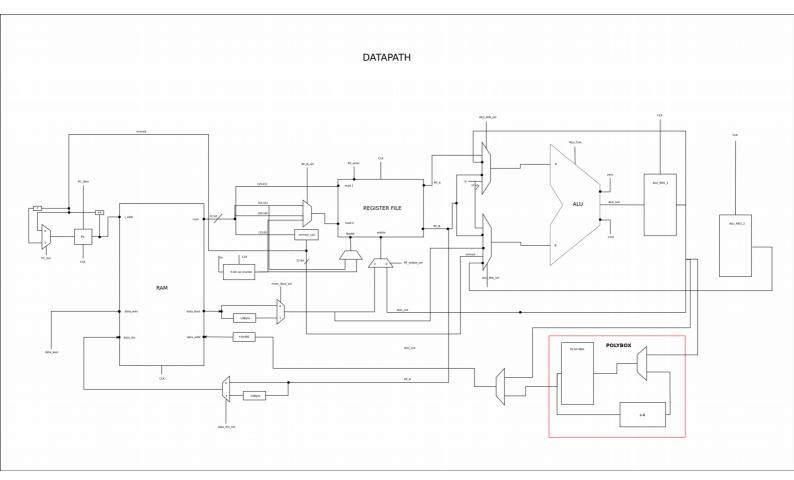
Ομάδα: LAB31235453

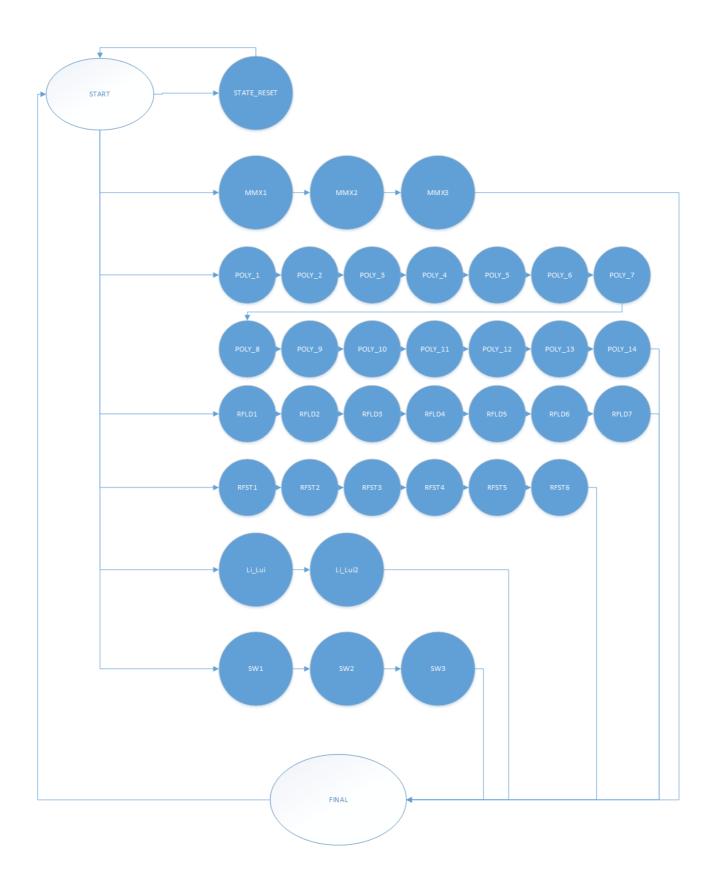
Μπαλαμπάνης Ηλίας 2014030127 Μποκαλίδης Αναστάσιος 2014030069

Αναφορά Εργαστηρίου 4

Περιγραφή Άσκησης

Μας ζητήθηκε να υλοποιήσουμε επιπλέον 4 custom εντολές. Παρακάτω παρατίθενται τα σχήματα του datapath και της FSM. Στην παρακάτω FSM απεικονίζουμε μόνο τις εντολές που χρησιμοποιήθηκαν στο εργαστήριο 4.





Εντολές:

- 1) addi MMX byte: Σε αυτήν την εντολή θέλουμε να προσθέσουμε κάθε byte του καταχωρητή που έρχεται από το instruction με το immed που παράγεται. Έτσι επιλέγουμε από τους πολυπλέκτες πριν την ALU να πάρουμε την τιμή του καταχωρητή που έρχεται από την addr1 και το Immed. Έπειτα προσθέσαμε μια νέα λειτουργία μέσα στην ALU ώστε να κάνει τους κατάλληλους υπολογισμούς. Όταν τελειώσει αυτή η λειτουργία, το τελικό αποτέλεσμα το αποθηκεύουμε σε ένα καταχωρητή στην έξοδο της ALU και μετά τον γράφουμε στον επιθυμητό καταχωρητή.
- 2) **POLY 2 :** Σε αυτήν την εντολή θέλουμε να υπολογίσουμε ένα πολυώνυμο $2^{\circ\circ}$ βαθμού. Αρχικά υπολογίζουμε την διεύθυνση που θα πάρουμε τον συντελεστή α από την μνήμη. Μετά υπολογίζουμε Το Χ² περνώντας μέσα στην ALU την ίδια τιμή από τους καταχωρητές. Τονίζουμε ότι έχουμε βάλει μέσα στην ALU την πράξη του πολ/σμου. Αφού υπολογίσουμε το X^2 το αποθηκεύουμε σε ένα καταχωρητή και μετά παίρνοντας την τιμή του υπολογίζουμε το αΧ². Την τιμή αυτήν την αποθηκεύουμε σε ένα καταχωρητή και αυξάνουμε την διεύθυνση της μνήμης κατά 4 έτσι ώστε να πάρουμε τον συντελεστή β. Ακολουθεί ο υπολογισμός του βΧ . Βάζουμε στην ΑLU τον συντελεστή β και παίρνουμε από τους καταχωρητές της Register File το Χ. Υπολογίζουμε το βΧ και το αποθηκεύουμε σε διαφορετικό καταχωρητή από αυτόν που είχαμε το αX^2 . Έπειτα αυξάνουμε και πάλι την διεύθυνση της μνήμης για να ανακτήσουμε το συντελεστή γ. Όταν το κάνουμε περνάμε μέσα στην ALU το αX² και τον συντελεστή γ και τα προσθέτουμε. Το αποτέλεσμα το αποθηκεύουμε στον καταχωρητή που είχαμε πριν το αX^2 . Τέλος παίρνουμε από τους 2 καταχωρητές το αX² + γ και το βX, τα περνάμε στην ALU, τα προσθέτουμε και το τελικό αποτέλεσμα το γράφουμε στον καταχωρητή που μας ορίζει το Instruction.
- 3) **RFLD**: Σε αυτήν την εντολή θέλουμε να αποθηκεύσουμε σε όλους τους καταχωρητές της Register File 31 συνεχόμενα στοιχεία από τη μνήμη. Αρχικά περνώντας στη ALU από τους επιθυμητούς καταχωρητές τις τιμές τους , υπολογίζουμε την πρώτη διεύθυνση στη μνήμη όπου βρίσκεται το πρώτο στοιχείο που θέλουμε να φορτώσουμε στον πρώτο καταχωρητή. Με το που υπολογίσουμε την διεύθυνση της μνήμης που θέλουμε , ενεργοποιούμε ένα μετρητή ο οποίος μετράει μέχρι το 31 και η έξοδος του μας δίνει τον καταχωρητή που θέλουμε να γράψουμε το αποτέλεσμα που φορτώσαμε από την μνήμη. Έτσι η διαδικασία έχει ως εξής. Διαβάζουμε από την μνήμη -> γράφουμε στον 1° καταχωρητή (ξεκινάμε από τον 0 όπου δεν μπορούμε να γράψουμε σε αυτόν) -> αυξάνουμε την διεύθυνση της μνήμης κατά 4 -> αυξάνουμε τον μετρητή κατά 1 θέση -> γράφουμε την νέα τιμή στον νέο καταχωρητή. Η διαδικασία αυτή συνεχίζεται μέχρι ο μετρητής να φτάσει στη θέση 31 και να μας δώσει RC='1'.

4) **RFST**: Σε αυτήν την εντολή θέλουμε να περάσουμε τα στοιχεία από τους καταχωρητές της Register File, σε συνεχόμενες θέσεις στη μνήμη RAM. Αρχικά υπολογίζουμε με παρόμοιο τρόπο με την προηγούμενη εντολή την 1^η διεύθυνση στη μνήμη που θέλουμε να γράψουμε. Έπειτα περνάμε από κάθε καταχωρητή την τιμή του σε μια νέα θέση στη μνήμη. Αυτό επιτυχάνεται με την αύξηση ενός μετρητή στη είσοδο που θέλουμε να διαβάζουμε τον νέο καταχωρητή και την περιοδική αύξηση της διεύθυνσης της μνήμης κατά 4.

Κυμματομορφές

Δίνοντας έμφαση στις 4 εντολές και όχι όσες υλοποιήθηκαν σε προηγούμενο και με βάση το πρόγραμμα που μας δόθηκε έχουμε:

MMX_addi_byte \$17, \$5, 1 -- \$17 = x"11110101" Name Value 1,140 ns la clock test_processor/clock 10000 ps 1281 out[31:0] instr[31:0] 11000100101100010000000000 alu_out_sig[31:0] 269484032 286327041 4112 269484032 33752069 alu_reg1_out[31:0] 33690130 alu_reg2_out[31:0] poly_out[31:0] 33690130 polymuxout[31:0] [0,10,11,12,13,2694840 32,4112,269488144,252641280,3855,252645135,1347420.. sig in mux32[0:31] [0, 10, 11, 12, 13, 269484032, 4112, 269 [0,10,11,12,13,269 ram[2047:0] [00 📑 alu_out[31:0] 286327041 🕌 rf_a[31:0] 👬 rf_b[31:0] 286327041 ddout[31:0] 📑 rf_b_sel[1:0] 01 17 ard2[4:0] 286327041 dout2[31:0] 33752069 din[31:0] output[4:0]

 $MMX_addi_byte $18, $6, 2 -- $18 = x"02021212"$

Name	Value		1,190 ns	1,200 ns	1,210 ns	1,220 ns	1,230 ns
To reset	0						
clock /test_processo	or/reset						
Clock_period	10000 ps						
alu_out[10:0]	261			1088		X	
pc_out[31:0]	108			100			*
instr[31:0]	110001001111001100000000000000000000000		*	11000100	1101001000000000	00000010	
튢 current_state	final		start	mmx1	mmx2	mmx3	final
Ilianout_sig[31:0]	269488144	<u> </u>	4112	*	33690130		4112
▶ 📑 a[31:0]	269488144		K		4112		
▶ 📑 b[31:0]	Θ		0	*	2		*
• alu_reg1_out[31:0]	336860180			286327041		X	
• alu_reg2_out[31:0]	Θ						
poly_out[31:0]	Θ						
olymuxout[31:0]	336860180			286327041		X	
sig_in_mux32[0:31]	[0,10,11,12,13,269484032,4112,269	[0,	0,11,12,13,2694840	32,4112,269488144,	252641280,3855,252	645135,1347420	[0,10,11,12,13,2
Tam[2047:0]	[00000000000000000000000000000000000000	[00	000000000000000000000000000000000000000	000000000000000000000000000000000000000	00000000000000000	000,00000000000	00000000000000
• I alu_out[31:0]	269488144		4112	*	33690130		4112
If_a[31:0]	269488144		K		4112		
▶ 📑 rf_b[31:0]	Θ			0		33690130	*
• ddout[31:0]	Θ						
rf_b_sel[1:0]	99		00	*	01		*
ard2[4:0]	Θ		0	*	18		*
▶ 📑 dout2[31:0]	Θ			0		33690130	*
▶ 📑 din[31:0]	336860180			286327041		X	
output[4:0]	0						

MMX_addi_byte \$19, \$7, 4 -- \$19 = x"14141414"

om In	Value		1,240 ns	1,250 ns	1,260 ns	1,270 ns	1,280 ns
To reset	0	┞					
∏ _o clock	1						
Clock_period	10000 ps						
▶ 🔜 alu_out[10:0]	4			132		k	
pc_out[31:0]	112			104			x
instr[31:0]	110001011011010000000000000000000000000		×	11000100	1111001100000000	00000100	
torrent_state	final		start	mmx1	mmx2	mmx3	final
▶ 🌄 alu_out_sig[31:0]	134744072		269488144		336860180		269488144
▶ 📑 a[31:0]	134744072		*		269488144		
▶ 📑 b[31:0]	Θ		0	k	4		*
Main alu_reg1_out[31:0]	269488144			33690130		*	
 Malu_reg2_out[31:0] 	Θ						
▶ 📆 poly_out[31:0]	Θ						
b olymuxout[31:0]	269488144			33690130		*	
sig_in_mux32[0:31]	[0,10,11,12,13,269484032,4112,269	[0,:	0,11,12,13,2694840	32,4112,269488144,	252641280,3855,252	645135,1347420	[0,10,11,12,13,2
▶ 🔣 ram[2047:0]	[00000000000000000000000000000000000000	[00	000000000000000000000000000000000000000	000000000000000000000000000000000000000	00000000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000
▶ 📑 alu_out[31:0]	134744072		269488144		336860180		269488144
▶ 📑 rf_a[31:0]	134744072		×		269488144		
▶ 📑 rf_b[31:0]	Θ			0		336860180	*
ddout[31:0]	3758161923						0
▶ 📑 rf_b_sel[1:0]	99		00		01		X
ard2[4:0]	0		0	(19		X
▶ 📑 dout2[31:0]	Θ			0		336860180	*
▶ 📑 din[31:0]	269488144			33690130		X	
output[4:0]	0						

 $MMX_addi_byte $20, $13, 8 -- $20 = x"10101010"$

Name	Value	I	1,290 ns	1,300 ns	1,310 ns	1,320 ns	1,330 ns
∏o reset	0						
U _G clock	1						
clock_period	10000 ps						
alu_out[10:0]	261			261			
pc_out[31:0]	116			108			*
▶ 📑 instr[31:0]	110001100001010100000000000010000		*	11000101	1011010000000000	00001000	
튢 current_state	final		start	mmx1	mmx2	mmx3	final
alu_out_sig[31:0]	33752069		134744072		269488144		134744072
▶ 📑 a[31:0]	33752069		X		134744072		
▶ 📑 b[31:0]	Θ		0		8		*
alu_reg1_out[31:0]	303240213			336860180			
alu_reg2_out[31:0]	Θ						
 Boly_out[31:0] 	Θ						
Bolymuxout[31:0]	303240213			336860180			
sig_in_mux32[0:31]	[0,10,11,12,13,269484032,4112,2694	[0,	10,11,12,13,2694840	32,4112,269488144,	52641280,3855,252	645135,1347420	[0,10,11,12,13,269
ram[2047:0]	[00000000000000000000000000000000000000	[00	000000000000000000000000000000000000000	000000000000000000000000000000000000000	00000000000000000	000,0000000000000	00000000000000000
alu_out[31:0]	33752069		134744072		269488144		134744072
▶ 📑 rf_a[31:0]	33752069		X		134744072		
▶ ■ rf_b[31:0]	Θ			0		269488144	*
• ddout[31:0]	Θ			0			*
rf_b_sel[1:0]	99		00		01		*
▶ 📑 ard2[4:0]	Θ		0		20		*
▶ 📑 dout2[31:0]	Θ			0		269488144	*
din[31:0]	303240213			336860180			
output[4:0]	0						

MMX_addi_byte \$21, \$16, 16 -- \$21 = x"12131415"

Name	Value	1	l,340 ns	1,350 ns	1,360 ns	1,370 ns	1,380 ns
∏ _e reset	0						
To clock	1						
Clock_period	10000 ps						
▶ 📆 alu_out[10:0]	1044			4		*	
• 🎇 pc_out[31:0]	120			11	2		*
instr[31:0]	10000010110101110000100000010000	LΞX		110001	10000101010000000	000010000	
튢 current_state	start	l⊐x	start	mmxl	mmx2	mmx3	final
· 📸 alu_out_sig[31:0]	90	l≕k	33752069	X	303240213		33752069
a[31:0]	80	l≕x			33752069		
b[31:0]	10		0	x	16		*
- 🌉 alu_reg1_out[31:0]	80			269488144		*	3
- 🌉 alu_reg2_out[31:0]	е						
poly_out[31:0]	е						
polymuxout[31:0]	80			269488144		*	3
sig_in_mux32[0:31]	[0,10,11,12,13,269484032,4112,2694	[0,10	0,11,12,13,2694840	32,4112,26948814	4,252641280,3855,2	52645135,1347420	[0,10,11,12,13
🥳 ram[2047:0]	[00000000000000000000000000000000000000	[000]	000000000000000000000000000000000000000	0,00000000000000	000000000000000000000000000000000000000	00,00000000000000	000000000000000
📑 alu_out[31:0]	90	l≕k	33752069	X	303240213		33752069
📑 rf_a[31:0]	80	l≕k			33752069		
■ rf_b[31:0]	10			0		303240213	*
ddout[31:0]	1			375816	1923		*
rf_b_sel[1:0]	00		00	X	01		*
ard2[4:0]	1		0	X	21		*
dout2[31:0]	10			0		303240213	*
din[31:0]	80			269488144		*	3
output[4:0]	е						

poly2 \$23, \$22, \$1 -- \$23 = MEM[1024+\$22]*\$1*\$1 + MEM[1024+\$22+4]*\$1 + MEM[1024+\$22+8] = 1*10*10+2*10+3 = 123

Name	Value	II.	1,430 ns	1,440 ns	1,450 ns	1,460 ns	1,470 ns	1,480 ns	1,490 ns	1,500 ns	1,510 ns	1,520 ns
la reset	8											
To clock	1											
la clock_period	10000 ps							10000 ps				
alu_out[10:0]	1846				1044				k	1045		1
pc_out[31:0]	120							120				
instr(31:0)	100000101101011110000100000010000	· I=k					100000101	1010111000010000	010000			
turrent_state	poly9		start	polyl	poly2	poly3	poly4	poly5	poly6	poly7	poly8	poly9
alu_out_sig[31:0]	20	- lo-x	90	x	80		x	100		200	2	0
a[31:0]	10	o k		1	30		1	0	1	00	1	0
- ■ b[31:0]	2	Iox	10	X	0		1	D	1	X	2	
alu_reg1_out(31:0)	100				80				k		100	
du_reg2_out(31:0)	80			0		K			80			
poly_out[31:0]	88			0			8	0	k	84		
polymuxout[31:0]	88				80				k	84		
sig_in_mux32[0:31]	[0,10,11,12,13,269484032,4112,26	94	[0,10,11,12,1	3,269484032,4112,2	9488144,252641280	,3855,252645135,13	4742016,2056,13474	4072,33751040,102	9,33752069,2863270	41,33690130,33686	180,269488144,303	40213,80,0,0,0,0
am[2047:0]	[00000000000000000000000000000000000000	000]	000000000000000000000000000000000000000	00,0000000000000	00000000000000000	000,0000000000	00000000000000000	0000,00000000,0000	000000000000000000	000000000000,00000	000000000000000000000000000000000000000	000000000000000
alu_out(31:0)	20	lo x	90	X	80		X	100		200	2	0
of_a[31:0]	80	le k						80				
rf_b(31:0)	10	lo k						10				
ddout[31:0]	2					1				X	2	
f_b_sel[1:0]	60							00				
ard2[4:0]	1	lo k						1				
dout2[31:0]	10	I⊡x						10				
din[31:0]	100				80				k		100	
- 📑 output[4:0]	0							0				
		Т										

Name	Value	1,	520 ns	1,530 ns	1,540 ns	1,550 ns	1,560 ns	1,570 ns	1,580 ns
la reset	9	┨┵┼							
To clock	1								
Clock_period	10000 ps							10000 ps	
▶ 🔣 alu_out[10:0]	1054				10	46			k
pc_out[31:0]	124				120				k
▶ 📑 instr[31:0]	011111000000010000000000000000000000000				1000001011	0101110000100000	10000		
turrent_state	sw2		poly9	poly10	poly11	poly12	poly13	poly14	final
alu_out_sig[31:0]	0		20	10	13	1:	23	143	80
▶ 📑 a[31:0]	0		10	10	10	10	03	123	80
▶ 📑 b[31:0]	0		2			K	20		0
• Image: Image	123			100		10	3	*	
• If alu_reg2_out[31:0]	20		80					20	
poly_out[31:0]	88							88	
polymuxout[31:0]	123				8	8			K
sig_in_mux32[0:31]	[0, 10, 11, 12, 13, 269484032, 4112, 269-	[0,10,	11,12,13,2694840	32,4112,269488144,	252641280,3855,252	645135,134742016,2	056,134744072,337	51040,1029,3375	[0,10,11,12,13,269
Tam[2047:0]	[00000000000000000000000000000000000000	[0000	0000000000000000	00000000000000000000	000000000000000000	000,000000000000	00000000000000000	0000,00000000,0000	000000000000000000
• I alu_out[31:0]	0		20	10	13	1:	23	143	80
If_a[31:0]	0					80			
rf_b[31:0]	11					10			
• ddout[31:0]	0		2	K					
rf_b_sel[1:0]	01					00			
▶ 📑 ard2[4:0]	2					1			
dout2[31:0]	11					10			
▶ 📑 din[31:0]	123			100		10	13	*	
• output[4:0]	0							0	

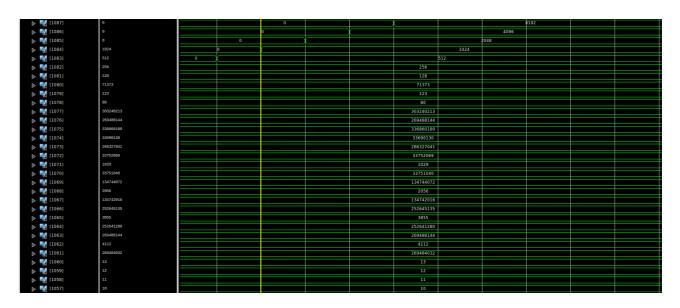
poly2 \$24, \$0, \$22

-- \$24 = 11*80*80 + 12*80 + 13 = 71373

ame	Value	Ι.	1,740 ns	1,750 ns	1,760 ns	1,770 ns	1,780 ns	1,790 ns	1,800 ns	1,810 ns	1,820 ns	1,830 ns
le reset	θ	1										
Ug clock	1											
laclock_period	10000 ps							10000 ps				
d alu_out[10:0]	1026			1026		*	1024		*	1025		
pc_out[31:0]	136							136				
instr[31:0]	10000000000110001011000000010000		*				10000000	0011000101100000	010000			
tourrent_state	poly9		start	poly1	poly2	poly3	poly4	poly5	poly6	poly7	poly8	poly9
alu_out_sig[31:0]	960	0	80		0		64	100	70400	58368	90	i0
a[31:0]	80			0			*	30	6400	70400	8	0
■ b[31:0]	12	0	80	X	0			30	11	x	12	
du_reg1_out[31:0]	78488			8		×	0		6400	X	7040	0
alu_reg2_out[31:0]	0			20		*			0			
poly_out[31:0]	8			88			*	•	*	4		
olymuxout[31:0]	8			8		×	0		*	4		
sig_in_mux32[0:31]	[0,10,11,12,13,269484832,4112,269	٠ 🗆	[0,10,11,12,13	,269484032,4112,26	488144,252641280,	3855,252645135,134	742016,2056,13474	4072,33751040,1029	33752069,2863270	1,33690130,336860	80,269488144,3032	0213,80,123,
l ram[2047:0]	[00000000000000000000000000000000000000	[00	00000000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000	000,0000000000000	000000000000000000000000000000000000000	0000,0000000000000	000000000000000000000000000000000000000	000000000000,00000	000000000000000000000000000000000000000	00000000000,0
alu_out[31:0]	960	0	80	X	0		64	100	70400	58368	91	i0
■ rf_a(31:0)	0							0				
rf_b[31:0]	80	0	*					80				
ddout[31:0]	12	0	K		13		K	11		X	12	
rf_b_sel[1:0]	66							00				
ard2[4:0]	22	0	K					22				
dout2[31:0]	80	0	K					80				
din[31:0]	78466			8		*	0		6400	X	7040	0
output[4:0]	θ							0				
		Π										
		Т										

Name	Value		1,830 ns	1,840 ns	1,850 ns	1,860 ns	1,870 ns	1,880 ns	1,890 ns
lo reset	0								
∏ _o clock	1								
Clock_period	10000 ps							10000 ps	
alu_out[10:0]	1056		x		10	026			
pc_out[31:0]	140				136				
instr[31:0]	111000000011001000000010000000				1000000000	0110001011000000	10000		
튢 current_state	li_lui2		poly9	poly10	poly11	poly12	poly13	poly14	final
alu_out_sig[31:0]	128		960	704	13	71	73	72333	0
▶ 📑 a[31:0]	0		80	704	00	70	13	71373	
▶ ■ b[31:0]	128		12	1	3	X	960		0
alu_reg1_out[31:0]	128			70400		70	13	k	71
alu_reg2_out[31:0]	960		0	X				960	
poly_out[31:0]	8	4	X					8	
polymuxout[31:0]	128	4	x			8			
sig_in_mux32[0:31]	[0, 10, 11, 12, 13, 269484032, 4112, 2694	[0,	10,11,12,13,2694840	32,4112,269488144,	52641280,3855,252	645135,134742016,	056,134744072,337	51040,1029,3375	[0,10,11,12,13,269
ram[2047:0]	[00000000000000000000000000000000000000	[00	00000000000000000	000000000000000000000000000000000000000	00000000000000000	00000000000000,000	00000000000000000	0000,00000000,0000	000000000000000000000000000000000000000
alu_out[31:0]	128		960	704	13	71	73	72333	0
rf_a[31:0]	0							0	
rf_b[31:0]	128					80			
▶ ■ ddout[31:0]	0		12	X		1	3		
rf_b_sel[1:0]	01					00			
ard2[4:0]	25					22			
▶ ■ dout2[31:0]	128					80			
▶ 📑 din[31:0]	128			70400		70	13	*	71
output[4:0]	0							0	

Η μνήμη μετά την rfst



Οι καταχωρητές μετά την εκτέλεση όλων των εντολών έχουν τις παρακάτω τιμές

