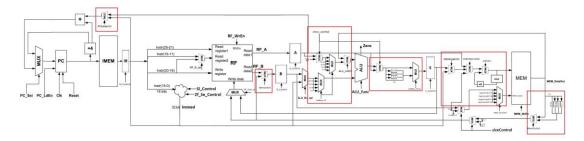
# Αναφορά Εργαστηρίου 4

#### LAB31231454

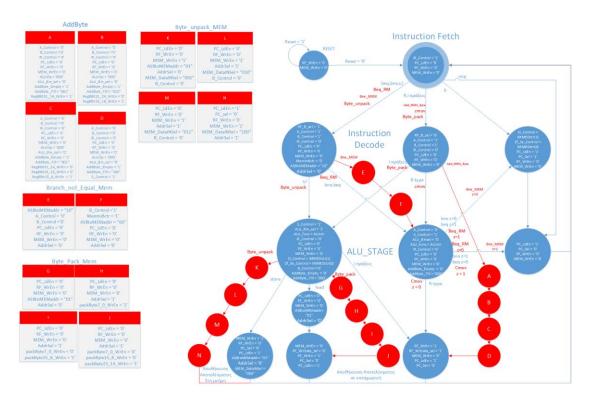
Μανώλης Πετράκος	
Μιχαήλ Δακανάλης	

## Προεργασία

### Datapath:



#### FSM:



Ότι καινούργιο είναι με κόκκινο χρώμα. Επίσης στην fsm οι καινούργιες καταστάσεις έχουν σε υπόμνημα τα σήματα τους.

## Περιγραφή Άσκησης

Προσθέσαμε ό,τι λογική χρειαζόταν (πολυπλέκτες, καταχωρητές κ.λπ.) στο datapath από το προηγούμενο εργαστήριο και προσθέσαμε ότι παραπάνω καταστάσεις και ελέγχους χρειάζονταν στην fsm.

Cmov: Μετά το Instr\_Fetch κάνει decode όπως οι R εντολές. Στο ALU\_Stage κάνει μία αφαίρεση του rt με το 0 χρησιμοποιώντας ένα πολυπλέκτη για να αντικαταστήσει το A και αν το αποτέλεσμα είναι διαφορετικό του μηδέν βάζει τον rs στον S από τον A και μετά τον αποθηκεύει στο αρχείο καταχωρητών. Αλλιώς απλά πάει στην επόμενη εντολή.

Add\_MMX\_byte: Μετά το Instr\_Fetch κάνει decode όπως οι R εντολές. Έπειτα σπάει τα περιεχόμενα κάθε καταχωρητή (A,B) σε 4 bytes χρησιμοποιώντας δύο πολυπλέκτες. Στους επόμενους τρεις κύκλους κάνει πράξεις με τα αντίστοιχα bytes και τα αποθηκεύει σε καταχωρητές ενός byte. Στον τέταρτο κάνει τη τελευταία πράξη και κατευθείαν ενώνει τα αποτελέσματα και τα αποθηκεύει στον S. Τέλος αποθηκεύει το αποτέλεσμα στον RF.

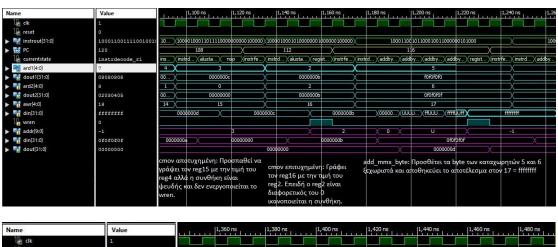
branch\_equal\_reg\_mem: Μετά το Instr\_Fetch κάνει decode και το ALU\_Stage όπως οι branch εντολές εκτός του ότι στην θέση του rs χρησιμοποιεί μια τιμή από την μνήμη, επιλεγμένη από τον Immediate με τη χρήση ενός πολυπλέκτη στην διεύθυνση της μνήμης και ενός δεύτερου στην πρώτη είσοδο της ALU. Τέλος αν το αποτέλεσμα της πράξης είναι 0 θα προσθέσει στον PC την τιμή του rs, που είναι στον A.

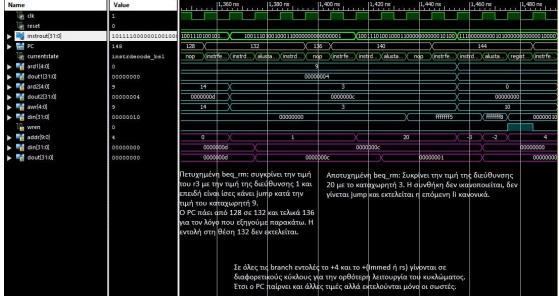
branch\_not\_equal\_mem: Μετά το Instr\_Fetch συνεχίζει στο decode όπως οι branch εντολές. Ύστερα μέσω ενός πολυπλέκτη παίρνει τις τιμές των rs και rt από τους καταχωρητές A και B και με αυτές παίρνουμε από την μνήμη δύο τιμές και τις συγκρίνουμε. Επειδή αυτή η διαδικασία χρειάζεται δύο κύκλους, κατά τον πρώτο αποθηκεύουμε στον καταχωρητή B το αποτέλεσμα του MEM[rt] και στον δεύτερο που φτάνει το MEM[rs] γίνεται ή πράξη κατευθείαν. Αν το αποτέλεσμα είναι διάφορο του μηδέν προσθέτουμε στον PC τον Immed της εντολής. Αλλιώς συνεχίζεται η εκτέλεση του προγράμματος σειριακά.

Byte\_pack\_mem: Μετά το Instr\_Fetch κάνει decode και το ALU\_Stage όπως οι load εντολές για να δημιουργήσει το base\_addr. Έπειτα βάζει το αποτέλεσμα σε έναν καταχωρητή, κατευθείαν βγάζει από την μνήμη τη πρώτη τιμή και αποθηκεύει το πρώτο byte της σε έναν καταχωρητή. Μετά επαναλαμβάνει την διαδικασία αυτή αυξάνοντας την διεύθυνση κατά 4 μέχρι να βγάλει δύο ακόμα αποτελέσματα και να αποθηκεύσει τα πρώτα τους byte σε έναν καταχωρητή το καθένα. Τέλος βγάζει την τελευταία τιμή και ενώνει το τελευταίο της byte με τις τιμές που έχουμε αποθηκεύσει και βάζουμε το αποτέλεσμα στην μονάδα καταχωρητών.

Byte\_unpack\_mem: Μετά το Instr\_Fetch κάνει decode και το ALU\_Stage όπως οι i-type εντολές για να δημιουργήσει το base\_addr. Χρησιμοποιεί την ίδια διαδικασία με τη προηγούμενη εντολή για να ελέγξει την διεύθυνση της μνήμης. Σε κάθε κύκλο αποθηκεύει το κάθε byte του rd sign extended στην αντίστοιχη θέση μνήμης.

### Κυμματομορφές





Name	me Value					1,500 ns		1,520	ns	1,5	1,540 ns			L,560 ns		1,580 ns		1,6	00 ns	ST 1 1 1	1,62	0 ns		1,640	,640 ns		
Ŭe dk	0																										
Ve reset	0																										
instrout[31:0]	1111000000110101000	11100000000001010000 X				101111000000100100000000000000100						e e	X 10			1 110 10 100 10 100000000000000				00000000 X					111000000		
▶ 🛗 PC	160	144	$\perp x$				148				X 1	52				156				$\equiv$ X $\equiv$					160		
le currentstate	bytepackb	alu (reg	jist)(i	nstrfe	inst	d\bn	ebt Xbner	ne)(	alusta)	instrd	X n	op (instr	fe)	instrd	bne	bt)bn	emeX	alusta.	. no	p (ins	trfe	instrd	) (alus	ta)(t	ytep by		
▶ 📑 ard1[4:0]	1						0										10					X		(			
▶ 📑 dout1[31:0]	0000000a					00000000								0000001				010	0			X					
▶ 📑 ard2[4:0]	0		0						9								10					X		8			
▶ 📆 dout2[31:0]	00000000	000	00000		$\subset$			0	0000004								00000	010				X					
▶ 📑 awr[4:0]	21		10		$\subset$				9								10					X ==		Ņ.			
▶ 📑 din[31:0]	00000050	fff)(	0	0000010	P)	$\supset$	00000004	$=$ $\times$	00000)	00000	$\propto$		0000	0000		(ff	fffffd X	ffff <del>ff</del> 0	$\times$	000	00000	4	(000	0X			
la wren	0												8								_						
▶ 📑 addr[9:0]	21	-2		4			1				0				X =				4				X	$\supset$	20 )		
▶ 📑 din(31:0)	00000000		00000	0000			00000004	_X	00000)			P00000000			$\times$	0000001	0 (								00000000		
• 🖷 dout[31:0]	00000001		(	00000000	0		(0000	0)(				D0000000				$\rightarrow$ C				000	00000				(00		
		πετυχημένη bne_mem: Συγκρίνει τις δείχνουν οι reg0 και reg9 και επειδή ένα jump κατά την τιμή του Immed.							είναι διαφορετικοί κάνει						ίχνοι	v ot re	g10 ко	u reg	υγκρίνει τις θέσεις τη: reg10 και επειδή είναι ανονικά στην επόμενη			ναι ίσει	δεν				



# Συμπεράσματα

Μάθαμε πως να επεκτείνουμε της εντολές ενός επεξεργαστή πολλαπλών κύκλων προσθέτοντας επιπλέον λογική στην σχεδίασή μας.