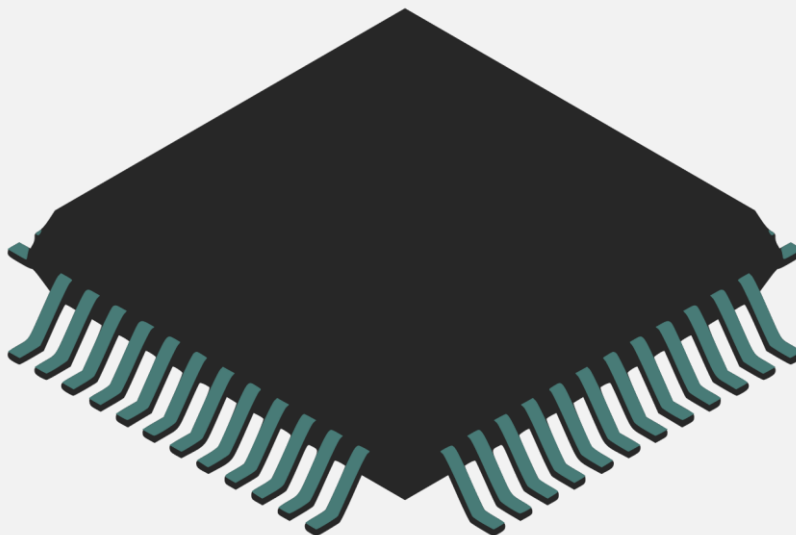


ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά 3^{ης} Εργαστηριακής Άσκησης

«ΟΛΟΚΛΗΡΩΣΗ ΚΑΙ ΠΡΟΣΩΜΟΙΩΣΗ DATAPATH ΕΝΟΣ
SINGLE-CYCLE ΕΠΕΞΕΡΓΑΣΤΗ»


















 Ζαχαριουδάκης Νικόλας 2016030073

 Γαλάνης Μιχάλης 2016030036

ΠΕΡΙΕΧΟΜΕΝΑ

*Οι σύνδεσμοι για τις παρακάτω ενότητες είναι διαδραστικοί.
Πιέστε πάνω στο επιθυμητό τμήμα για τη μετάβαση σε αυτό.*

	ΕΙΣΑΓΩΓΗ.....	1
	Σκοπός Εργαστηρίου	1
	Προαπαιτούμενα.....	1
	A – ΟΛΟΚΛΗΡΩΣΗ DATAPATH.....	1
	A. Επισκόπηση	1
	A. Σχεδίαση & Υλοποίηση	2
	Δ. Κώδικας VHDL (Byte Checker)	3
	B – ΕΠΑΛΗΘΕΥΣΗ DATAPATH.....	4
	B. Επαλήθευση Datapath (Testbench)	4
	B.1. Πρόγραμμα Αναφοράς #1	4
	B.1. Προσομοίωση Π.Α #1 (Κυματομορφές)	5
	B.2. Πρόγραμμα Αναφοράς #2	6
	B.2. Προσομοίωση Π.Α #2 (Κυματομορφές)	7
	B.3. Πρόγραμμα Αναφοράς #3	8
	B.3. Προσομοίωση Π.Α #3 (Κυματομορφές)	8

ΕΙΣΑΓΩΓΗ

Σκοπός Εργαστηρίου

Σκοπός της τρίτης εργαστηριακής άσκησης είναι η ολοκλήρωση και η επαλήθευση του Datapath που σχεδιάσαμε στην προηγούμενη άσκηση. Στη παραπάνω διαδικασία περιλαμβάνεται η ένωση των επιμέρους βαθμίδων με μοναδικό – κοινό instance της μνήμης καθώς και η επιβεβαίωση ορθής λειτουργίας του datapath με διαφορετικά αρχεία αρχικοποίησης της μνήμης.

Προαπαιτούμενα

Χρειάζεται η άπταιστη γνώση της VHDL και των δομών λειτουργίας και η εξοικείωση στα εργαλεία σχεδιασμού της Xilinx. Θα χρησιμοποιηθεί επίσης πλήρως το υλικό που παράχθηκε στην προηγούμενη εργαστηριακή άσκηση.

Α – ΟΛΟΚΛΗΡΩΣΗ DATAPATH

Αν και η ένωση των βαθμίδων του Datapath είχε ήδη επιτευχθεί από τη προηγούμενη εργαστηριακή άσκηση, σε αυτό το σημείο γίνεται η αναλυτική περιγραφή του.

Α. Επισκόπηση

Υπενθυμίζουμε ότι το Datapath ενός επεξεργαστή αποτελείται από τις παρακάτω βαθμίδες:

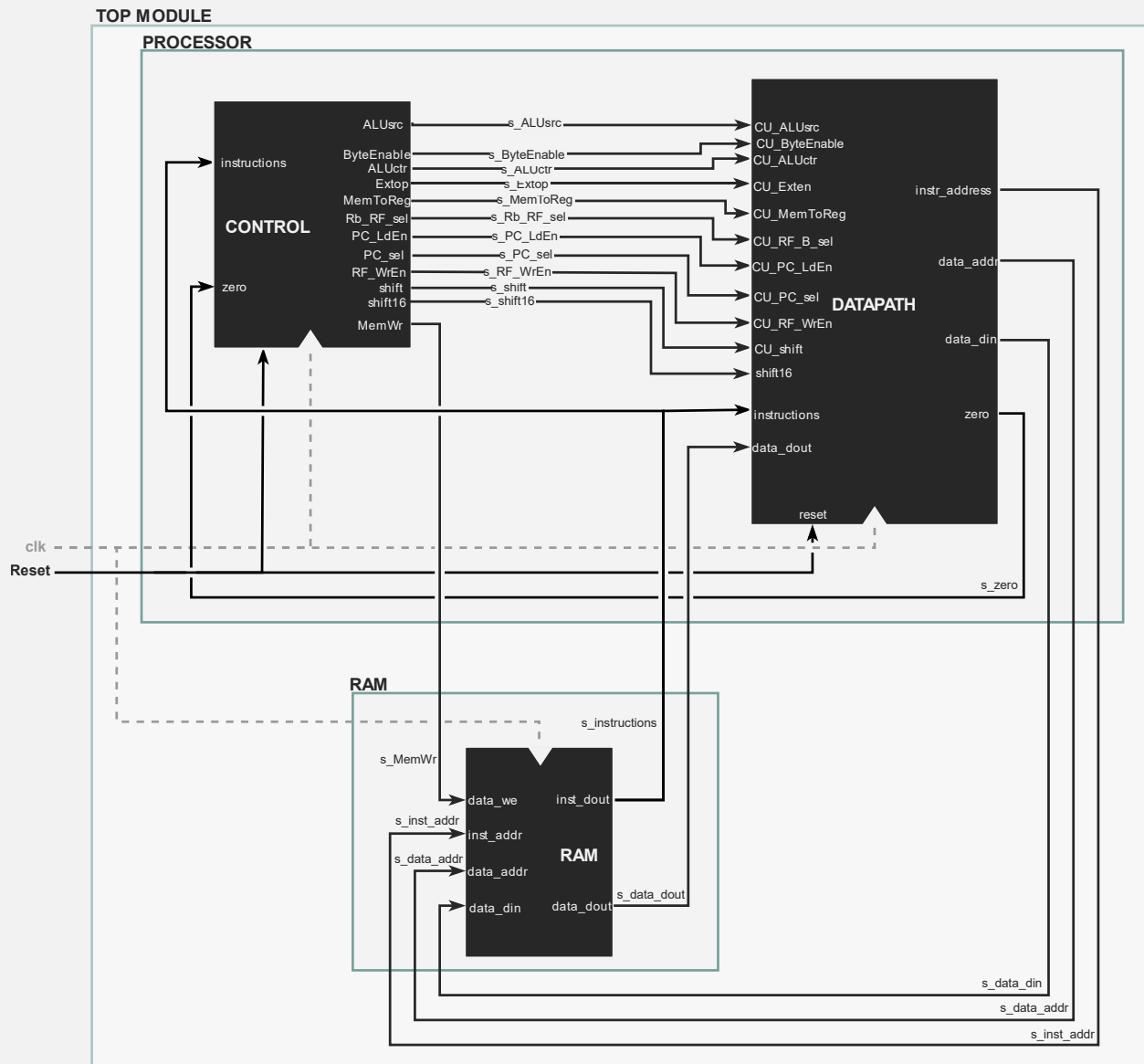
- ALUSTAGE (Βαθμίδα εκτέλεσης εντολών)
- DECSTAGE (Βαθμίδα αποκωδικοποίησης εντολών)
- MEMSTAGE (Βαθμίδα πρόσβασης μνήμης)
- IFSTAGE (Βαθμίδα ανάκλασης εντολών)

Για τις ανάγκες των εντολών **lb, sb**, δημιουργήθηκαν επίσης 2 modules ByteCheckerLoad & ByteCheckerStore τα οποία αναλύονται παρακάτω.

Τη μνήμη RAM θα μπορούσε κάποιος να τη τοποθετήσει ως εσωτερικό module του Datapath για τις ανάγκες της εργαστηριακής άσκησης. Εμείς επειδή έχουμε την ολοκληρωμένη δομή του επεξεργαστή όπως αυτή παρουσιάζεται σε επόμενα εργαστήρια, η μνήμη RAM στην υλοποίησή μας δεν αποτελεί μέρος του Datapath και ούτε καν του ίδιου του επεξεργαστή.

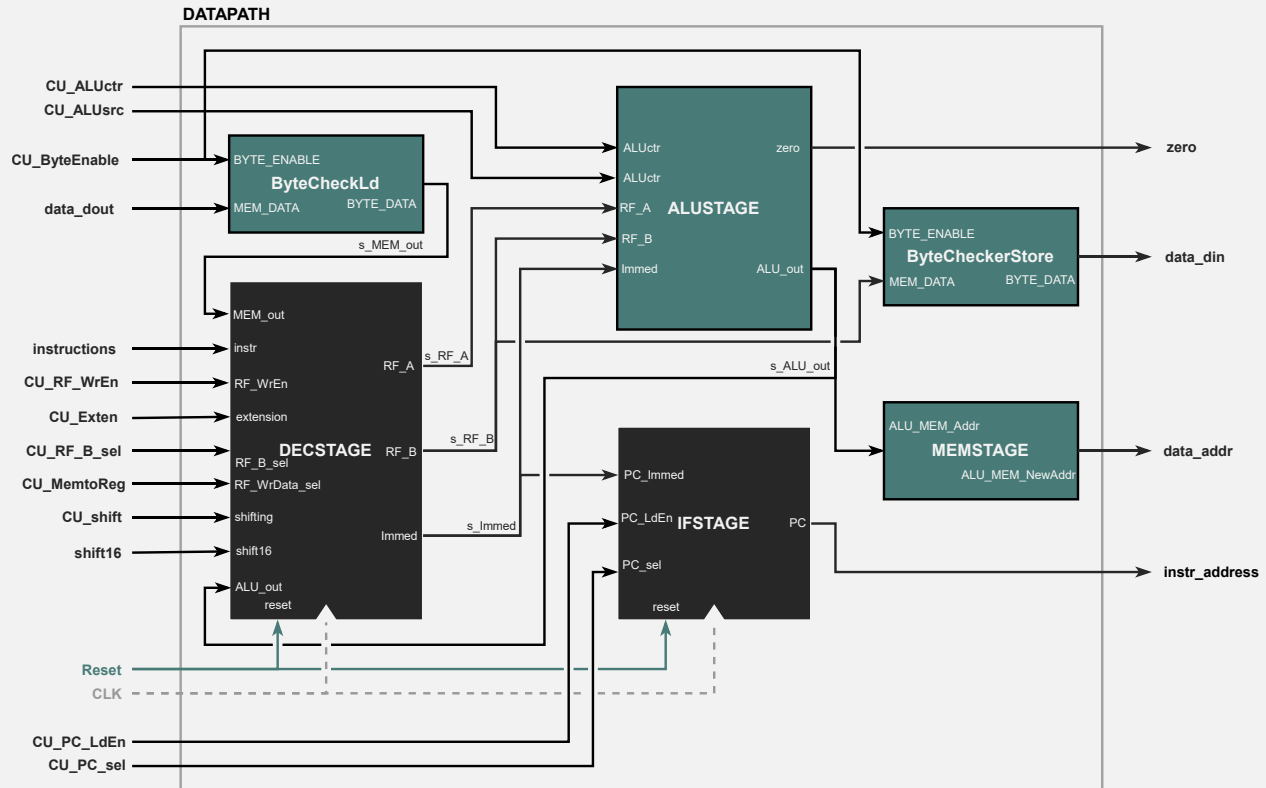
Α. Σχεδίαση & Υλοποίηση

Παρακάτω παρουσιάζεται το block diagram υψηλού επιπέδου που περιγράφει τη λειτουργία του επεξεργαστή:



⚠ Παρατήρηση: Στο Datapath όλες οι εισόδους που προέρχονται από το Control Unit έχουν όνομα με πρόθεμα **CU**.

Παραθέτουμε επίσης και το block diagram του Datapath συμπεριλαμβάνοντας τα ενδιάμεσα σήματα (signals).



Οι μόνες αλλαγές που υπήρξαν στο εσωτερικό του Datapath ήταν η προσθήκη των 2 instances ByteCheckerLoad, ByteCheckerStore του module ByteChecker. Αυτά είναι υπεύθυνα για τη μετατροπή ενός 32-bit σήματος σε 8-bit για την υποστήριξη των λειτουργιών lb, sb (ανάγνωση/εγγραφή ενός byte). Όπως φαίνεται στο παραπάνω διάγραμμα, αυτά τοποθετήθηκαν απευθείας στο Datapath και όχι στο εσωτερικό του DECSTAGE καθώς δεν ασχολούνται με «αποκωδικοποίηση εντολών». Επαλήθευση για το συγκεκριμένο module δεν υλοποιήθηκε λόγω της απλότητάς του.

Μια άλλη μικρότερη αλλαγή ήταν η προσθήκη εισόδου shift16 στη βαθμίδα αποκωδικοποίησης εντολών για την αποκωδικοποίηση της εντολής **lui**.

Δ. Κώδικας VHDL (Byte Checker)

```
begin
  process (MEM_DATA, BYTE_ENABLE)
  begin
    if BYTE_ENABLE = '1' then
      BYTE_DATA(31 downto 8) <= (31 downto 8 => '0');
      BYTE_DATA(7 downto 0) <= MEM_DATA(7 downto 0);
    else
      BYTE_DATA <= MEM_DATA;
    end if;
  end process;
end Behavioral;
```

B – ΕΠΑΛΗΘΕΥΣΗ DATAPATH

Αφού πλέον έχουμε ενώσει τις βαθμίδες του Datapath και τη μνήμη, δεν είναι απαραίτητο να βάζουμε χειροκίνητα τιμές στο testbench. Η διαδικασία εκτέλεσης εντολών και ανάγνωση/εγγραφή δεδομένων επιτυγχάνεται απευθείας από τη μνήμη.

Για την επαλήθευση διαφόρων τμημάτων του συστήματος αρκεί λοιπόν στο testbench να ξεκινήσουμε τη διαδικασία εκτέλεσης εντολών στη μνήμη από τη θέση 0.

Στην υλοποίησή μας χρησιμοποιούμε 3 προγράμματα αναφοράς. Το κάθε πρόγραμμα αναφοράς κάνει χρήση ενός αρχείου ram.data το οποίο διαθέτει αρχικοποιημένες τιμές σε διάφορες θέσεις μνήμης.

B. Επαλήθευση Datapath (Testbench)

Ο παρακάτω κώδικα του Testbench είναι κοινός και για τα 3 προγράμματα αναφοράς.

```
begin
    reset <= '1';
    wait for clk_period*1;

    reset <= '0';
    wait;
end process;
```

B.1. Πρόγραμμα Αναφοράς #1

Το πρώτο πρόγραμμα αναφοράς είναι υπεύθυνο για την επαλήθευση της εκτέλεσης διαφόρων εντολών γενικού χαρακτήρα, μεταξύ των οποίων εκτέλεση πράξεων, ανάγνωση από/εγγραφή σε μνήμη/καταχωρητές. Πιο συγκεκριμένα:

Θέση	Εντολή	Opcode	rs	rd	rt	func	Immed
00	addi r5,r0,8	110000	00000	00101	-	-	00000000000001000
04	ori r3,r0,ABCD	110011	00000	00011	-	-	1010101111001101
08	sw r3,4(r0)	011111	00000	00011	-	-	00000000000000100
0C	lw r10,-4(r5)	001111	00101	01010	-	-	11111111111111100
10	lb r16,4(r0)	000011	00000	10000	-	-	00000000000000100
14	nand r4,r10,r16	100000	01010	00100	10000	110010	-
18	li r17,8000	111000	00000	10001	-	-	10000000000000000

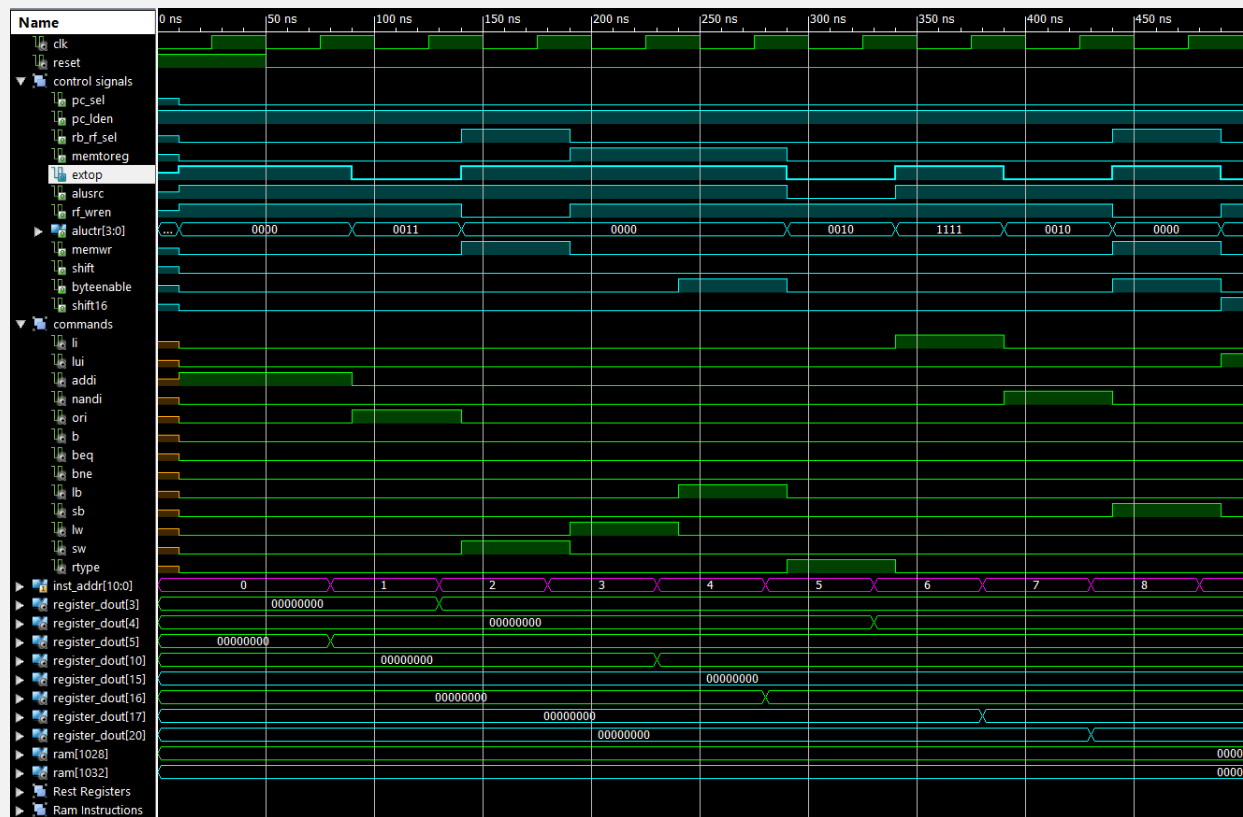
1C	nandi r20,r0,FFFF	110010	00000	10100	-	-	1111111111111111
20	sb r10,8(r0)	000111	00000	01010	-	-	0000000000001000
24	lui r15,FFFF	111001	00000	01111	-	-	1111111111111111

🕒 B.1. Προσομοίωση Π.Α #1 (Κυματομορφές)

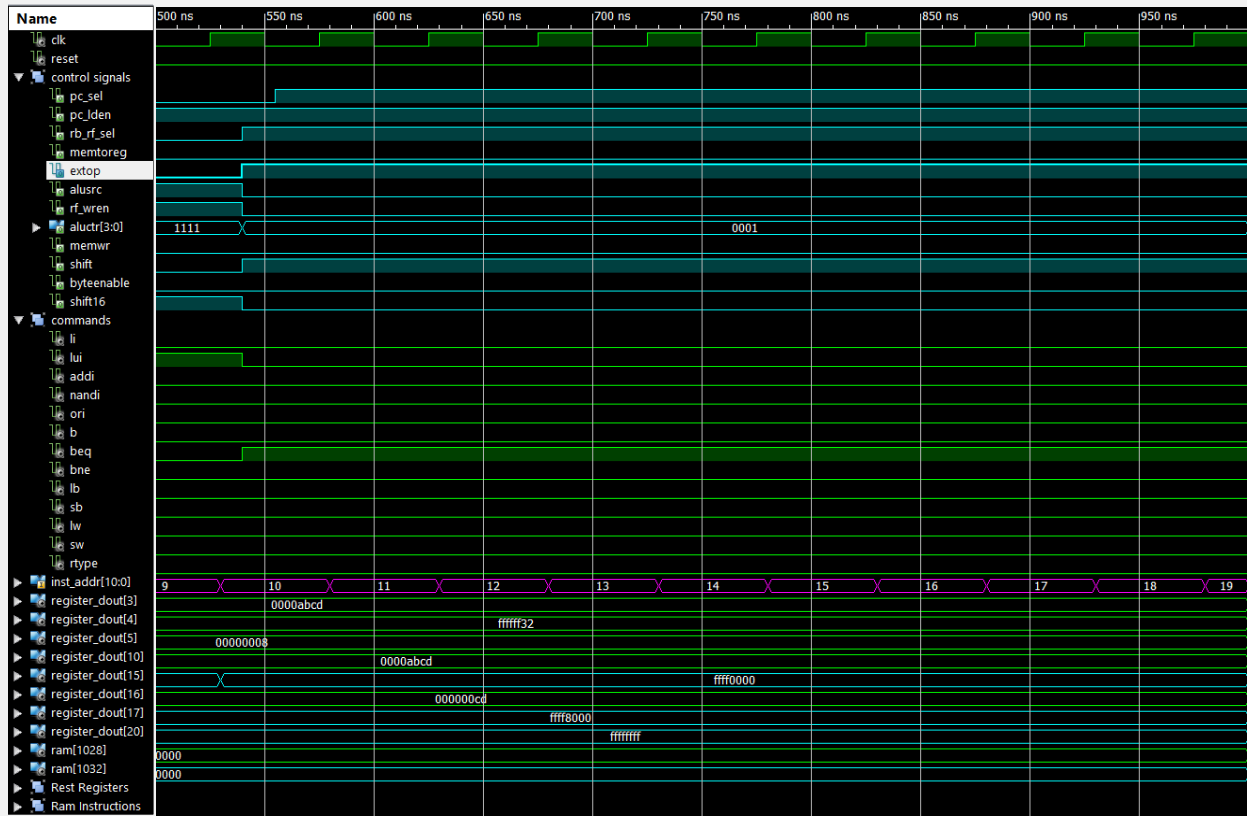
Στις παρακάτω κυματομορφές φαίνονται, για κάθε περίπτωση που έχουμε, τι εντολή παράγεται απο την μνήμη μας και ποια είναι τα σήματα αυτόματης εξόδου απο το control unit, που δέχεται τις εντολές. Επιπλέον, στις επιμέρους περιπτώσεις φαίνονται και οι καθυστερήσεις που είχαμε βάλει στο προηγούμενο εργαστήριο στα submodules, όπως για παράδειγμα στην ανάγνωση της πρώτης θέσης μνήμης που την παραγόμενη εντολή και σήματα τα λαμβάνουμε μετα απο 10ns, η οποία είναι και η καθυστέρηση της εξόδου της μνήμης μας. Το latency αυτό φαίνεται στο ίδιο παράδειγμα και στο αποτέλεσμα της εγγραφής του register, αφού την έξοδο την παίρνουμε 5ns μετά την θετική ακμή του ρολογιού.

Υπενθυμίζεται επίσης ότι στο DECSTAGE πριν την είσοδο Din του RF προηγείται ένας multiplexer, που έχει καθυστέρηση εξόδου ίση με 5 ns.

0 ns – 500 ns:



500 ns – 1000 ns:



✎ B.2. Πρόγραμμα Αναφοράς #2

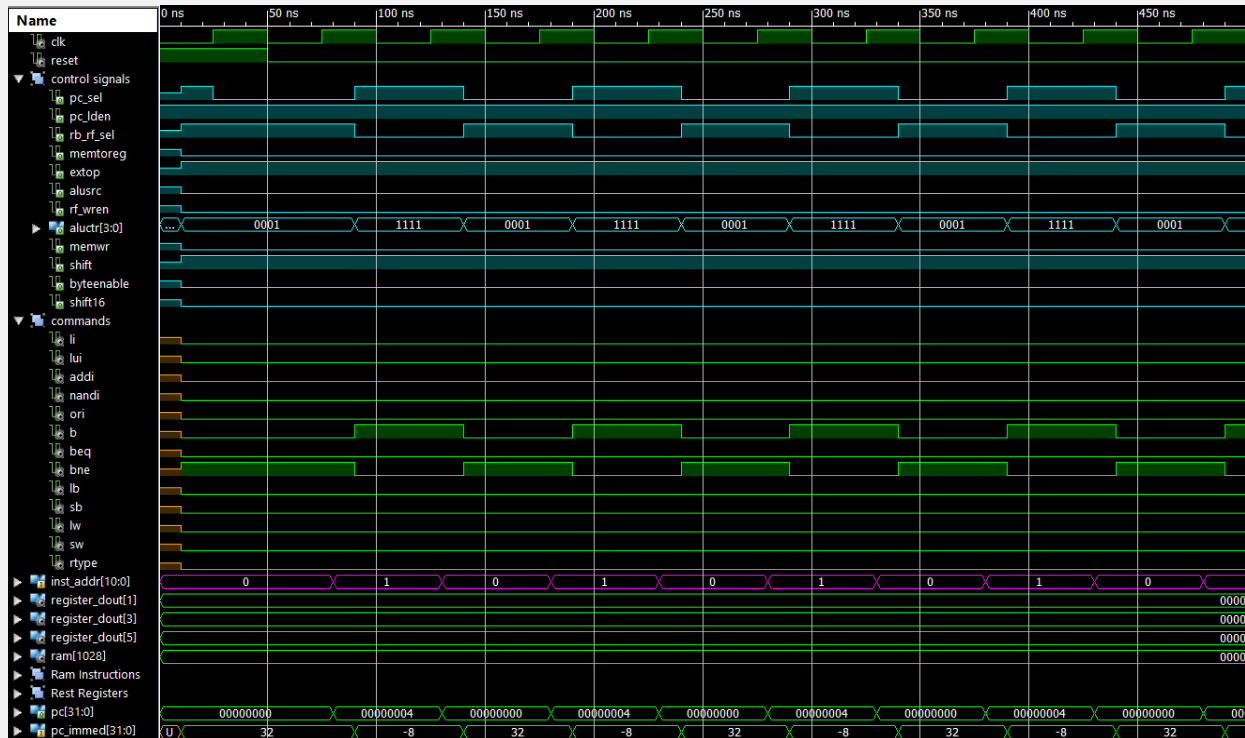
Το δεύτερο πρόγραμμα αποσκοπεί στη μελέτη συμπεριφοράς του συστήματος όταν τροφοδοτηθεί με αποτυχημένες διακλαδώσεις.

Θέση	Εντολή	Opcode	rs	rd	rt	func	Immed
00	bne r5,r5,8	000001	00101	00101	-	-	000000000 0001000
04	b -2	111111	00000	00000	-	-	111111111 1111110
08	add r1,r2,r3	100000	00010	00001	00011	110000	-

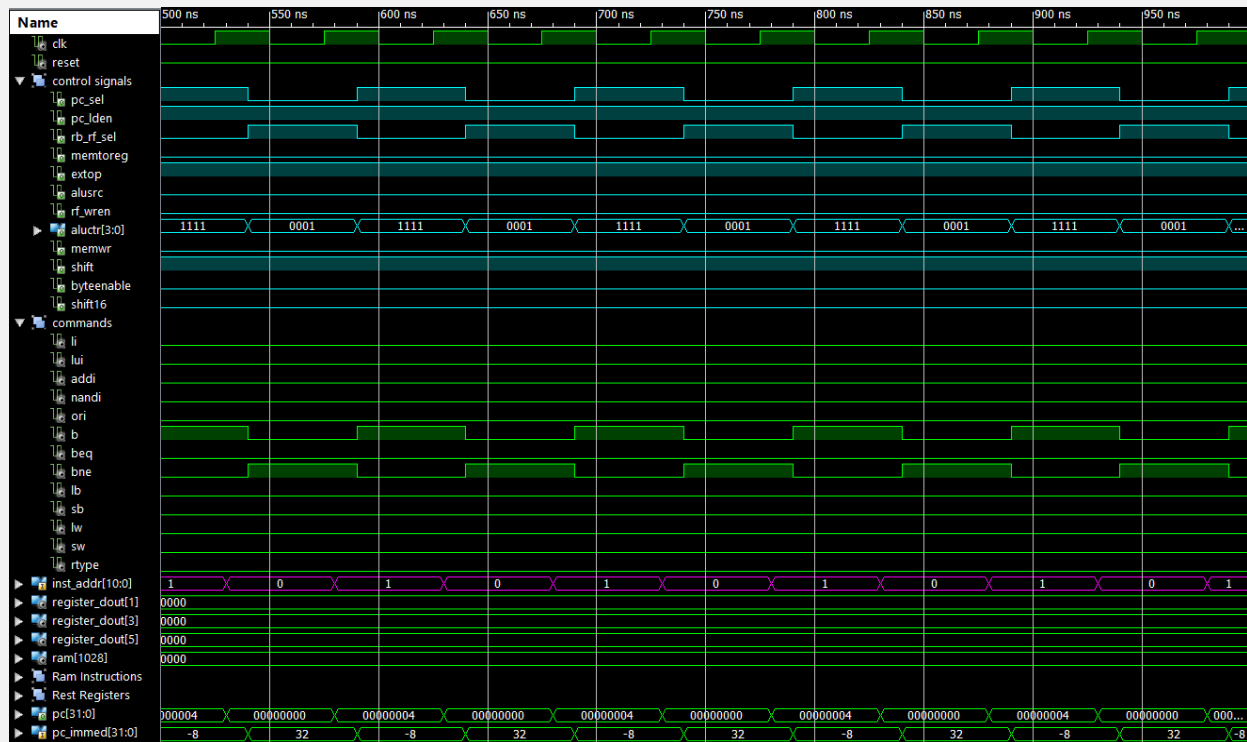
🕒 B.2. Προσομοίωση Π.Α #2 (Κυματομορφές)

Σε αυτές τις κυματομορφές παρατηρούμε ότι η υλοποίησή μας για τα δύο είδη branch (b,bne) δουλεύουν ορθά καθώς κατά την σύγκριση bne ενός καταχωρητή με τον εαυτό του, όταν έχουμε ανεπιτυχές branch not equal, ο PC αυξάνεται κατά 4. Τέλος, στην περίπτωση b -2 η νέα τιμή του program counter πρέπει να είναι $PC = PC + 4 + \text{immed} = 4 + 4 - 8 = 0$.

0 ns – 500 ns:



500 ns – 1000 ns:



B.3. Πρόγραμμα Αναφοράς #3

Το τρίτο και τελικό πρόγραμμα αποσκοπεί στην επαλήθευση του συστήματος για μια επιτυχημένη διακλάδωση.

Θέση	Εντολή	Opcode	rs	rd	rt	func	Immed
00	addi r5,r5,8	110000	00000	00101	-	-	000000000 0001000
04	ori r3,r0,ABCD	110011	00000	00011	-	-	101010111 1001101
08	bne r5,r3,8	000001	00101	00011	-	-	000000000 0001000
2C	add r1,r2,r3	100000	00010	00001	00011	110000	-

B.3. Προσομοίωση Π.Α #3 (Κυματομορφές)

Στο τελευταίο πρόγραμμα αναφοράς όπου επαληθεύουμε επιτυχημένη διακλάδωση (bne), παρατηρούμε οι πρώτες 3 εντολές να εκτελούνται κανονικά. Η 3^η εντολή είναι bne δυο καταχωρητών με διαφορετικές τιμές, οπότε ο PC θα πάρει τιμή $PC = PC + 4 +$

$\text{Imm(Byte)} = 8 + 4 + 8 \cdot 4 = 44$ έτσι ώστε η add που βρίσκεται στη θέση μνήμης εκείνη, να αποθηκεύσει το αποτέλεσμα στον καταχωρητή R1.

0 ns – 300 ns:

