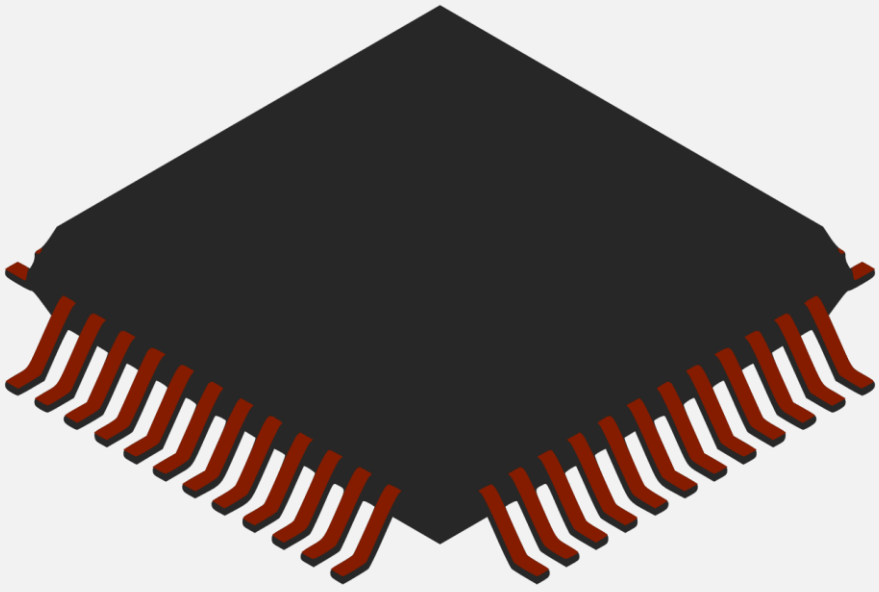
ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά 2ης Εργαστηριακής Άσκησης

«ΣΧΕΔΙΑΣΗ ΒΑΣΙΚΩΝ ΒΑΘΜΙΔΩΝ ΤΟΥ DATAPATH ΕΝΟΣ ΑΠΛΟΥ 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

Σκοπός Εργαστηρίου

Σκοπός της δεύτερης εργαστηριακής άσκησης είναι η σχεδίαση κάποιων βασικών βαθμίδων που ορίζουν την αρχιτεκτονική συνόλου εντολών ενός non-pipelined επεξεργαστή. Οι βαθμίδες προς μελέτη είναι οι παρακάτω:

- Βαθμίδα Ανάκλησης Εντολών
- Βαθμίδα Αποκωδικοποίησης Εντολών
- Βαθμίδα Εκτέλεσης Εντολών (ALU)
- Βαθμίδα Πρόσβασης Μνήμης

Προαιρετική ήταν η ένωση όλων των βαθμίδων για την δημιουργία του Datapath.

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.

ΒΑΣΙΚΗ ΘΕΩΡΙΑ

Η σχεδίαση του επεξεργαστή μας βασίζεται στην αρχιτεκτονική συνόλου εντολών CHARIS-4 (custom-based). Ακολουθούν κάποια τεχνικά χαρακτηριστικά. Αυτή η αρχιτεκτονική περιέχει 32 καταχωρητές των 32 bits ο καθένας όπου ο καταχωρητής R_0 έχει πάντα την τιμή 0. Οι 32-bit εντολές που υποστηρίζονται χωρίζονται σε αριθμητικές – λογικές εντολές (add, sub, and, not, or, shr, shl, sla, rol, ror, li, addi, andi, ori), εντολές διακλάδωσης (b, beq, bneq) και εντολές μνήμης (lb, sb, lw, sw). Οι εντολές αναπαρίστανται με δυο διαφορετικούς τρόπους αναλόγως με το αν έχουμε Immediate αριθμό η όχι:

Register – Type Format

OPCODE[31:26]	RS[25:21]	RD[20:16]	RT[15:10]	N/A	FUNC[5:0]

Immediate – Type Format

OPCODE[31:26]	RS[25:21]	RD[20:16]	IMMEDIATE[15:0]

Παρατηρούμε ότι για τις ανάγκες του εργαστηρίου, δεν έχουμε διαθέσιμες J – Format εντολές που περιγράφουν αναπήδηση του κώδικα. Όλες οι διαθέσιμες εντολές παρουσιάζονται στον επόμενο πίνακα:

OPCODE	FUNC	FORMAT	ΕΝΤΟΛΗ	ΠΡΑΞΗ
100000	**110000**	R-Type	add	RF[rd] = RF[rs] + RF[rt]
100000	**110001**	R-Type	sub	RF[rd] = RF[rs] - RF[rt]
100000	**110010**	R-Type	nand	RF[rd] = RF[rs] NAND RF[rt]
100000	**110100**	R-Type	not	RF[rd] = NOT RF[rs]
100000	**110011**	R-Type	or	RF[rd] = RF[rs] OR RF[rt]
100000	**111000**	R-Type	sra	RF[rd] = RF[rs] >>1
100000	**111001**	R-Type	sll	RF[rd] = RF[rs] <<1 (Log, zero fill LSB)
100000	**111010**	R-Type	srl	RF[rd] = RF[rs] >>1 (Log, zero fill MSB)
100000	**111100**	R-Type	rol	RF[rd] = Rotate left(RF[rs])
100000	**111101**	R-Type	ror	RF[rd] = Rotate right(RF[rs])
111000	**-**	I-Type	li	RF[rd] = SignExtend(Imm)
111001	**-**	I-Type	lui	RF[rd] = Imm << 16 (zero-fill)
110000	**-**	I-Type	addi	RF[rd] = RF[rs] + SignExtend(Imm)
110010	**-**	I-Type	nandi	RF[rd] = RF[rs] NAND ZeroFill(Imm)
110011	**-**	I-Type	ori	RF[rd] = RF[rs] | ZeroFill(Imm)
111111	**-**	I-Type	b	PC = PC + 4 + (SignExtend(Imm) << 2)
000000	**-**	I-Type	beq	PC = (RF[rs] == RF[rd]) ? PC+ 4 + (SignExtend(Imm) << 2) : PC+4
000001	**-**	I-Type	bne	PC = (RF[rs] != RF[rd]) ? PC+ 4 + (SignExtend(Imm) << 2) : PC+4
000011	**-**	I-Type	lb	RF[rd] = ZeroFill(31 downto 8) & MEM[RF[rs] + SignExtend(Imm)](7 downto 0) 000111
000111	**-**	I-Type	sb	MEM[RF[rs] + SignExtend(Imm)] = ZeroFill(31 downto 8) & RF[rd] (7 downto 0)
001111	**-**	I-Type	lw	RF[rd] = MEM[RF[rs] + SignExtend(Imm)]
011111	**-**	I-Type	sw	(MEM[RF[rs] + SignExtend(Imm)]) = RF[rd]

Επειδή η εργαστηριακή άσκηση είναι εκτενής, σχεδιάζουμε και επαληθεύουμε κάθε βαθμίδα ξεχωριστά που είναι υπεύθυνη για τη λειτουργία κάποιου υποσυστήματος.

Α – ΚΩΔΙΚΟΠΟΙΗΣΗ & ALU

Η βαθμίδα της κωδικοποίησης δεν αποτελεί μέρος του Datapath. Αποτελεί το Control Path του επεξεργαστή μας και διαχειρίζεται (FSM) τις λειτουργίες του Datapath σύμφωνα με τον προηγούμενο πίνακα τιμών. Θα μελετηθεί σε μεγάλο βάθος στα επόμενα εργαστήρια. Για την άσκηση αυτή αρκεί να γνωρίζουμε πως γίνεται η κωδικοποίηση των εντολών και την αντίληψη για την επικοινωνία Control Path με τις βαθμίδες του Datapath. Η ALU από την άλλη βρίσκεται στο Datapath και είναι υπεύθυνη για την εκτέλεση πράξεων και εντολών.

Β – ΜΝΗΜΗ RAM (2048 x 32)

Β. Επισκόπηση

Για τις ανάγκες της εργαστηριακής άσκησης, χρειαζόμαστε ένα block μνήμης RAM. Στη περίπτωσή μας αυτή έχει μέγεθος 2048 θέσεων των 32 bits. Επειδή έχουμε ανάγνωση/εγγραφή εντολών/δεδομένων, δεσμεύουμε τις πρώτες $(0x400)_{Hex} = 1024$ θέσεις μνήμης για εντολές και τις υπόλοιπες για δεδομένα.

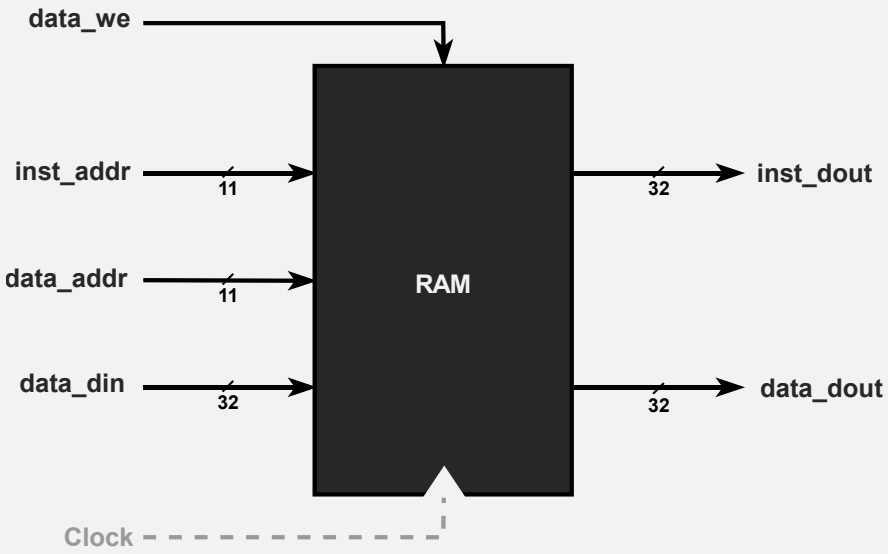
⚠ **Παρατήρηση:** Επειδή η διεθυνσιοδότηση της μνήμης γίνεται με διευθύνσεις byte, τα περιεχόμενα της μνήμης πρέπει να είναι ευθυγραμμισμένα σε πολλαπλάσια των 4 bytes.

Παραθέτουμε τις εισόδους και εξόδους της μνήμης όπως αυτές παρουσιάστηκαν στην εκφώνηση της άσκησης:

Όνομα Σήματος	Πλάτος Σήματος	Είδος Σήματος	Περιγραφή
Inst_addr	**11 bits**	**Είσοδος**	Διεύθυνση εντολής
data_we	**1 bit**	**Είσοδος**	Σημαία ενεργοποίησης εγγραφής στη μνήμη
data_addr	**11 bits**	**Είσοδος**	Διεύθ. για ανάγνωση/εγγραφή δεδομένων
data_din	**32 bits**	**Είσοδος**	Δεδομένα που θα εγγραφούν στη μνήμη
Clk	**1 bit**	**Είσοδος**	Ρολόι
data_dout	**32 bits**	**Έξοδος**	Δεδομένα που διαβάστηκαν από τη μνήμη
Inst_dout	**32 bits**	**Έξοδος**	Εντολή που διαβάστηκε από την μνήμη

Β. Σχεδίαση & Υλοποίηση

Μπορούμε να απεικονίσουμε τη μνήμη μας με το ακόλουθο σχήμα:



Η RAM στη VHDL είναι ένα VHD αρχείο και ο κώδικάς του μας δόθηκε από το προσωπικό του εργαστηρίου. Μας δόθηκε επίσης και το αρχείο ram.data το οποίο είναι υπεύθυνο για την αρχικοποίησή της. Για τους δύο αυτούς λόγους δεν παραθέτουμε το τμήμα του κώδικα της.

⚠ **Παρατήρηση:** Στη γενική περίπτωση δε θα βάζαμε τη RAM στο εσωτερικό του Datapath. Επειδή όμως στη 2η εργαστηριακή άσκηση ασχολούμαστε μόνο με τη λειτουργία του Datapath, αναγκαστικά εντάχθηκε σε αυτό.

Γ – ΒΑΘΜΙΔΑ ΑΝΑΚΛΗΣΗΣ ΕΝΤΟΛΩΝ

Η βαθμίδα ανάκλησης εντολών είναι υπεύθυνη για την αύξηση του Program Counter ανάλογα με τη περίπτωσή μας. Για επιτυχημένο branch έχουμε PC = PC + 4 + Imm ενώ σε οποιαδήποτε άλλη περίπτωση έχουμε PC = PC + 4.

Γ. Επισκόπηση

Αυτή η βαθμίδα αποτελείται από:

- Έναν καταχωρητή PC των 32 bit
- Έναν αυξητή (incrementor) που υπολογίζει το PC + 4
- Έναν αθροιστή (adder) που υπολογίζει το (PC + 4) + Immediate
- Έναν πολυπλέκτη 2 σε 1 των 32 bits που επιλέγει κατάλληλο PC increment

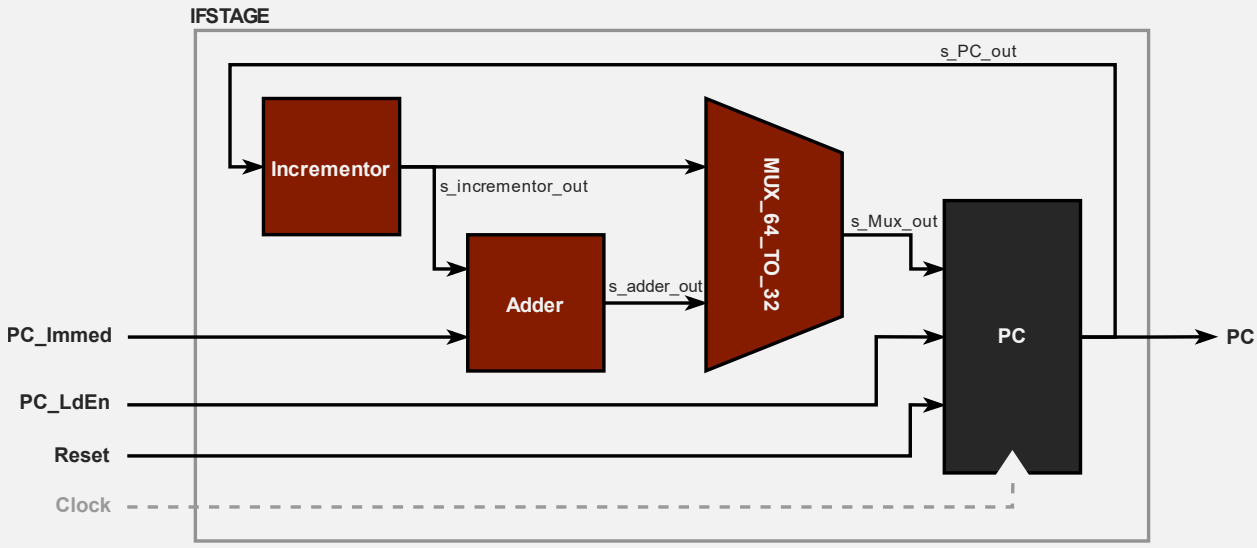
⚠ **Παρατήρηση:** Κατόπιν διόρθωσης της εκφώνησης, η μνήμη RAM πλέον δεν αποτελεί μέρος αυτής της βαθμίδας, παρά βρίσκεται σε υψηλότερο επίπεδο μαζί με τη βαθμίδα MEMSTAGE (Βλ. Ζ – ΒΑΘΜΙΔΑ ΟΛΙΚΗΣ ΜΝΗΜΗΣ).

Παρακάτω φαίνονται οι είσοδοι και έξοδοι του IFSTAGE:

Όνομα Σήματος	Πλάτος Σήματος	Είδος Σήματος	Περιγραφή
PC_Immed	**32 bits**	**Είσοδος**	Τιμή Immediate για εντολές b, beqz, bnez
PC_sel	**1 bit**	**Είσοδος**	Επιλογή για PC + 4, PC + 4 + Immediate
PC_LdEn	**1 bit**	**Είσοδος**	Ενεργοποίηση εγγραφής στον PC
Reset	**1 bit**	**Είσοδος**	Αρχικοποίηση του καταχωρητή PC
Clk	**1 bit**	**Είσοδος**	Ρολόι
PC	**32 bits**	**Έξοδος**	Διεύθυνση εντολής στην μνήμη

Γ. Σχεδίαση & Υλοποίηση

Σχεδιάζουμε τη βαθμίδα ανάκλησης εντολών και παράγεται το ακόλουθο διάγραμμα:



Η υλοποίηση των επιμέρους components είναι απλή εφαρμογή γνωστής ύλης VHDL και δε χρήζει ανάλυσης. Η ευθυγράμμιση της μνήμης έχει ως αποτέλεσμα η κάθε επόμενη εντολή της τρέχουσας να αυξάνει τη διεύθυνση του PC κατά 4. Το μόνο ίσως αξιοσημείωτο είναι ότι ο αυξητής προσθέτει στην εισαγώμενη διεύθυνση 4 bytes και

όχι 4 bits. Για το λόγο αυτό δεν παραθέτουμε τον κώδικα για τη βαθμίδα αυτή (είναι όμως διαθέσιμος σε ξεχωριστό αρχείο .vhd).

Γ. Επαλήθευση IFSTAGE (Testbench)

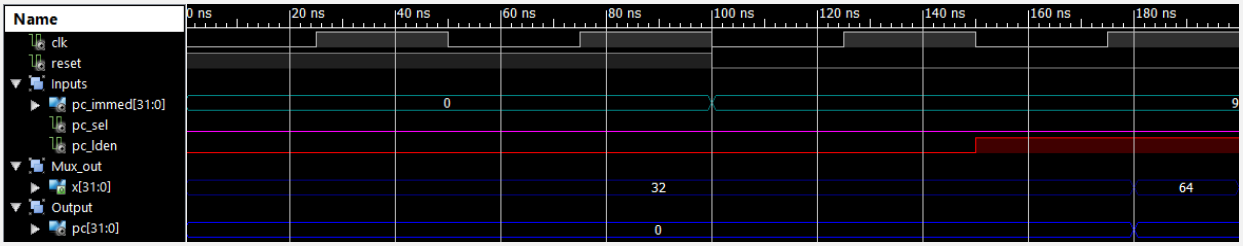
Η συγκεκριμένη βαθμίδα έχει σχετικά λίγα corner cases οπότε επαληθεύσαμε τη λειτουργία της βαθμίδας ανάκλησης μελετώντας τις παρακάτω περιπτώσεις εισόδου:

```
begin
            -------------------------------------------------------------
            Reset <= '1';
            wait for clk_period*2;

            --Reset is disabled but PC_load is also disabled
            Reset <= '0';
            PC_LdEn <= '0';
            PC_Sel <= '0'; -- PC <- PC + 4 bytes
            PC_Immed <= "00000000000000000000000001100000";
            wait for clk_period;

            --Load is now enabled, OUTPUT should be PC <- PC + 4 bytes
            PC_LdEn <= '1';
            wait for clk_period;

            --OUTPUT should now be PC <- PC + 4 bytes + Immediate
            PC_Sel <='1'; --PC <- PC + 4 bytes + Immediate
            wait for clk_period;

            --OUTPUT should now be PC <- PC + 4 bytes
            PC_Sel <= '0'; -- PC <- PC + 4 bytes
            wait for clk_period;

            --Changed Immed Value, OUTPUT should be PC <- PC + 4 + Immediate
            PC_Immed <= "00000000000011110000000000000000";
            PC_Sel <='1'; --PC <- PC + 4 bytes + Immediate
            wait for clk_period;


            --Load is now disabled, OUTPUT should be PC <- PC
            PC_Immed <= "00000000000000000000000000000000";
            PC_LdEn <= '0';
      wait;
            -------------------------------------------------------------
   end process;
```

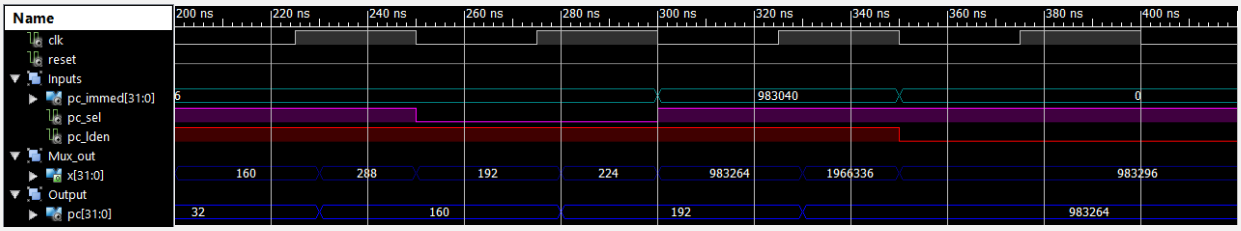
Γ. Προσομοίωση IFSTAGE (Κυματομορφές)

Παρακάτω παρουσιάζεται η προσομοίωση του IFSTAGE:

0 ns – 200 ns:



200 ns – 400 ns:



Δ – ΒΑΘΜΙΔΑ ΑΠΟΚΩΔΙΚΟΠΟΙΗΣΗΣ ΕΝΤΟΛΩΝ

Αυτή η βαθμίδα είναι υπεύθυνη για την αποκωδικοποίηση των R – format εντολών που εξέρχονται από τη μνήμη.

Δ. Επισκόπηση

Αυτή η βαθμίδα αποτελείται από:

- Το αρχείο καταχωρητών που κατασκευάσαμε στο προηγούμενο εργαστήριο
- Έναν πολυπλέκτη 64 σε 32 bits που επιλέγει τα δεδομένα (ALU ή MEM)
- Τον Immediate Extender ο οποίος επεκτείνει τον Immediate από 16 σε 32 bits

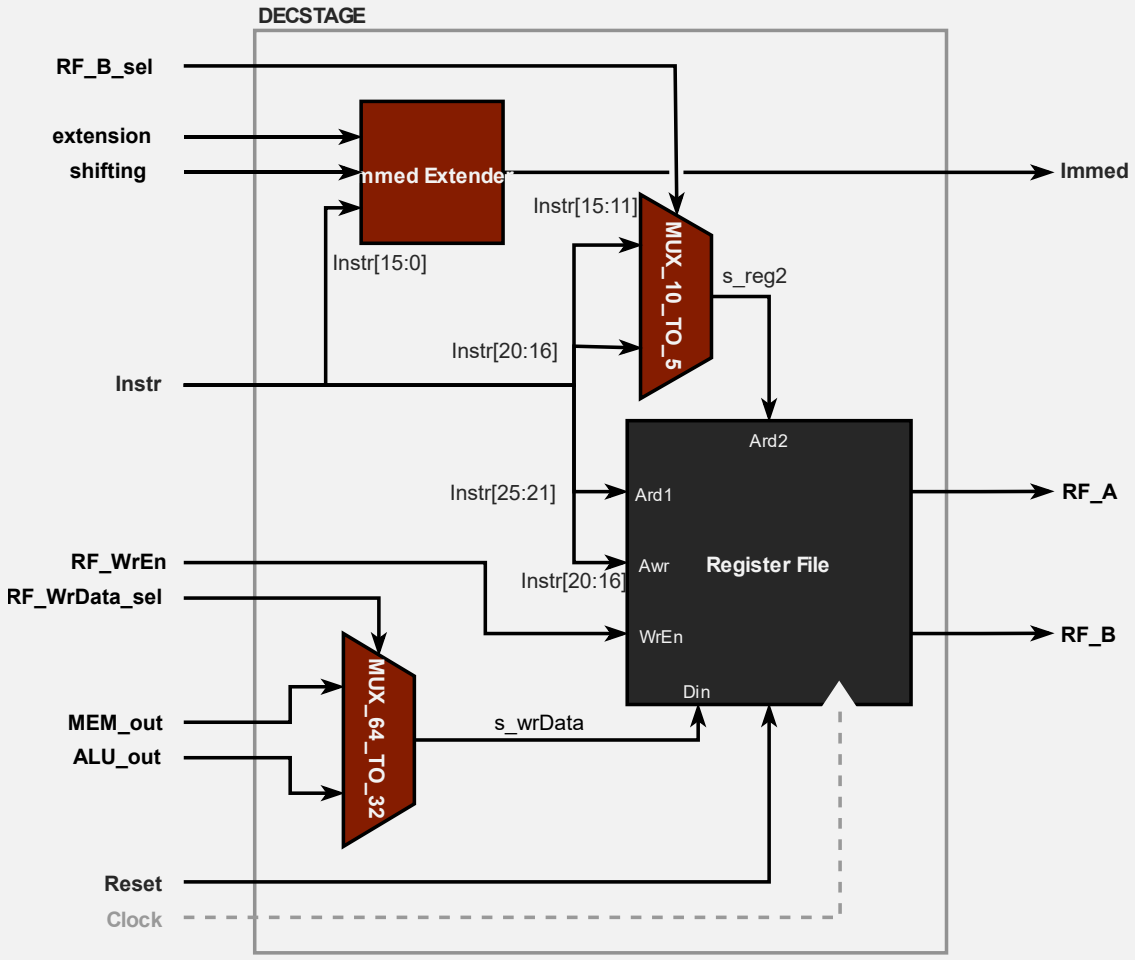
Παρουσιάζουμε παρομοίως τον πίνακα εισόδων/εξόδων του υποσυστήματος αυτού:

Όνομα Σήματος	Πλάτος Σήματος	Είδος Σήματος	Περιγραφή
Instr	**32 bits**	**Είσοδος**	Η εντολή που πρέπει να αποκωδικοποιηθεί
RF_WrEn	**1 bit**	**Είσοδος**	Ενεργοποίηση εγγραφής καταχωρητή
PC_LdEn	**1 bit**	**Είσοδος**	Ενεργοποίηση εγγραφής στον PC
ALU_out	**32 bits**	**Είσοδος**	Δεδομένα εγγραφής προερχόμενα από την ALU

MEM_out	32 bits	Είσοδος	Δεδομένα εγγραφής προερχόμενα από τη μνήμη
RF_WrData_Sel	1 bit	Είσοδος	Επιλογή Προέλευσης Δεδομένων (ALU/MEM)
RF_B_sel	1 bit	Είσοδος	Επιλογή δεύτερου καταχωρητή
extension	1 bit	Είσοδος	Πληροφορία για extension
Shifting	1 bit	Είσοδος	Πληροφορία για shifting
Clk	1 bit	Είσοδος	Ρολόι
Immed	32 bits	Έξοδος	Immediate προς τις επόμενες βαθμίδες
RF_A	32 bits	Έξοδος	Αποτέλεσμα 1ου καταχωρητή
RF_B	32 bits	Έξοδος	Αποτέλεσμα 2ου καταχωρητή

Δ. Σχεδίαση & Υλοποίηση

Εάν θελήσουμε να αναπαραστήσουμε τις συνδέσεις του DECODE STAGE, το παρακάτω θα είναι το αποτέλεσμά μας:



Η υλοποίηση των 2 πολυπλεκτών δε χρειάζεται επεξήγηση καθώς ήταν πολύ απλή. Το αρχείο καταχωρητών χρησιμοποιήθηκε έτοιμο από τη προηγούμενη εργαστηριακή άσκηση και έχει ήδη μελετηθεί πλήρως.

Το καινούργιο και κάπως ενδιαφέρον τμήμα είναι αυτό του Immediate Extender. Χρειαστήκαμε 2 επιπλέον (1 bit) εισόδους του DECSTAGE (shifting, extension) για να έχουμε το κατάλληλο αποτέλεσμα αναλόγως εάν θέλουμε να εφαρμόσουμε ολίσθηση και zero-filling.

Δ. Κώδικας VHDL (Immediate Extender)

```
begin
im32out: process(imm16,extension,shifting)
      begin
            if (extension = '1') then
                  if (shifting = '1') then
                        imm32 <= STD_LOGIC_VECTOR(shift_left(signed((31
downto 16 => imm16(15)) & imm16),2));
                  else
                        imm32 <= (31 downto 16 => imm16(15)) & imm16;
                  end if;
            else
                  if (shifting = '1') then
                        imm32 <= STD_LOGIC_VECTOR(shift_left(signed((31
downto 16 => '0') & imm16),2));
                  else
                        imm32 <=(31 downto 16 => '0') & imm16;
                  end if;
            end if;
      end process;
end Behavioral;
```

Δ. Επαλήθευση Immediate Extender (Testbench)

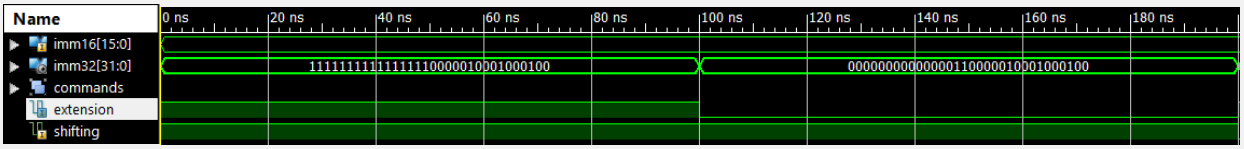
Επαληθεύσαμε με μια περίπτωση ανά συνδυασμό (μιας και δεν υπάρχουν corner cases) την παραπάνω λειτουργία:

```
begin
      imm16 <= "1100000100010001";
      extension <='1';
      shifting <='1';
      wait for 100 ns;

      imm16 <= "1100000100010001";
      extension <='0';
      shifting <='1';
      wait for 100 ns;

      imm16 <= "1100000100010001";
```

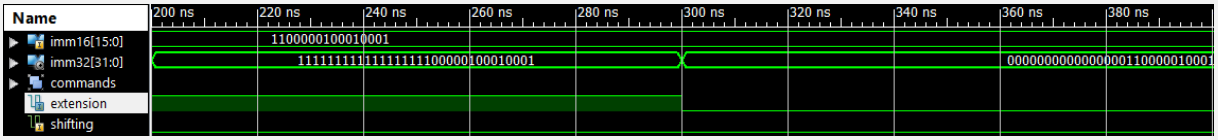
```
        extension <='1';
        shifting <='0';
        wait for 100 ns;

        imm16 <= "1100000100010001";
        extension <='0';
        shifting <='0';
        wait;
    end process;
```

⏱ Δ. Προσομοίωση Immediate Extender (Κυματομορφές)

0 ns – 200 ns:



200 ns – 400 ns



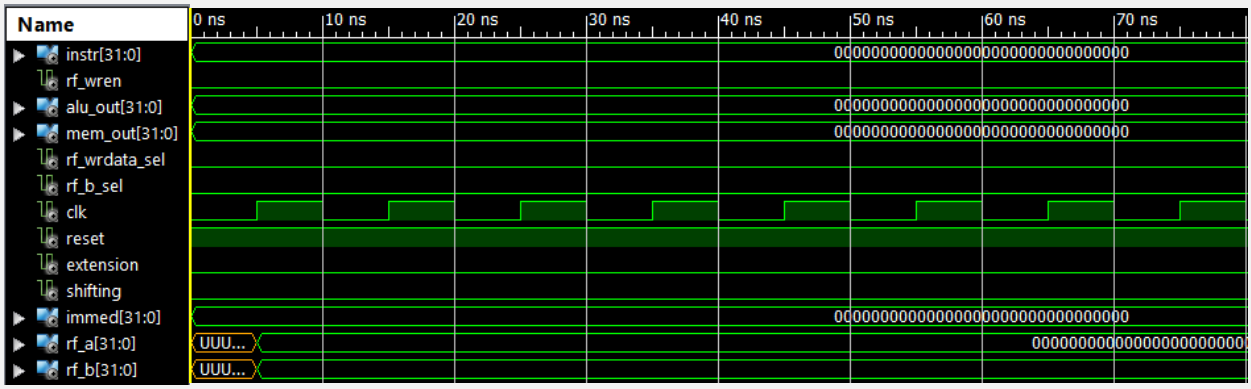
📋 Δ. Επαλήθευση DECSTAGE (Testbench)

Ο κώδικας του Testbench είναι πολύ εκτενές και προτιμήσαμε να μη τον παραθέσουμε στην αναφορά. Είναι διαθέσιμος στο αρχείο DESTAGE_test.vhd και περιέχει καθαρά σχόλια για τις περιπτώσεις που εξετάσαμε.

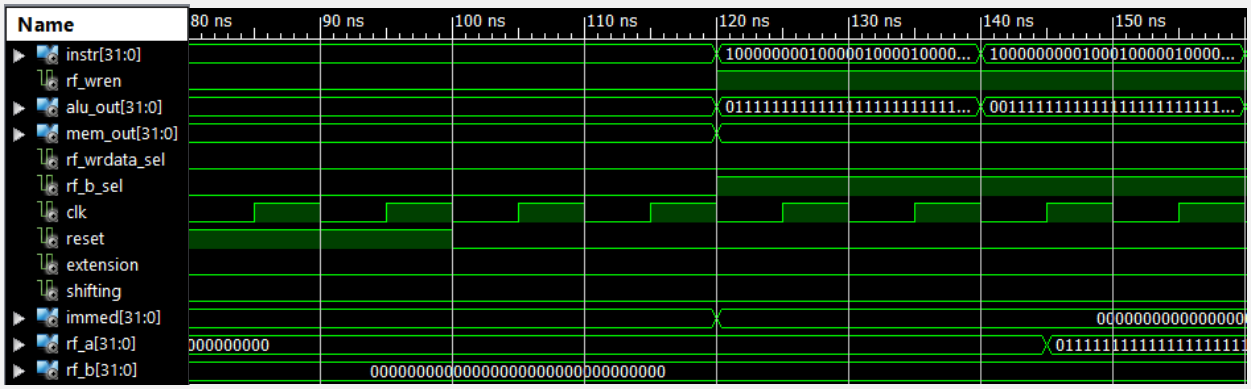
⏱ Δ. Προσομοίωση DECSTAGE (Κυματομορφές)

Το αποτέλεσμα των κυματομορφών φαίνεται παρακάτω:

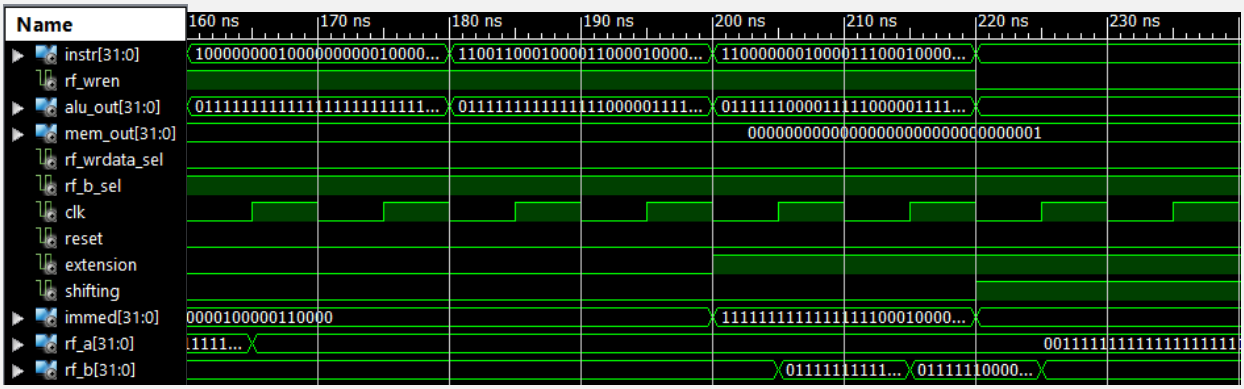
0 ns – 80 ns:



80 ns – 160 ns:



160 ns – 240 ns:



Ε – ΒΑΘΜΙΔΑ ΕΚΤΕΛΕΣΗΣ ΕΝΤΟΛΩΝ

Αυτή η βαθμίδα είναι υπεύθυνη για την εκτέλεση των εντολών και πράξεων.

Ε. Επισκόπηση

Βασικά components της βαθμίδας συνιστούν:

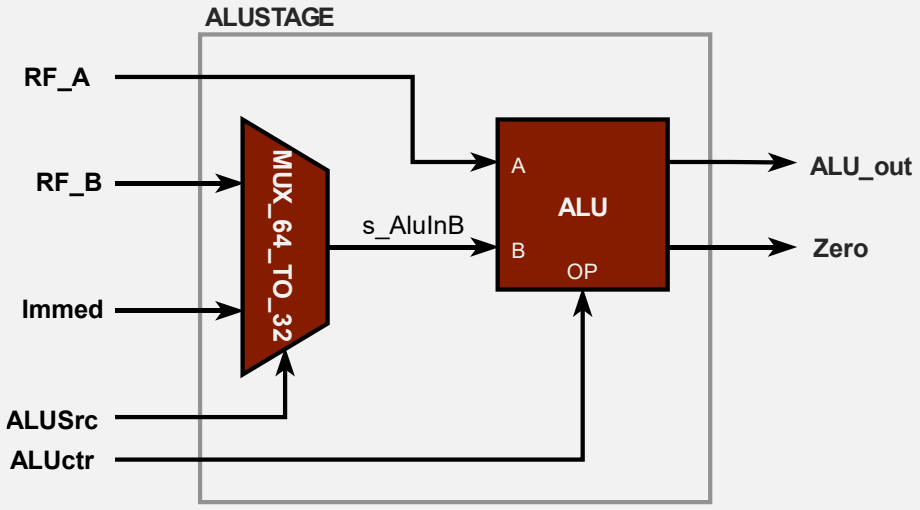
- Η ALU που χρησιμοποιήθηκε από το πρώτο εργαστήριο
- Ένας πολυπλέκτης που επιλέγει τον 2ο τελεστέο ανάμεσα σε Immed και Register.

Πίνακας εισόδων / εξόδων:

Όνομα Σήματος	Πλάτος Σήματος	Είδος Σήματος	Περιγραφή
RF_A	**32 bits**	**Είσοδος**	RF[rs]
RF_B	**32 bits**	**Είσοδος**	RF[rt] ή RF[rd]
Immed	**32 bits**	**Είσοδος**	Immediate
ALU_Bin_sel	**1 bit**	**Είσοδος**	Επιλογή 2ου τελεστέου ανάμεσα σε Immed/Reg
ALU_func	**4 bits**	**Είσοδος**	Πράξη ALU
ALU_out	**32 bits**	**Έξοδος**	Αποτέλεσμα ALU

Ε. Σχεδίαση & Υλοποίηση

Σχηματική αναπαράσταση του ALU STAGE:



Ο κώδικας είναι ουσιαστικά επανάληψη γνωστής ύλης VHDL και δε χρειάζεται να αναλυθεί κάποιο τμήμα του.

Ε. Επαλήθευση ALUSTAGE (Testbench)

Δημιουργήσαμε το ακόλουθο testbench για να επαληθεύσουμε τη λειτουργία της βαθμίδας:

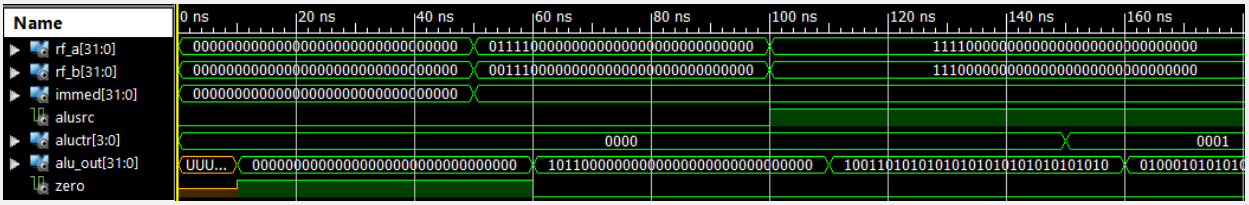
```
begin
	wait for 50 ns;
		--   ADD__SUB
		RF_A  <= "01111000000000000000000000000000";
		RF_B  <= "00111000000000000000000000000000";
		Immed <= "10101010101010101010101010101010";
		ALUSrc <= '0';
		ALUctr <= "0000";
		wait for 50  ns;

		--   ADD__SUB
		RF_A<= "11110000000000000000000000000000";
		RF_B <= "11100000000000000000000000000000";
		ALUSrc <= '1';
		ALUctr <= "0000";
		wait for 50  ns;
			ALUctr <= ALUctr +1;
		wait for 50  ns;

		--   ADD__SUB__AND__OR__NOT
		RF_A<= "10000000000000000000000000000000";
		RF_B <= "11111111111111111111111111111111";
		ALUctr <= "0000";
		wait for 50  ns;
		for i in 1 to 3 loop
			ALUctr <= ALUctr +1;
			wait for 50 ns;
		end loop;

		-- SHIFT_RIGHT ____ SHIFT_LEFT
		RF_A<= "10000000000000000000000000000001";
		ALUctr <= "1000";
		wait for 50 ns;
		for i in 1 to 2 loop
			ALUctr <= ALUctr +1;
			wait for 50 ns;
		end loop;

		-- ROTATE
		RF_A<= "00011010101010101010101010100011";
		ALUctr <= "1100";
		wait for 50 ns;
```

```
            for i in 1 to 2 loop
                  ALUctr <= ALUctr +1;
                  wait for 50 ns;
            end loop;
      wait;
end process;
```

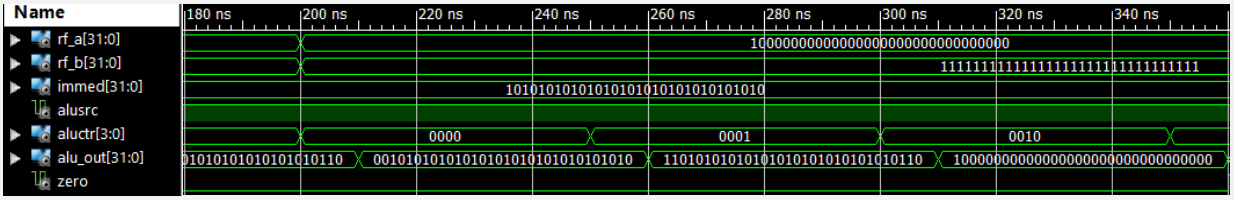
⏱ Ε. Προσομοίωση ALUSTAGE (Κυματομορφές)

Προσομοίωση που παράχθηκε σύμφωνα με το παραπάνω testbench:

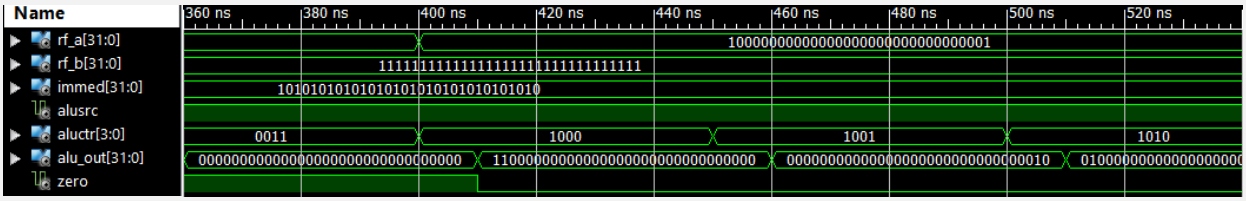
0 ns – 180 ns:



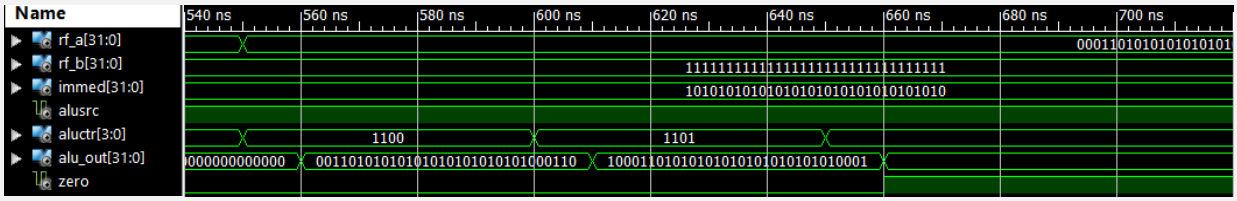
180 ns – 360 ns:



360 ns – 540 ns:



540 ns – 720 ns:



ΣΤ - ΒΑΘΜΙΔΑ ΠΡΟΣΒΑΣΗΣ ΜΝΗΜΗΣ

ΣΤ. Επισκόπηση

Η βαθμίδα πρόσβασης μνήμης περιέχει τον αυξητή διεύθυνσης ο οποίος ευθύνεται για τη σωστή ένταξη των δεδομένων στη μνήμη (με offset 0x400 σε σχέση με τις διευθύνσεις εντολών).

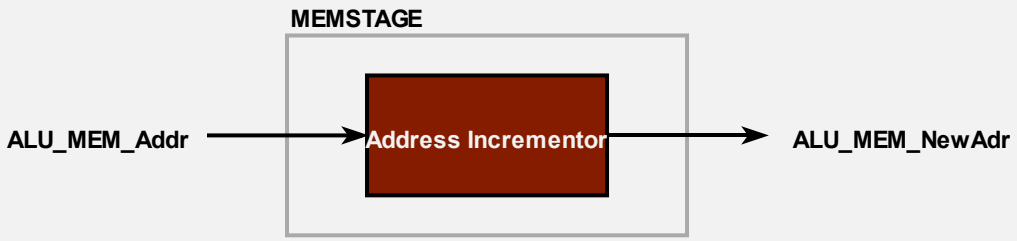
⚠ **Παρατήρηση:** Κατόπιν διόρθωσης της εκφώνησης, η μνήμη RAM πλέον δεν αποτελεί μέρος αυτής της βαθμίδας, παρά βρίσκεται σε υψηλότερο επίπεδο μαζί με τη βαθμίδα MEMSTAGE (Βλ. Ζ – ΒΑΘΜΙΔΑ ΟΛΙΚΗΣ ΜΝΗΜΗΣ).

Πίνακας Εισόδων / Εξόδων:

Όνομα Σήματος	Πλάτος Σήματος	Είδος Σήματος	Περιγραφή
MEM_DataIn	**32 bits**	**Είσοδος**	Αποτέλεσμα RF[rd] για αποθήκευση στη μνήμη για εντολές swap και sb, sw
MEM_DataOut	**32 bits**	**Έξοδος**	Δεδομένα που φορτώθηκαν από τη μνήμη προς εγγραφή σε καταχωρητή για εντολές lb, lw

ΣΤ. Σχεδίαση & Υλοποίηση

Σχηματική αναπαράσταση του MEMSTAGE:



Ο κώδικας είναι πολύ απλός, για αυτό και δε συμπεριλαμβάνεται στην αναφορά αυτή.

Testbench δε χρειάστηκε για αυτή τη βαθμίδα. Η λειτουργία της μαζί με τη μνήμη αξιολογούνται σε υψηλότερο επίπεδο.

Z - ΒΑΘΜΙΔΑ ΟΛΙΚΗΣ ΜΝΗΜΗΣ

Z. Επισκόπηση

Μετά την αλλαγή της εκφώνησης, αποφασίσαμε να κατασκευάσουμε ένα MEMORY_T module ως βαθμίδα του Datapath και να περιλαμβάνει το IFSTAGE, MEMSTAGE και τη μνήμη RAM. Αυτό δεν έχει κάποιο πρακτικό νόημα και έγινε μόνο για να υπάρχει κοινή μνήμη μεταξύ των βαθμίδων και γιατί βοηθάει στην επαλήθευσή του.

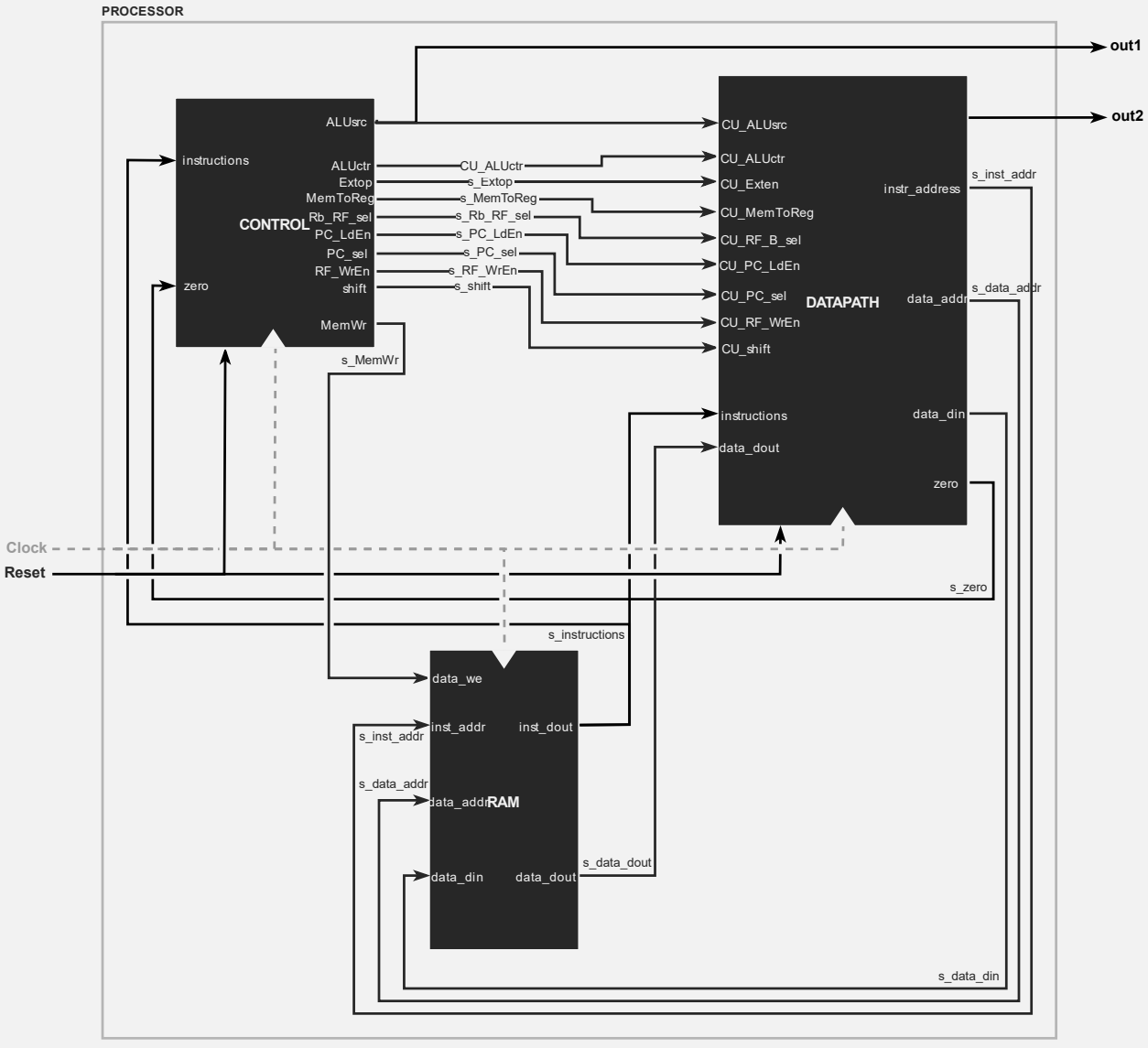
Z. Επαλήθευση MEMORY_T (Testbench)

```
begin
                reset <= '1';
      wait for 100 ns;
            reset      <= '0';
                data_we    <= '0';
         CU_PC_sel  <= '0';
         CU_PC_LdEn <= '0';
         s_ALU_out  <= "10101010100000000000001010101010";
         s_data_din <= "10101010100000000000000000000000";
      wait for clk_period*3;
                data_we    <= '0';
         CU_PC_sel  <= '1';
         CU_PC_LdEn <= '0';
         s_ALU_out  <= "10101010100000000000001010101010";
         s_data_din <= "10101010100000000000000000000000";
      wait for clk_period*3;
                data_we    <= '1';
         CU_PC_sel  <= '0';
         CU_PC_LdEn <= '0';
         s_ALU_out  <= "10101010100000000000001010101010";
         s_data_din <= "10101010100000000000000000000000";
             wait for clk_period*3;
                data_we    <= '0';
         CU_PC_sel  <= '1';
         CU_PC_LdEn <= '1';
         s_ALU_out  <= "10101010100000000000001010101010";
         s_data_din <= "10101010100000000000000000000000";
            wait for clk_period*3;
                data_we    <= '0';
         CU_PC_sel  <= '0';
         CU_PC_LdEn <= '0';
         s_ALU_out  <= "10101010100000000000001010101010";
         s_data_din <= "10101010100000000000000000000000";
      wait;
   end process;
```

H – HIGHER LEVEL BLOCK DIAGRAMS

Παρακάτω παρουσιάζεται ένα sneak – peek για τη δομή των βαθμίδων του επεξεργαστή όπως θα καταλήξουμε στις επόμενες εργαστηριακές ασκήσεις. Το Control Unit έχει δημιουργηθεί αλλά δεν έχει υλοποιηθεί σε αυτό το στάδιο.

PROCESSOR – TOP MODULE



Παρακάτω παρουσιάζεται ιδανικά πως θα κατασκευαστεί το Datapath στις επόμενες εργαστηριακές ασκήσεις έτσι ώστε η μνήμη RAM να μην αποτελεί μέρος του επεξεργαστή.

DATAPATH