



**Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών  
ΔΠΜΣ Ηλεκτρονικού Αυτοματισμού**

---

Όνομα: Μιχαήλ  
Επώνυμο: Παπαγεωργίου  
Α.Μ.: 7110132300201

---

**Σχεδίαση Ψηφιακών Συστημάτων**  
**Project 1 – Μεθοδολογία σχεδίασης επεξεργαστή ενός κύκλου**

---

# Περιεχόμενα

<b>0. ΕΙΣΑΓΩΓΗ .....</b>	<b>8</b>
<b>1. ΠΕΡΙΓΡΑΦΗ ΤΩΝ ΣΤΟΙΧΕΙΩΝ ΚΑΙ ΤΗΣ ΔΟΜΗΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ .....</b>	<b>9</b>
1.1 ΔΙΑΔΡΟΜΗ ΔΕΔΟΜΕΝΩΝ.....	9
1.1.1 ΜΕΤΡΗΤΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ.....	9
1.1.2 ΜΝΗΜΗ ΕΝΤΟΛΩΝ.....	10
1.1.3 ΑΘΡΟΙΣΤΗΣ PCPLUS4 .....	10
1.1.4 ΠΟΛΥΠΛΕΚΤΗΣ ΛΟΓΙΚΗΣ ΕΠΟΜΕΝΗΣ ΕΝΤΟΛΗΣ PCN.....	11
1.1.5 ΑΘΡΟΙΣΤΗΣ PCPLUS8 .....	11
1.1.6 ΠΟΛΥΠΛΕΚΤΗΣ A1 REG .....	11
1.1.7 ΠΟΛΥΠΛΕΚΤΗΣ A2 REG .....	12
1.1.8 ΠΟΛΥΠΛΕΚΤΗΣ A3 REG .....	12
1.1.9 ΠΟΛΥΠΛΕΚΤΗΣ WD3 REG .....	13
1.1.10 ΑΡΧΕΙΟ ΚΑΤΑΧΩΡΗΤΩΝ .....	13
1.1.11 EXTENDER .....	14
1.1.12 ΠΟΛΥΠΛΕΚΤΗΣ SRCB.....	14
1.1.13 ΜΟΝΑΔΑ ALU.....	15
1.1.14 ΚΑΤΑΧΩΡΗΤΕΣ ΚΑΤΑΣΤΑΣΗΣ.....	15
1.1.15 ΠΟΛΥΠΛΕΚΤΗΣ RESULT .....	16
1.1.16 ΜΝΗΜΗ ΔΕΔΟΜΕΝΩΝ .....	16
1.2 ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ .....	17
1.2.1 ΣΧΕΔΙΑΣΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ ΕΝΤΟΛΩΝ (INSTRDEC) .....	18
1.2.2 ΣΧΕΔΙΑΣΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ ΣΗΜΑΤΩΝ ΕΓΚΡΙΣΗΣ ΕΓΓΡΑΦΗΣ (WELOGIC).....	28
1.2.3 ΣΧΕΔΙΑΣΗ ΛΟΓΙΚΗΣ ΕΠΙΛΟΓΗΣ ΔΙΕΥΘΥΝΣΗΣ ΕΠΟΜΕΝΗΣ ΕΝΤΟΛΗΣ (PCLOGIC).....	30
1.2.4 ΣΧΕΔΙΑΣΗ ΛΟΓΙΚΗΣ ΕΛΕΓΧΟΥ ΣΥΝΘΗΚΗΣ (CONDLOGIC).....	31
1.3 ΕΠΕΞΕΡΓΑΣΤΗΣ .....	33
1.4 ALU .....	34
1.4.1 ΥΠΟΜΟΝΑΔΑ ADDSUB.....	34
1.4.2 ΥΠΟΜΟΝΑΔΑ MOVEOPERATIONS.....	35
1.4.3 ΥΠΟΜΟΝΑΔΑ LOGICALOPERATIONS .....	36
1.4.4 ΥΠΟΜΟΝΑΔΑ SHIFTOPERATIONS.....	38
1.4.5 ΥΠΟΜΟΝΑΔΑ NORR.....	40
1.4.6 ΧΡΗΣΗ ΠΟΡΩΝ .....	40
1.4.7 ΤΑΧΥΤΗΤΑ ΛΕΙΤΟΥΡΓΙΑΣ ALU .....	42
1.5 ΣΥΧΝΟΤΗΤΑ ΛΕΙΤΟΥΡΓΙΑΣ ΕΠΕΞΕΡΓΑΣΤΗ .....	44
<b>2. ΕΠΑΛΗΘΕΥΣΗ ΤΗΣ ΟΡΘΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑΣ.....</b>	<b>48</b>
2.1 ΔΗΜΙΟΥΡΓΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ ΣΕ ASSEMBLY .....	48
2.2 ΔΙΑΓΡΑΜΜΑΤΑ ΧΡΟΝΙΣΜΟΥ ALU.....	54
2.2.1 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ ROR .....	59
2.2.2 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ ASR .....	60
2.2.3 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ LSR .....	60
2.2.4 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ LSL .....	61
2.2.5 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ AND .....	62
2.2.6 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ OR .....	63
2.2.7 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ XOR.....	64
2.2.8 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ ADD .....	65
2.2.9 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ SUB.....	65
2.2.10 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ MOV .....	66
2.2.11 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ MVN .....	67
2.2.12 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ NOP .....	68

2.2.13	<i>ΣΧΟΛΙΑΣΜΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ ΧΡΟΝΙΣΜΟΥ ALU</i>	68
2.3	<i>ΔΙΑΓΡΑΜΜΑΤΑ ΧΡΟΝΙΣΜΟΥ ΕΠΕΞΕΡΓΑΣΤΗ</i>	69
2.3.1	<i>ΕΝΤΟΛΗ MOV R0, 20</i>	70
2.3.2	<i>ΕΝΤΟΛΗ MOV R1, R0</i>	70
2.3.3	<i>ΕΝΤΟΛΗ NOP</i>	71
2.3.4	<i>ΕΝΤΟΛΗ STR R1, [R0,2]</i>	72
2.3.5	<i>ΕΝΤΟΛΗ STR R1, [R0,-2]</i>	73
2.3.6	<i>ΕΝΤΟΛΗ LDR R2, [R0,2]</i>	74
2.3.7	<i>ΕΝΤΟΛΗ LDR R3, [R0,-2]</i>	75
2.3.8	<i>ΕΝΤΟΛΗ SUB R0, R1, 19</i>	76
2.3.9	<i>ΕΝΤΟΛΗ ADD R4, R1, 30</i>	77
2.3.10	<i>ΕΝΤΟΛΗ SUB R2, R0, R1</i>	78
2.3.11	<i>ΕΝΤΟΛΗ ADD R1, R0, R3</i>	78
2.3.12	<i>ΕΝΤΟΛΗ AND R3, R0, R1</i>	79
2.3.13	<i>ΕΝΤΟΛΗ AND R3, R1, 15</i>	80
2.3.14	<i>ΕΝΤΟΛΗ XOR R3, R0, R1</i>	81
2.3.15	<i>ΕΝΤΟΛΗ XOR R3, R1, 15</i>	82
2.3.16	<i>ΕΝΤΟΛΗ OR R3, R0, R1</i>	83
2.3.17	<i>ΕΝΤΟΛΗ OR R3, R1, 15</i>	84
2.3.18	<i>ΕΝΤΟΛΗ LSL R3, R1, 2</i>	85
2.3.19	<i>ΕΝΤΟΛΗ LSR R1, R3, 2</i>	86
2.3.20	<i>ΕΝΤΟΛΗ ASR R1, R2, 2</i>	87
2.3.21	<i>ΕΝΤΟΛΗ ROR R1, R3, 2</i>	88
2.3.22	<i>ΕΝΤΟΛΗ MVN R0, 2</i>	89
2.3.23	<i>ΕΝΤΟΛΗ MVN R4, R3</i>	90
2.3.24	<i>ΕΝΤΟΛΗ CMP R2, 15</i>	91
2.3.25	<i>ΕΝΤΟΛΗ BEQ COROUTINE_EQ</i>	92
2.3.26	<i>ΕΝΤΟΛΗ BLNE COROUTINE_NE</i>	93
2.3.27	<i>ΕΝΤΟΛΗ ADDS R0, R1, -1</i>	93
2.3.28	<i>ΕΝΤΟΛΗ BCC COROUTINE_CC</i>	94
2.3.29	<i>ΕΝΤΟΛΗ BCS COROUTINE_CS</i>	95
2.3.30	<i>ΕΝΤΟΛΗ SUBS R0, R2, R1</i>	96
2.3.31	<i>ΕΝΤΟΛΗ BPL COROUTINE_PL</i>	97
2.3.32	<i>ΕΝΤΟΛΗ BMI COROUTINE_MI</i>	98
2.3.33	<i>ΕΝΤΟΛΗ SUBS R0, R1, 21</i>	99
2.3.34	<i>ΕΝΤΟΛΗ BVS COROUTINE_VS</i>	99
2.3.35	<i>ΕΝΤΟΛΗ BVC COROUTINE_VC</i>	100
2.3.36	<i>ΕΝΤΟΛΗ BLS COROUTINE_HI</i>	101
2.3.37	<i>ΕΝΤΟΛΗ BHI COROUTINE_LS</i>	102
2.3.38	<i>ΕΝΤΟΛΗ ADDGE R4, R1, 30</i>	103
2.3.39	<i>ΕΝΤΟΛΗ ADDGT R4, R1, 30</i>	104
2.3.40	<i>ΕΝΤΟΛΗ ADDLT R4, R1, 30</i>	105
2.3.41	<i>ΕΝΤΟΛΗ MOVLE PC, R14</i>	105
2.3.42	<i>ΕΝΤΟΛΗ B MAIN</i>	106
2.3.43	<i>ΣΧΟΛΙΑΣΜΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ ΧΡΟΝΙΣΜΟΥ ΕΠΕΞΕΡΓΑΣΤΗ</i>	107

### 3. ΑΝΑΛΥΣΗ ΤΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΥΝΘΕΣΗΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

108

### 4. ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΑΣ ROR..... 112

4.1	ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΗΣ ROR	112
4.2	ΑΛΗΛΕΠΙΔΡΑΣΗ ΜΕ ΤΟ CONTROL UNIT	112

## Κατάλογος πινάκων

Πίνακας 1 – Είσοδος InstrDec - op .....	18
Πίνακας 2 – Είσοδος InstrDec - funct.....	18
Πίνακας 3 – Είσοδος InstrDec - sh.....	19
Πίνακας 4 – Είσοδος InstrDec - shamt5 .....	19
Πίνακας 5 – 'Εξοδος InstrDec - RegSrc.....	19
Πίνακας 6 – 'Εξοδος InstrDec - AluSrc.....	19
Πίνακας 7 – 'Εξοδος InstrDec - ImmSrc.....	19
Πίνακας 8 – 'Εξοδος InstrDec – ALUControl .....	20
Πίνακας 9 – 'Εξοδος InstrDec - MemtoReg .....	20
Πίνακας 10 – 'Εξοδος InstrDec - NoWrite_in.....	20
Πίνακας 11 – Πίνακας αποκωδικοποιητή εντολών .....	27
Πίνακας 12 – Είσοδος WELogic - op.....	28
Πίνακας 13 – Είσοδος WELogic - S/L .....	28
Πίνακας 14 – Είσοδος WELogic - 1L.....	28
Πίνακας 15 – Πίνακας αληθείας αποκωδικοποιητή σημάτων έγκρισης εγγραφής.....	29
Πίνακας 16 – Είσοδος PCLogic - op.....	30
Πίνακας 17 – Πίνακας αληθείας λογικής επιλογής διεύθυνσης επόμενης εντολής .....	31
Πίνακας 18 – Είσοδος CONDLogic - cond .....	31
Πίνακας 19 – Είσοδος CONDLogic - σημαίες .....	32
Πίνακας 20 – Πίνακας αληθείας λογικής ελέγχου συνθήκης (CONDLogic) .....	33
Πίνακας 21 - Εντολές και αντίστοιχες τιμές σημάτων κατά τη δοκιμή του επεξεργαστή .....	53
Πίνακας 22 – Εντολη ROR.....	112

## Κατάλογος εικόνων

Εικόνα 1 – Διάγραμμα RTL της διαδρομής δεδομένων .....	9
Εικόνα 2 – Διάγραμμα RTL του μετρητή προγράμματος .....	10
Εικόνα 3 – Διάγραμμα RTL της μνήμης εντολών .....	10
Εικόνα 4 – Διάγραμμα RTL του αθροιστή PCPlus4 .....	10
Εικόνα 5 – Διάγραμμα RTL του πολυπλέκτη λογικής επόμενης εντολής PCN .....	11
Εικόνα 6 – Διάγραμμα RTL του αθροιστή PCPlus8 .....	11
Εικόνα 7 – Διάγραμμα RTL του πολυπλέκτη A1 REG .....	12
Εικόνα 8 – Διάγραμμα RTL του πολυπλέκτη A' 2 REG .....	12
Εικόνα 9 – Διάγραμμα RTL του πολυπλέκτη A3 REG .....	13
Εικόνα 10 – Διάγραμμα RTL του πολυπλέκτη WD3 REG .....	13
Εικόνα 11 – Διάγραμμα RTL του αρχείου καταχωρητών .....	14
Εικόνα 12 – Διάγραμμα RTL της μονάδας επέκτασης .....	14
Εικόνα 13 – Διάγραμμα RTL του πολυπλέκτη SrcB .....	14
Εικόνα 14 – Διάγραμμα RTL της μονάδας ALU .....	15
Εικόνα 15 – Διάγραμμα RTL των καταχωρητών κατάστασης .....	16
Εικόνα 16 – Διάγραμμα RTL του πολυπλέκτη Result .....	16
Εικόνα 17 – Διάγραμμα RTL της μνήμης δεδομένων .....	17
Εικόνα 18 – Διάγραμμα RTL της μονάδας ελέγχου .....	17
Εικόνα 19 – Διάγραμμα RTL του αποκωδικοποιητή εντολών .....	21
Εικόνα 20 – Διάγραμμα RTL του αποκωδικοποιητή σημάτων έγκρισης εγγραφής .....	29
Εικόνα 21 – Διάγραμμα RTL της λογικής επιλογής διεύθυνσης επόμενης εντολής .....	30
Εικόνα 22 – Διάγραμμα RTL της λογικής ελέγχου συνθήκης .....	32
Εικόνα 23 – Διάγραμμα RTL επεξεργαστή .....	33
Εικόνα 24 – Διάγραμμα RTL της μονάδας ALU .....	34
Εικόνα 25 – Διάγραμμα RTL της μονάδας AddSub .....	35
Εικόνα 26 – Διάγραμμα RTL της μονάδας add .....	35

Εικόνα 27 – Διάγραμμα RTL της μονάδας MoveOperationsComponent .....	36
Εικόνα 28 – Διάγραμμα RTL των μονάδων MOV και MVN .....	36
Εικόνα 29 – Διάγραμμα RTL των μονάδας LogicalOperations .....	37
Εικόνα 30 – Διάγραμμα RTL των μονάδων Andd, Orr και Xorr .....	38
Εικόνα 31 – Διάγραμμα RTL των μονάδας LogicalOperations .....	39
Εικόνα 32 – Διάγραμμα RTL των μονάδων ASR, LSL, LSR .....	39
Εικόνα 33 – Διάγραμμα RTL των μονάδων ROR .....	40
Εικόνα 34 – Διάγραμμα RTL των μονάδας NORR .....	40
Εικόνα 35 – Χρήση πόρων από την ALU .....	41
Εικόνα 36 – Ποσοστό χρήσης πόρων από την ALU .....	41
Εικόνα 37 – Χρόνος διάδοσης ALU .....	42
Εικόνα 38 – Μονοπάτι μέγιστου χρόνου διάδοσης στην ALU .....	43
Εικόνα 39 – Χρόνος μόλυνσης ALU .....	43
Εικόνα 40 – Μέγιστη συχνότητα λειτουργίας .....	45
Εικόνα 41 – Design Timing Summary .....	45
Εικόνα 42 – Διάγραμμα χειρότερης κρίσιμης διαδρομής .....	45
Εικόνα 43 – Μέγιστοι χρόνοι διάδοσης επεξεργαστή .....	46
Εικόνα 44 – Διαδρομή με τον ελάχιστο χρόνο διάδοσης επεξεργαστή .....	47
Εικόνα 45 – Ελάχιστοι χρόνοι διάδοσης επεξεργαστή .....	47
Εικόνα 46 – Αποτέλεσμα εντολής ROR σε Behavioral Simulation .....	59
Εικόνα 47 – Αποτέλεσμα εντολής ROR σε Synthesis Simulation .....	59
Εικόνα 48 – Αποτέλεσμα εντολής ROR σε Implementation Simulation .....	59
Εικόνα 49 – Αποτέλεσμα εντολής ASR σε Behavioral Simulation .....	60
Εικόνα 50 – Αποτέλεσμα εντολής ASR σε Synthesis Simulation .....	60
Εικόνα 51 – Αποτέλεσμα εντολής ASR σε Implementation Simulation .....	60
Εικόνα 52 – Αποτέλεσμα εντολής LSR σε Behavioral Simulation .....	61
Εικόνα 53 – Αποτέλεσμα εντολής LSR σε Synthesis Simulation .....	61
Εικόνα 54 – Αποτέλεσμα εντολής LSR σε Implementation Simulation .....	61
Εικόνα 55 – Αποτέλεσμα εντολής LSL σε Behavioral Simulation .....	62
Εικόνα 56 – Αποτέλεσμα εντολής LSL σε Synthesis Simulation .....	62
Εικόνα 57 – Αποτέλεσμα εντολής LSL σε Implementation Simulation .....	62
Εικόνα 58 – Αποτέλεσμα εντολής AND σε Behavioral Simulation .....	62
Εικόνα 59 – Αποτέλεσμα εντολής AND σε Synthesis Simulation .....	63
Εικόνα 60 – Αποτέλεσμα εντολής AND σε Implementation Simulation .....	63
Εικόνα 61 – Αποτέλεσμα εντολής OR σε Behavioral Simulation .....	63
Εικόνα 62 – Αποτέλεσμα εντολής OR σε Synthesis Simulation .....	63
Εικόνα 63 – Αποτέλεσμα εντολής OR σε Implementation Simulation .....	64
Εικόνα 64 – Αποτέλεσμα εντολής XOR σε Behavioral Simulation .....	64
Εικόνα 65 – Αποτέλεσμα εντολής XOR σε Synthesis Simulation .....	64
Εικόνα 66 – Αποτέλεσμα εντολής XOR σε Implementation Simulation .....	64
Εικόνα 67 – Αποτέλεσμα εντολής ADD σε Behavioral Simulation .....	65
Εικόνα 68 – Αποτέλεσμα εντολής ADD σε Synthesis Simulation .....	65
Εικόνα 69 – Αποτέλεσμα εντολής ADD σε Implementation Simulation .....	65
Εικόνα 70 – Αποτέλεσμα εντολής SUB σε Behavioral Simulation .....	66
Εικόνα 71 – Αποτέλεσμα εντολής SUB σε Synthesis Simulation .....	66
Εικόνα 72 – Αποτέλεσμα εντολής SUB σε Implementation Simulation .....	66
Εικόνα 73 – Αποτέλεσμα εντολής MOV σε Behavioral Simulation .....	66
Εικόνα 74 – Αποτέλεσμα εντολής MOV σε Synthesis Simulation .....	67
Εικόνα 75 – Αποτέλεσμα εντολής MOV σε Implementation Simulation .....	67
Εικόνα 76 – Αποτέλεσμα εντολής MVN σε Behavioral Simulation .....	67
Εικόνα 77 – Αποτέλεσμα εντολής MVN σε Synthesis Simulation .....	67
Εικόνα 78 – Αποτέλεσμα εντολής MVN σε Implementation Simulation .....	68
Εικόνα 79 – Αποτέλεσμα εντολής NOP σε Behavioral Simulation .....	68

Εικόνα 80 – Αποτέλεσμα εντολής NOP σε Synthesis Simulation .....	68
Εικόνα 81 – Αποτέλεσμα εντολής NOP σε Implementation Simulation.....	68
Εικόνα 82 – Αποτέλεσμα εντολής MOV R0, 20 σε Behavioral Simulation .....	70
Εικόνα 83 – Αποτέλεσμα εντολής MOV R0, 20 σε Implementation Simulation .....	70
Εικόνα 84 – Αποτέλεσμα εντολής MOV R1, R0 σε Behavioral Simulation .....	71
Εικόνα 85 – Αποτέλεσμα εντολής MOV R1, R0 σε Implementation Simulation .....	71
Εικόνα 86 – Αποτέλεσμα εντολής NOP σε Behavioral Simulation.....	71
Εικόνα 87 – Αποτέλεσμα εντολής MOV R1, R0 σε Implementation Simulation .....	72
Εικόνα 88 – Αποτέλεσμα εντολής STR R1, [R0,2] σε Behavioral Simulation .....	72
Εικόνα 89 – Αποτέλεσμα εντολής STR R1, [R0,2] σε Implementation Simulation .....	73
Εικόνα 90 – Αποτέλεσμα εντολής STR R1, [R0,-2] σε Behavioral Simulation.....	73
Εικόνα 91 – Αποτέλεσμα εντολής STR R1, [R0,-2] σε Implementation Simulation.....	74
Εικόνα 92 – Αποτέλεσμα εντολής LDR R2, [R0,2] σε Behavioral Simulation.....	74
Εικόνα 93 – Αποτέλεσμα εντολής LDR R2, [R0,2] σε Implementation Simulation.....	75
Εικόνα 94 – Αποτέλεσμα εντολής LDR R2, [R0,-2] σε Behavioral Simulation .....	75
Εικόνα 95 – Αποτέλεσμα εντολής LDR R2, [R0,-2] σε Implementation Simulation.....	76
Εικόνα 96 – Αποτέλεσμα εντολής SUB R0, R1, 19 σε Behavioral Simulation .....	76
Εικόνα 97 – Αποτέλεσμα εντολής SUB R0, R1, 19 σε Implementation Simulation .....	77
Εικόνα 98 – Αποτέλεσμα εντολής ADD R4, R1, 30 σε Behavioral Simulation .....	77
Εικόνα 99 – Αποτέλεσμα εντολής ADD R4, R1, 30 σε Implementation Simulation .....	77
Εικόνα 100 – Αποτέλεσμα εντολής SUB R2, R0, R1 σε Behavioral Simulation .....	78
Εικόνα 101 – Αποτέλεσμα εντολής SUB R2, R0, R1 σε Implementation Simulation .....	78
Εικόνα 102 – Αποτέλεσμα εντολής ADD R1, R0, R3 σε Behavioral Simulation .....	79
Εικόνα 103 – Αποτέλεσμα εντολής ADD R1, R0, R3 σε Implementation Simulation .....	79
Εικόνα 104 – Αποτέλεσμα εντολής AND R3, R0, R1 σε Behavioral Simulation .....	80
Εικόνα 105 – Αποτέλεσμα εντολής AND R3, R0, R1 σε Implementation Simulation .....	80
Εικόνα 106 – Αποτέλεσμα εντολής AND R3, R1, 15 σε Behavioral Simulation .....	81
Εικόνα 107 – Αποτέλεσμα εντολής AND R3, R1, 15 σε Implementation Simulation .....	81
Εικόνα 108 – Αποτέλεσμα εντολής XOR R3, R0, R1 σε Behavioral Simulation .....	82
Εικόνα 109 – Αποτέλεσμα εντολής XOR R3, R0, R1 σε Implementation Simulation .....	82
Εικόνα 110 – Αποτέλεσμα εντολής XOR R3, R1, 15 σε Behavioral Simulation .....	83
Εικόνα 111 – Αποτέλεσμα εντολής XOR R3, R1, 15 σε Implementation Simulation .....	83
Εικόνα 112 – Αποτέλεσμα εντολής OR R3, R0, R1, σε Behavioral Simulation .....	84
Εικόνα 113 – Αποτέλεσμα εντολής OR R3, R0, R1, σε Implementation Simulation .....	84
Εικόνα 114 – Αποτέλεσμα εντολής OR R3, R1, 15 σε Behavioral Simulation .....	85
Εικόνα 115 – Αποτέλεσμα εντολής OR R3, R1, 15 σε Implementation Simulation .....	85
Εικόνα 116 – Αποτέλεσμα εντολής LSL R3, R1, 2 σε Behavioral Simulation .....	86
Εικόνα 117 – Αποτέλεσμα εντολής LSL R3, R1, 2 σε Implementation Simulation .....	86
Εικόνα 118 – Αποτέλεσμα εντολής LSR R1, R3, 2 σε Behavioral Simulation .....	87
Εικόνα 119 – Αποτέλεσμα εντολής LSR R1, R3, 2 σε Implementation Simulation .....	87
Εικόνα 120 – Αποτέλεσμα εντολής ASR R1, R2, 2 σε Behavioral Simulation .....	88
Εικόνα 121 – Αποτέλεσμα εντολής ASR R1, R2, 2 σε Implementation Simulation .....	88
Εικόνα 122 – Αποτέλεσμα εντολής ROR R1, R3, 2 σε Behavioral Simulation .....	89
Εικόνα 123 – Αποτέλεσμα εντολής ROR R1, R3, 2 σε Implementation Simulation .....	89
Εικόνα 124 – Αποτέλεσμα εντολής MVN R0, 2 σε Behavioral Simulation.....	90
Εικόνα 125 – Αποτέλεσμα εντολής MVN R0, 2 σε Implementation Simulation.....	90
Εικόνα 126 – Αποτέλεσμα εντολής MVN R4, R3 σε Behavioral Simulation .....	90
Εικόνα 127 – Αποτέλεσμα εντολής MVN R4, R3 σε Implementation Simulation.....	91
Εικόνα 128 – Αποτέλεσμα εντολής CMP R2, 15 σε Behavioral Simulation .....	91
Εικόνα 129 – Αποτέλεσμα εντολής CMP R2, 15 σε Implementation Simulation .....	92
Εικόνα 130 – Αποτέλεσμα εντολής BEQ COROUTINE_EQ σε Behavioral Simulation.....	92
Εικόνα 131 – Αποτέλεσμα εντολής BEQ COROUTINE_EQ σε Implementation Simulation.....	92
Εικόνα 132 – Αποτέλεσμα εντολής BLNE COROUTINE_NE σε Behavioral Simulation .....	93

Εικόνα 133 – Αποτέλεσμα εντολής BLNE COROUTINE_NE σε Implementation Simulation .....	93
Εικόνα 134 – Αποτέλεσμα εντολής ADDS R0, R1, -1 σε Behavioral Simulation .....	94
Εικόνα 135 – Αποτέλεσμα εντολής ADDS R0, R1, -1 σε Implementation Simulation .....	94
Εικόνα 136 – Αποτέλεσμα εντολής BCC COROUTINE_CC σε Behavioral Simulation .....	94
Εικόνα 137 – Αποτέλεσμα εντολής BCC COROUTINE_CC σε Implementation Simulation .....	95
Εικόνα 138 – Αποτέλεσμα εντολής BCS COROUTINE_CS σε Behavioral Simulation .....	95
Εικόνα 139 – Αποτέλεσμα εντολής BCS COROUTINE_CS σε Implementation Simulation .....	96
Εικόνα 140 – Αποτέλεσμα εντολής SUBS R0, R2, R1 σε Behavioral Simulation .....	96
Εικόνα 141 – Αποτέλεσμα εντολής SUBS R0, R2, R1 σε Implementation Simulation .....	97
Εικόνα 142 – Αποτέλεσμα εντολής BPL COROUTINE_PL σε Behavioral Simulation .....	97
Εικόνα 143 – Αποτέλεσμα εντολής BPL COROUTINE_PL σε Implementation Simulation .....	98
Εικόνα 144 – Αποτέλεσμα εντολής BMI COROUTINE_MI σε Behavioral Simulation .....	98
Εικόνα 145 – Αποτέλεσμα εντολής BMI COROUTINE_MI σε Implementation Simulation .....	98
Εικόνα 146 – Αποτέλεσμα εντολής SUBS R0, R1, 21 σε Behavioral Simulation .....	99
Εικόνα 147 – Αποτέλεσμα εντολής SUBS R0, R1, 21 σε Implementation Simulation .....	99
Εικόνα 148 – Αποτέλεσμα εντολής BVS COROUTINE_VS σε Behavioral Simulation .....	100
Εικόνα 149 – Αποτέλεσμα εντολής BVS COROUTINE_VS σε Implementation Simulation .....	100
Εικόνα 150 – Αποτέλεσμα εντολής BVC COROUTINE_VC σε Behavioral Simulation .....	101
Εικόνα 151 – Αποτέλεσμα εντολής BVC COROUTINE_VC σε Implementation Simulation .....	101
Εικόνα 152 – Αποτέλεσμα εντολής BLS COROUTINE_HI σε Behavioral Simulation .....	101
Εικόνα 153 – Αποτέλεσμα εντολής BLS COROUTINE_HI σε Implementation Simulation .....	102
Εικόνα 154 – Αποτέλεσμα εντολής BHI COROUTINE_LS σε Behavioral Simulation .....	102
Εικόνα 155 – Αποτέλεσμα εντολής BHI COROUTINE_LS σε Implementation Simulation .....	103
Εικόνα 156 – Αποτέλεσμα εντολής ADDGE R4, R1, 30 σε Behavioral Simulation .....	103
Εικόνα 157 – Αποτέλεσμα εντολής ADDGE R4, R1, 30 σε Implementation Simulation .....	104
Εικόνα 158 – Αποτέλεσμα εντολής ADDGT R4, R1, 30 σε Behavioral Simulation .....	104
Εικόνα 159 – Αποτέλεσμα εντολής ADDGT R4, R1, 30 σε Implementation Simulation .....	104
Εικόνα 160 – Αποτέλεσμα εντολής ADDLT R4, R1, 30 σε Behavioral Simulation .....	105
Εικόνα 161 – Αποτέλεσμα εντολής ADDLT R4, R1, 30 σε Implementation Simulation .....	105
Εικόνα 162 – Αποτέλεσμα εντολής MOVLE PC, R14 σε Behavioral Simulation .....	106
Εικόνα 163 – Αποτέλεσμα εντολής MOVLE PC, R14 σε Implementation Simulation .....	106
Εικόνα 164 – Αποτέλεσμα εντολής B MAIN σε Behavioral Simulation .....	106
Εικόνα 165 – Αποτέλεσμα εντολής B MAIN σε Implementation Simulation .....	106
Εικόνα 166 – Καρτέλα Project Summary .....	108
Εικόνα 167 – Καρτέλα Project Utilization – χρησιμοποιούμενοι πόροι .....	108
Εικόνα 168 – Κατανομή Slice Registers .....	109
Εικόνα 169 – Κατανομή LUT as Memory .....	109
Εικόνα 170 – Κατανομή πόρων μεταξύ Datapath και Control .....	110
Εικόνα 171 – Ποσοστό χρησιμοποιούμενων πόρων .....	110
Εικόνα 172 – Καταναλισκόμενη ενέργεια .....	111

## **Ο. ΕΙΣΑΓΩΓΗ**

Σκοπός της εργασίας είναι η προσομοίωση ενός επεξεργαστή αρχιτεκτονικής ARM σε FPGA με χρήση της γλώσσα περιγραφής υλικού VHDL. Ο επεξεργαστής θα είναι 32bit, ενός κύκλου και θα μπορεί να υλοποιήσει υποσύνολο των εντολών που μπορεί να εκτελέσει ένα ARM επεξεργαστής.

Αρχικά θα γίνει η σχεδίαση των ψηφιακών δομικών στοιχείων που αποτελούν τη διαδρομή δεομένων και της διαδρομή δεδομένων. Έπειτα γίνεται η σχεδίαση της μονάδας ελέγχου και τέλος του επεξεργαστή.

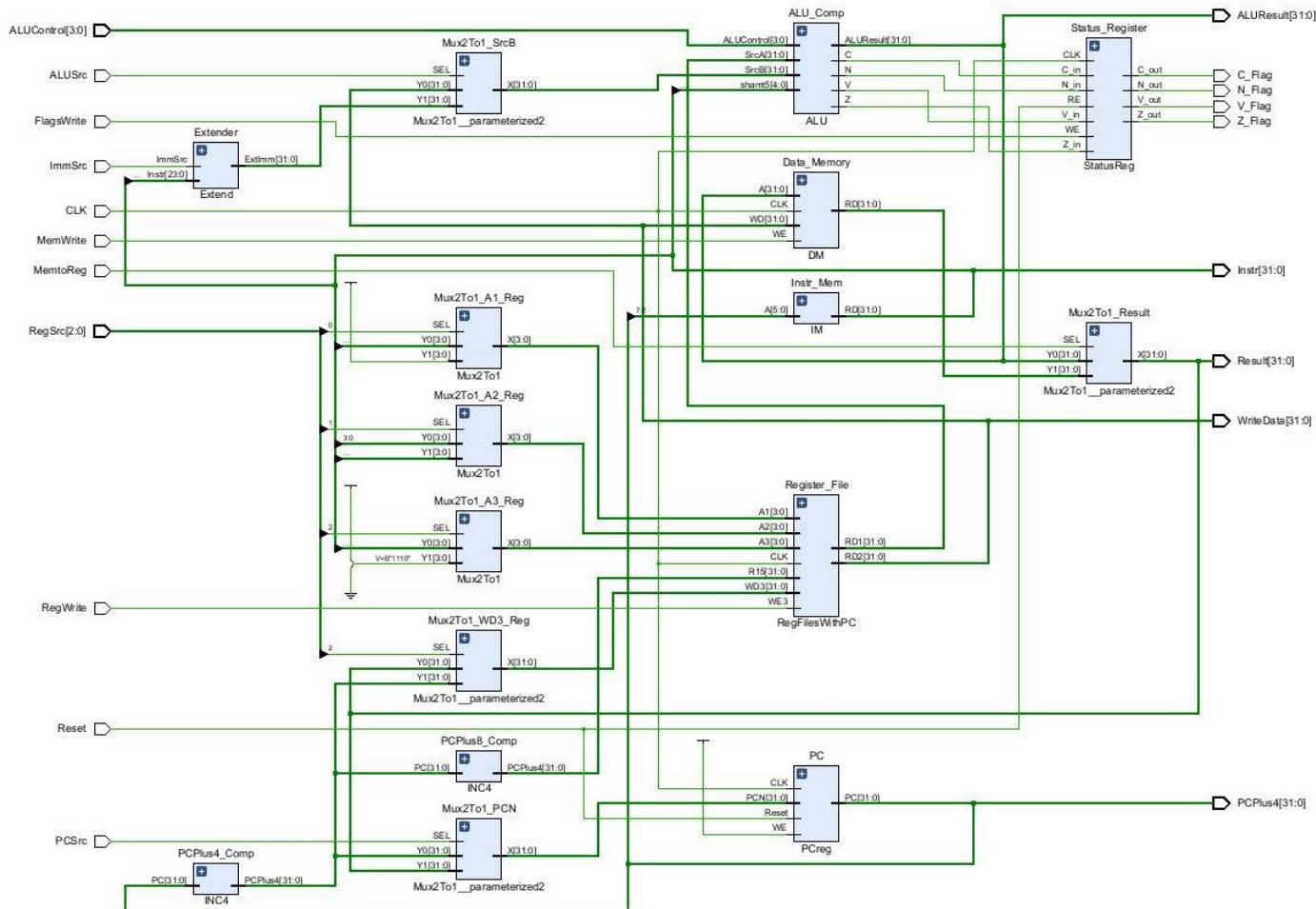
Αφού υλοποιηθεί ο επεξεργαστής γίνεται η επαλήθευση της ορθής σχεδίασης της μονάδας ALU, του αρχείου καταχωρητών, της μονάδας ελέγχου και τέλος του επεξεργαστή.

# 1. ΠΕΡΙΓΡΑΦΗ ΤΩΝ ΣΤΟΙΧΕΙΩΝ ΚΑΙ ΤΗΣ ΔΟΜΗΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

Θα γίνει περιγραφή της διαδρομής δεδομένων, της μονάδας ελέγχου, του επεξεργαστή, της ALU καθώς και της μέγιστης συχνότητας λειτουργίας του επεξεργαστή.

## 1.1 ΔΙΑΔΡΟΜΗ ΔΕΔΟΜΕΝΩΝ

Το σχηματικό διάγραμμα σε επίπεδο RTL φαίνεται στην παρακάτω εικόνα.



**Εικόνα 1 – Διάγραμμα RTL της διαδρομής δεδομένων**

### 1.1.1 ΜΕΤΡΗΤΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Η εκτέλεση κάθε εντολής ξεκινά από τον μετρητή προγράμματος ο οποίος αποθηκεύει την διεύθυνση της επόμενης εντολής που θα εκτελεστεί. Για την υλοποίηση του μετρητή χρησιμοποιείται παραμετροποιήμενος καταχωρητής με reset και write enable. Άλλες είσοδοι πέρα από το write enable και το reset επίσης είναι το σήμα του ρολογιού και η διεύθυνση της επόμενης εντολής.

Η είσοδος του **ρολογιού** είναι συνδεδεμένη στο σήμα ρολογιού του επεξεργαστή.

Όταν κατά την ανερχόμενη ακμή του ρολογιού το **reset** είναι '1' τότε θέτει σαν διεύθυνση επόμενης εντολής την τιμή '0' η οποία είναι η διεύθυνση της πρώτης εντολής που εκτελείται μετά από reset. Ομοίως με το ρολόι και το reset είναι συνδεδεμένο με το reset του γενικού κυκλώματος.

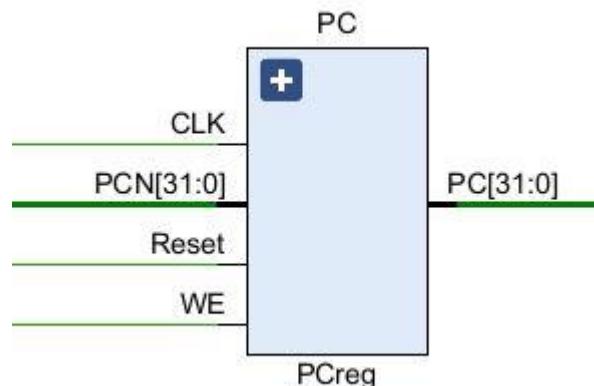
Όταν κατά την ανερχόμενη ακμή του ρολογιού το **write enable** είναι '1' τότε η τιμή στην είσοδο του καταχωρητή εγγράφεται σε αυτόν και διατηρείται και στην έξοδό του, μέχρι την επόμενη εγγραφή.

Τέλος όπως αναφέρθηκε και στην προηγούμενη παράγραφο η είσοδος **PCN** εγγράφεται στην έξοδο **PC**.

Ο PC είναι συνδεδεμένος στην είσοδο του με τον πολυπλέκτη **PCN** μέσω του οποίου γίνεται η επιλογή της πηγής της επόμενης εντολής, δηλαδή εάν αυτή θα προέρχεται από ALU ή τον αθροιστή **PCPlus4**.

Στην έξοδο του ο PC είναι συνδεδεμένος με την μνήμη δεδομένων όπου πηγαίνει και η διέυθυνση της επόμενης εντολής ώστε αυτή να διαβαστεί και με τον αθροιστή **PCPlus4** ο οποίος αυξάνει την τιμή της επόμενης εντολής κατά 4 byte.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του μετρητή προγράμματος.



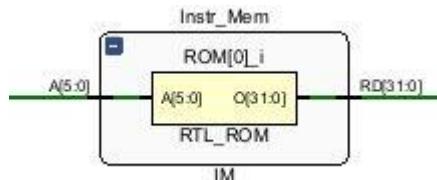
**Εικόνα 2 – Διάγραμμα RTL του μετρητή προγράμματος**

### 1.1.2 ΜΝΗΜΗ ΕΝΤΟΛΩΝ

Στην μνήμη εντολών είναι αποθηκευμένες οι προς εκτέλεση εντολές. Η μνήμη είναι ένας παραμετροποιημένος πίνακας καταχωρητών. Στην συγκεκριμένη περίπτωση περιέχει 64 καταχωρήσεις λέξεων των 32 bit.

Η είσοδος **A** της μνήμης είναι η διεύθυνση της επόμενης εντολής και η έξοδος **RD** είναι η προς εκτέλεση εντολή. Δεν υπάρχει είσοδος ρολογιού στη ROM και ως εκ τούτου το διάβασμα γίνεται ασύγχρονα.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μνήμης εντολών.



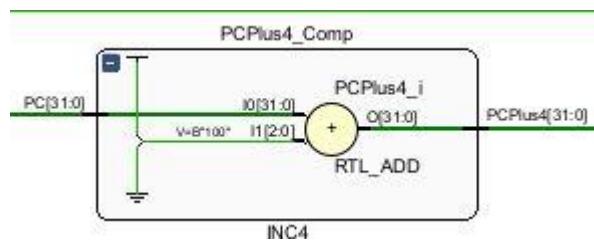
**Εικόνα 3 – Διάγραμμα RTL της μνήμης εντολών**

### 1.1.3 ΑΘΡΟΙΣΤΗΣ PCPLUS4

Ο αθροιστής PCPlus4 αυξάνει κατά 4 byte την διεύθυνση για τον υπολογισμό της επόμενης εντολής. Στην αρχιτεκτονική ARM η μνήμη είναι byte addressable και το μήκος λέξης - και συνεπώς το μέγεθος μίας εντολής – είναι 32 bit. Η αύξηση του περιεχομένου του PC κατά 4 δίνει τη διεύθυνση της επόμενης προς εκτέλεση εντολής, αν το πρόγραμμα πρέπει να συνεχίσει να εκτελείται σειριακά.

Η είσοδος **PC** είναι η διεύθυνση της εντολής που εκτελείται. Η έξοδος **PCPlus4** είναι διεύθυνση της επόμενης εντολής.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του αθροιστή PCPlus4.



**Εικόνα 4 – Διάγραμμα RTL του αθροιστή PCPlus4**

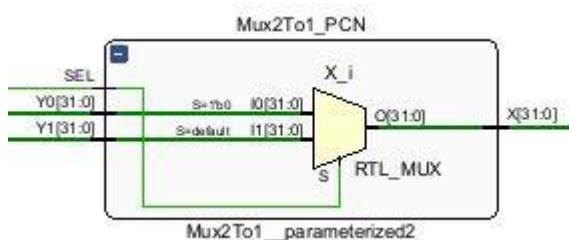
#### 1.1.4 ΠΟΛΥΠΛΕΚΤΗΣ ΛΟΓΙΚΗΣ ΕΠΟΜΕΝΗΣ ΕΝΤΟΛΗΣ PCN

Για την επιλογή της διεύθυνσης της επόμενης εντολής που πρόκειται να εκτελεστεί χρησιμοποιείται παραμετροποιημένος πολυπλέκτης 2 σε 1 των N (N=32) bit. Η επιλογή γίνεται μεταξύ της του αθροιστή **PCPlus4**, όταν έχουμε σειριακή ροή εντολών ή όταν δεν ικανοποιείται η συνθήκη κατά την εκτέλεση εντολής υπό συνθήκη και του **result** όποτε εκτελείται εντολή διακλάδωσης είτε όταν έχουμε εντολής επεξεργασίας δεδομένων ή εντολή LDR με καταχωρητή προορισμό τον R15.

Πέρα από τις εισόδου PCPlus4 και result υπάρχει και η είσοδος **PCSrc** μέσω της οποία γίνεται η επιλογή της εξόδου. Η ανάλυση της τιμής που παίρνει η είσοδος PCSrc θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου.

Όπως αναφέρθηκε και παραπάνω ο πολυπλέκτης λογικής επόμενης εντολής συνδέεται στην έξοδο τους με τον μετρητή προγράμματος. Στις εισόδους του συνδέεται με τον αθροιστή PCPlus4 καθώς και τον πολυπλέκτη **Result**.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του πολυπλέκτη λογικής επόμενης εντολής PCN.



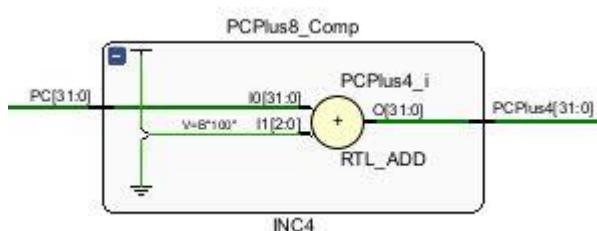
**Εικόνα 5 – Διάγραμμα RTL του πολυπλέκτη λογικής επόμενης εντολής PCN**

#### 1.1.5 ΑΘΡΟΙΣΤΗΣ PCPLUS8

Ο αθροιστής PCPlus8 αυξάνει κατά 8 byte την διεύθυνση της τρέχουσας εντολής. Στην αρχιτεκτονική ARM η τιμή του μετρητή προγράμματος που διαβάζεται από το αρχείο καταχωρητών είναι η PC+08. Οι εντολές διακλάδωσης υπολογίζουν την απόσταση σε λέξεις της επόμενης εντολής που πρέπει να εκτελεστεί, όχι από τη διεύθυνση της τρέχουσας, αλλά από την τιμή PC+08.

Ως είσοδο ο συγκεκριμένος αθροιστής έχει την **PCPlus4**. Η έξοδος τους πηγαίνει στην είσοδο **R15** του αρχείου καταχωρητών.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του αθροιστή PCPlus8.



**Εικόνα 6 – Διάγραμμα RTL του αθροιστή PCPlus8**

#### 1.1.6 ΠΟΛΥΠΛΕΚΤΗΣ A1 REG

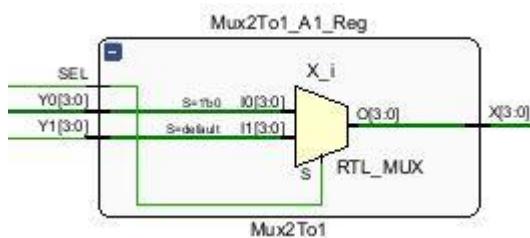
Ο συγκεκριμένος πολυπλέκτης υπάρχει για την επιλογή του καταχωρητή που θα διαβαστεί από την έξοδο RD1 του αρχείου καταχωρητών.

Η επιλογή γίνεται μεταξύ της εισόδου **Rn**, δηλαδή των bit 19-16 των εντολών (εκτός από τις εντολές διακλάδωσης) και της εισόδου με σταθερή τιμή '15' η οποία διαβάζει την έξοδο του **PCPlus8** και χρησιμοποιείται στις εντολές διακλάδωσης. Τέλος υπάρχει και η είσοδος **RegSrc(0)** η οποία έρχεται από τη

μονάδα ελέγχου και μέσω της οποία γίνεται η επιλογή της εξόδου. Η ανάλυση της τιμής που παίρνει η είσοδος RegSrc(0) θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου.

Η έξοδος του πολυπλέκτη πηγαίνει στην είσοδο **A1** του αρχείου καταχωρητών.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του πολυπλέκτη A1 REG.



**Εικόνα 7 – Διάγραμμα RTL του πολυπλέκτη A1 REG**

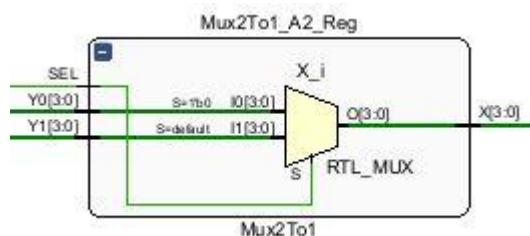
### 1.1.7 ΠΟΛΥΠΛΕΚΤΗΣ A2 REG

Ο συγκεκριμένος πολυπλέκτης υπάρχει για την επιλογή του καταχωρητή που θα διαβαστεί από την έξοδο **RD2** του αρχείου καταχωρητών.

Η επιλογή γίνεται μεταξύ της εισόδου **Rm**, δηλαδή των bit 3-0 των εντολών επεξεργασίας δεδομένων όταν I=0 και της εισόδου Rd δηλαδή των bit 15-12 των εντολών πλην των εντολών διακλάδωσης όταν I=1. Τέλος υπάρχει και η είσοδος RegSrc(1) η οποία έρχεται από τη μονάδα ελέγχου και μέσω της οποία γίνεται η επιλογή της εξόδου. Η ανάλυση της τιμής που παίρνει η είσοδος RegSrc(1) θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου.

Η έξοδος του πολυπλέκτη πηγαίνει στην είσοδο A2 του αρχείου καταχωρητών.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του πολυπλέκτη A2 REG.



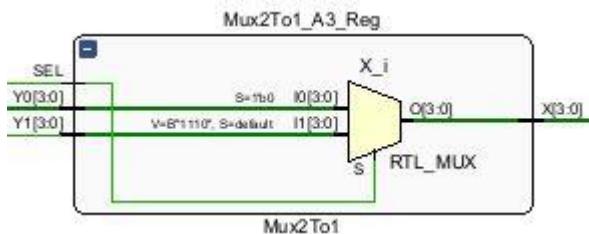
**Εικόνα 8 – Διάγραμμα RTL του πολυπλέκτη A '2 REG**

### 1.1.8 ΠΟΛΥΠΛΕΚΤΗΣ A3 REG

Ο συγκεκριμένος πολυπλέκτης υπάρχει για την επιλογή του καταχωρητή στον οποίο θα γίνει εγγραφή. Η επιλογή γίνεται μεταξύ της εισόδου **Rd**, δηλαδή των bit 19-16 των εντολών (εκτός από τις εντολές διακλάδωσης) και της εισόδου με σταθερή τιμή '**14**'. Ο καταχωρητής R14 λέγεται link register και χρησιμοποιείται σε εντολής διακλάδωσης BL. Τέλος υπάρχει και η είσοδος **RegSrc(2)** η οποία έρχεται από τη μονάδα ελέγχου και μέσω της οποία γίνεται η επιλογή της εξόδου. Η ανάλυση της τιμής που παίρνει η είσοδος RegSrc(2) θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου.

Η έξοδος του πολυπλέκτη πηγαίνει στην είσοδο **A3** του αρχείου καταχωρητών.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του πολυπλέκτη A3 REG.



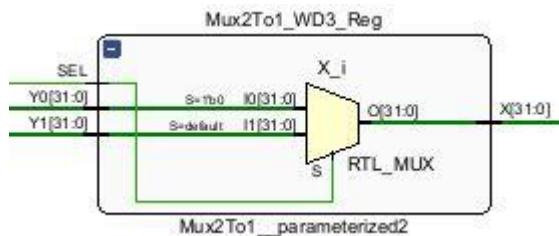
**Εικόνα 9 – Διάγραμμα RTL του πολυπλέκτη A3 REG**

### 1.1.9 ΠΟΛΥΠΛΕΚΤΗΣ WD3 REG

Ο συγκεκριμένος πολυπλέκτης υπάρχει για την επιλογή της τιμής που θα εγγραφεί στον καταχωρητή που επιλέγεται από τον πολυπλέκτη A3 REG.

Η επιλογή γίνεται μεταξύ της εισόδου PCPlus4 όταν έχουμε εντολή διακλάδωσης BL και ως εκ τούτου πρέπει να εγγραφεί η εντολή που θα εκτελεστεί μετά τη διακλάδωση και της εισόδου **result** η οποία είναι είτε το αποτέλεσμα της **ALU** είτε τα δεδομένα που διαβάστηκαν από τη μνήμη δεδομένων. Τέλος υπάρχει και η είσοδος **RegSrc(2)** η οποία έρχεται από τη μονάδα ελέγχου και μέσω της οποία γίνεται η επιλογή της εξόδου. Η ανάλυση της τιμής που παίρνει η είσοδος RegSrc(2) θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου. Η έξοδος του πολυπλέκτη πηγαίνει στην είσοδο **WD3** του αρχείου καταχωρητών.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του πολυπλέκτη WD3 REG.



**Εικόνα 10 – Διάγραμμα RTL του πολυπλέκτη WD3 REG**

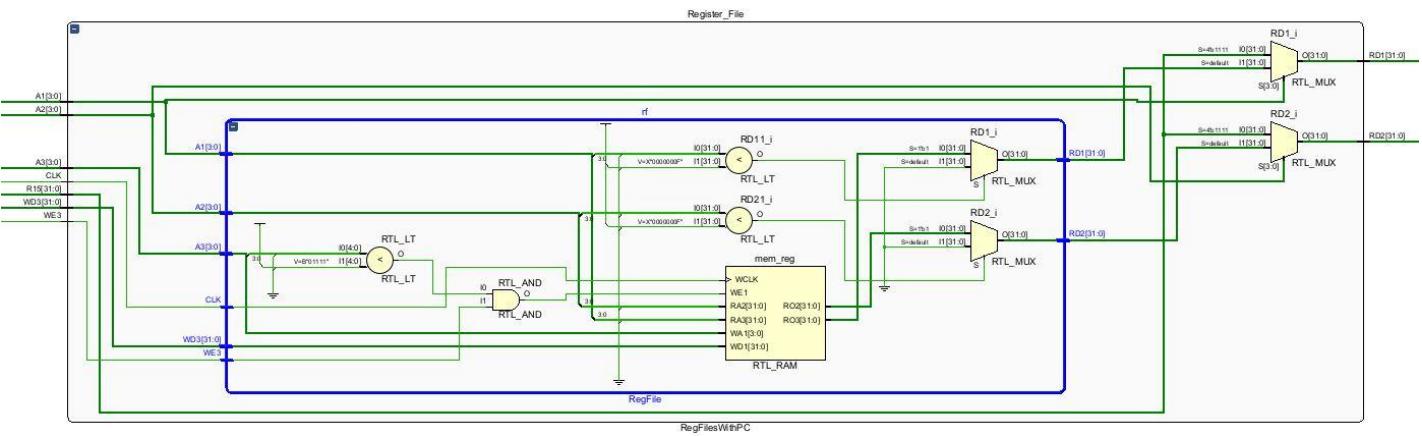
### 1.1.10 ΑΡΧΕΙΟ ΚΑΤΑΧΩΡΗΤΩΝ

Στην αρχιτεκτονική ARM ορίζονται 15 καταχωρητές (R0-R14) συν τον καταχωρητή R15 ο οποίος είναι PC+8. Οι είσοδοι **A1** και **A2** δηλώνουν τους καταχωρητές που θα διαβαστούν από τις εξόδους **RD1** και **RD2** και συνδέονται με τους πολυπλέκτες που περιεγράφηκαν παραπάνω. Η ανάγνωση των καταχωρητών γίνεται ασύγχρονα. Η είσοδος **A3** δηλώνει τον καταχωρητή στον οποίο θα γίνει εγγραφή και η είσοδος **WD3** φέρνει τα δεδομένα τα οποία θα εγγραφούν. Η εγγραφή γίνεται σύγχρονα. Η είσοδος **R15** έχει την τιμή PC+8.

Επίσης υπάρχει η είσοδος **CLK** η οποία είναι συνδεδεμένη στο σήμα ρολογιού του επεξεργαστή και η write enable. Κατά την ανερχόμενη ακμή του ρολογιού το **write enable** είναι '1' τότε η τιμή στην είσοδο του WD3 εγγράφεται στον καταχωρητή της εισόδου A3 και διατηρείται και στην έξοδό του, μέχρι την επόμενη εγγραφή. Το σήμα write enable προέρχεται από την μονάδα ελέγχου και περιγράφεται στο αντίστοιχο κεφάλαιο.

Η υλοποίηση του αρχείο καταχωρητών έγινε με χρήση του component rf. Το συγκεκριμένο component διαχειρίζεται την ανάγνωση και την εγγραφή στους καταχωρητές R0-R14 το οποίο διασφαλίζει ότι δε γίνει εγγραφή στον καταχωρητή R15. Ο R15 υλοποιείται σε ξεχωριστό καταχωρητή.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του αρχείου καταχωρητών.

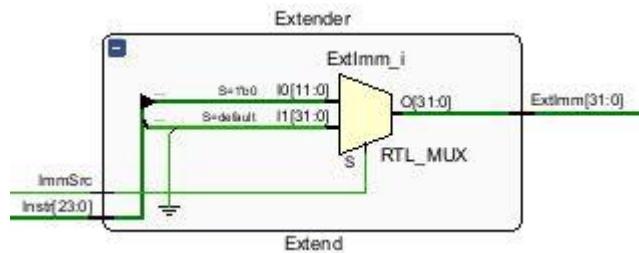


**Εικόνα 11 – Διάγραμμα RTL του αρχείου καταχωρητών**

### 1.1.11 EXTENDER

Η μονάδα επέκτασης κάνει επέκταση μηδενός από τα 12 bit στα 32 bit ή πρόσημου από τα 24 bit στα 32bit. Έχει σαν είσοδο **τα bit 23-0 της εντολής** καθώς και το **ImmSrc**. Όταν το ImmSrc ισούται με '1' δηλαδή έχουμε εντολή με τελεστέο όχι καταχωρητή αλλά αριθμό γίνεται επέκταση μηδενός από τα 12 bit. Όταν έχουμε εντολής διακλάδωσης γίνεται επέκταση προσήμου από τα 24 bit.

Η έξοδος της μονάδας επέκτασης πηγαίνει στον **πολυπλέκτη** ο οποίος επιλέγει την είσοδο SrcB στην ALU. Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας επέκτασης.



**Εικόνα 12 – Διάγραμμα RTL της μονάδας επέκτασης**

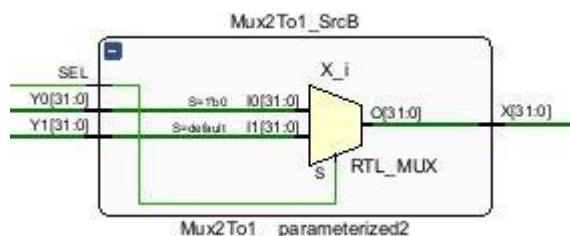
### 1.1.12 ΠΟΛΥΠΛΕΚΤΗΣ SRCB

Ο συγκεκριμένος πολυπλέκτης υπάρχει για την επιλογή της τιμής της εισόδου SrcB της ALU.

Η επιλογή γίνεται μεταξύ της εισόδου **RD2** όταν έχουμε πράξη μεταξύ καταχωρητών και της εισόδου ExtImm όταν έχουμε πράξη μεταξύ καταχωρήτη και σταθερής τιμής/. Τέλος υπάρχει και η είσοδος **AluSrc** η οποία έρχεται από τη μονάδα ελέγχου και μέσω της οποία γίνεται η επιλογή της εξόδου. Η ανάλυση της τιμής που παίρνει η είσοδος AluSrc θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου.

Η έξοδος του πολυπλέκτη πηγαίνει στην είσοδο **SrcB** της ALU.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του πολυπλέκτη SrcB.

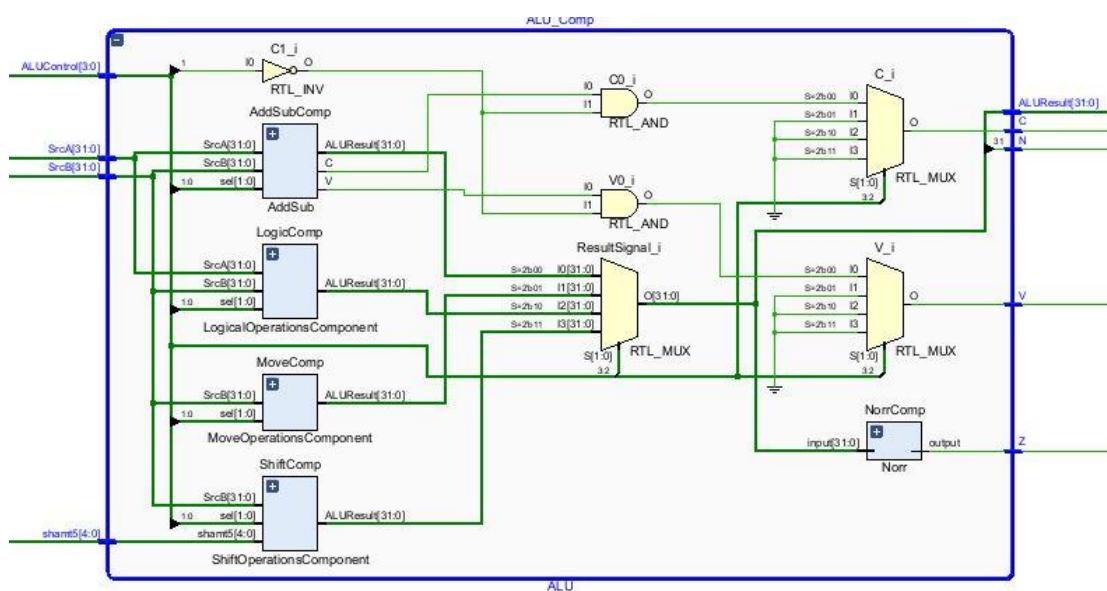


**Εικόνα 13 – Διάγραμμα RTL του πολυπλέκτη SrcB**

### 1.1.13 ΜΟΝΆΔΑ ALU

Στην μονάδα ALU εκτελούνται όλες οι πράξεις που εκτελεί ο επεξεργαστής. Η είσοδος **SrcA** της μονάδας είναι ο πρώτος τελεστέος της πράξης και έρχεται από το αρχείο καταχωρητών. Η είσοδος **SrcB** της μονάδας είναι ο δεύτερος τελεστέος και έρχεται από τον πολυπλέκτη SrcB ο οποίος περιεγράφη στην προηγούμενη παράγραφο. Υπογραμμίζεται πράξεις στην αρχιτεκτονική ARM πράξεις εκτελούνται μόνο μεταξύ καταχωρητών ή καταχωρητών και σταθερών τιμών. Η είσοδος **shamt5** δείχνει την ποσότητα ολίσθησης όταν εκτελούνται πράξεις ολίσθησης (ASR, LSL, LSR, ROR). Τέλος υπάρχει και η είσοδος **ALUControl** η οποία έρχεται από τη μονάδα ελέγχου και μέσω της οποία γίνεται η επιλογή της πράξης που θα υλοποιηθεί. Η ανάλυση της τιμής που παίρνει η είσοδος ALUControl θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου. Η έξοδος της μονάδας ALU είναι το αποτέλεσμα της πράξης, δηλαδή το **ALUResult** και οι **σημαίες C,Z,N,V**. Αναλυτική περιγραφή της μονάδας ALU και των λειτουργιών της θα γίνει στο αντίστοιχο κεφάλαιο.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας ALU.



**Εικόνα 14 – Διάγραμμα RTL της μονάδας ALU**

### 1.1.14 ΚΑΤΑΧΩΡΗΤΕΣ ΚΑΤΑΣΤΑΣΗΣ

Το αρχείο καταχωρητών κρατάει την τιμή των σημαιών C, Z, N, V. Έχει σαν εισόδους τις τέσσερις **σημαίες** η τιμή των οποίων έρχεται από την ALU. Για την υλοποίηση του χρησιμοποιείται παραμετροποιήμενος καταχωρητής με reset και write enable. Άλλες είσοδοι πέρα από το write enable και το reset επίσης είναι το σήμα του ρολογίου.

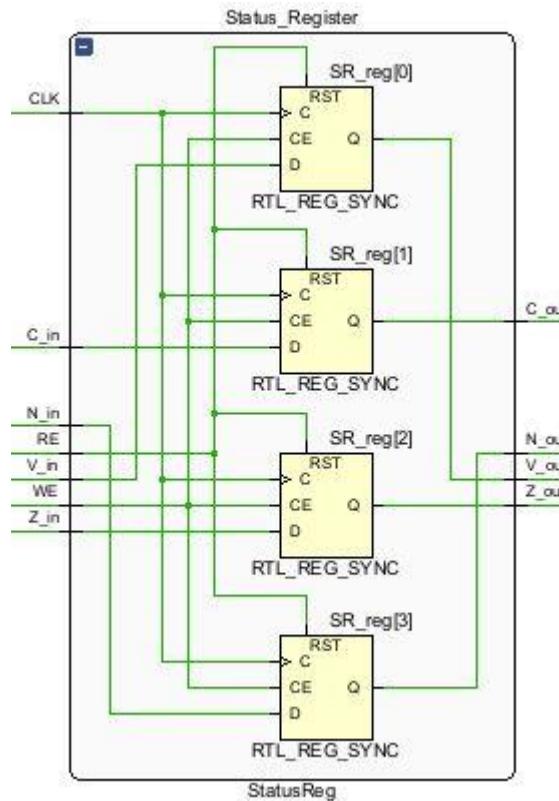
Η είσοδος του **ρολογιού** είναι συνδεδεμένη στο σήμα ρολογιού του επεξεργαστή.

Όταν κατά την ανερχόμενη ακμή του ρολογιού το **reset** είναι '1' τότε θέτει σε όλες τις σημαίες την τιμή '0'. Ομοίως με το ρολόι και το reset είναι συνδεδεμένο με το reset του γενικού κυκλώματος.

Όταν κατά την ανερχόμενη ακμή του ρολογιού το **flags write** είναι '1' τότε οι τιμές στην είσοδο της μονάδας εγγράφονται στους καταχωρητές και διατηρούνται και στην έξοδό του, μέχρι την επόμενη εγγραφή. Το **flags write** ελέγχεται από την μονάδα ελέγχου και η σχετική ανάλυση θα γίνει στο αντίστοιχο κεφάλαιο.

Έξοδος της μονάδας είναι οι σημαίες. Το σήμα τους πηγαίνει προς την μονάδα ελέγχου.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL των καταχωρητών κατάστασης,



**Εικόνα 15 – Διάγραμμα RTL των καταχωρητών κατάστασης**

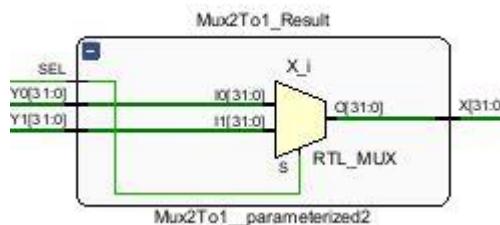
### 1.1.15 ΠΟΛΥΠΛΕΚΤΗΣ RESULT

Ο συγκεκριμένος πολυπλέκτης υπάρχει για την επιλογή μεταξύ του αποτελέσματος της ALU (**ALUResult**) και της εξόδου της μνήμης δεδομένων (RD). Όταν εκτελείται η εντολή LOAD ο πολυπλέκτης επιλέγει την έξοδο της μνήμης δεδομένων. Στις υπόλοιπες εντολές την έξοδο της ALU.

Η επιλογή γίνεται με χρήση του σήματος MemtoReg το οποίο έρχεται από τη μονάδα ελέγχου. Η ανάλυση της τιμής που παίρνει η είσοδος MemtoReg θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου.

Η έξοδος του πολυπλέκτη πηγαίνει στον πολυπλέκτη WD3 REG και στον πολυπλέκτη λογικής επόμενης εντολης.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του πολυπλέκτη Result.



**Εικόνα 16 – Διάγραμμα RTL του πολυπλέκτη Result**

### 1.1.16 ΜΝΗΜΗ ΔΕΔΟΜΕΝΩΝ

Η μνήμη δεδομένων διαθέτει την είσοδο **A** για τον προσδιορισμό της διεύθυνσης ανάγνωσης και εγγραφής. Η συγκεκριμένη είσοδος συνδέεται με την έξοδο της ALU. Η είσοδος WD είναι η είσοδος προσδιορισμού των προς εγγραφή δεδομένων. Επίσης διαθέτει είσοδο ρολογιού και, write enable.

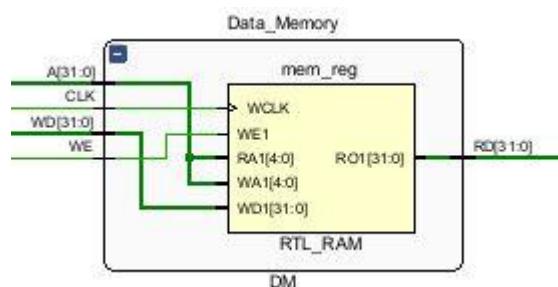
Η είσοδος του **ρολογιού** είναι συνδεδεμένη στο σήμα ρολογιού του επεξεργαστή.

Όταν κατά την ανερχόμενη ακμή του ρολογιού το **write enable** είναι '1' τότε οι τιμές στην είσοδο της μονάδας εγγράφονται στους καταχωρητές και διατηρούνται και στην έξοδό του, μέχρι την επόμενη εγγραφή. Το write enable ελέγχεται από την μονάδα ελέγχου και η σχετική ανάλυση θα γίνει στο αντίστοιχο κεφάλαιο.

Η ανάγνωση από τη μνήμη γίνεται ασύγχρονα.

Η έξοδος RD της μνήμης συνδέεται με τον πολυπλέκτη Result.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μνήμης δεδομένων.



**Εικόνα 17 – Διάγραμμα RTL της μνήμης δεδομένων**

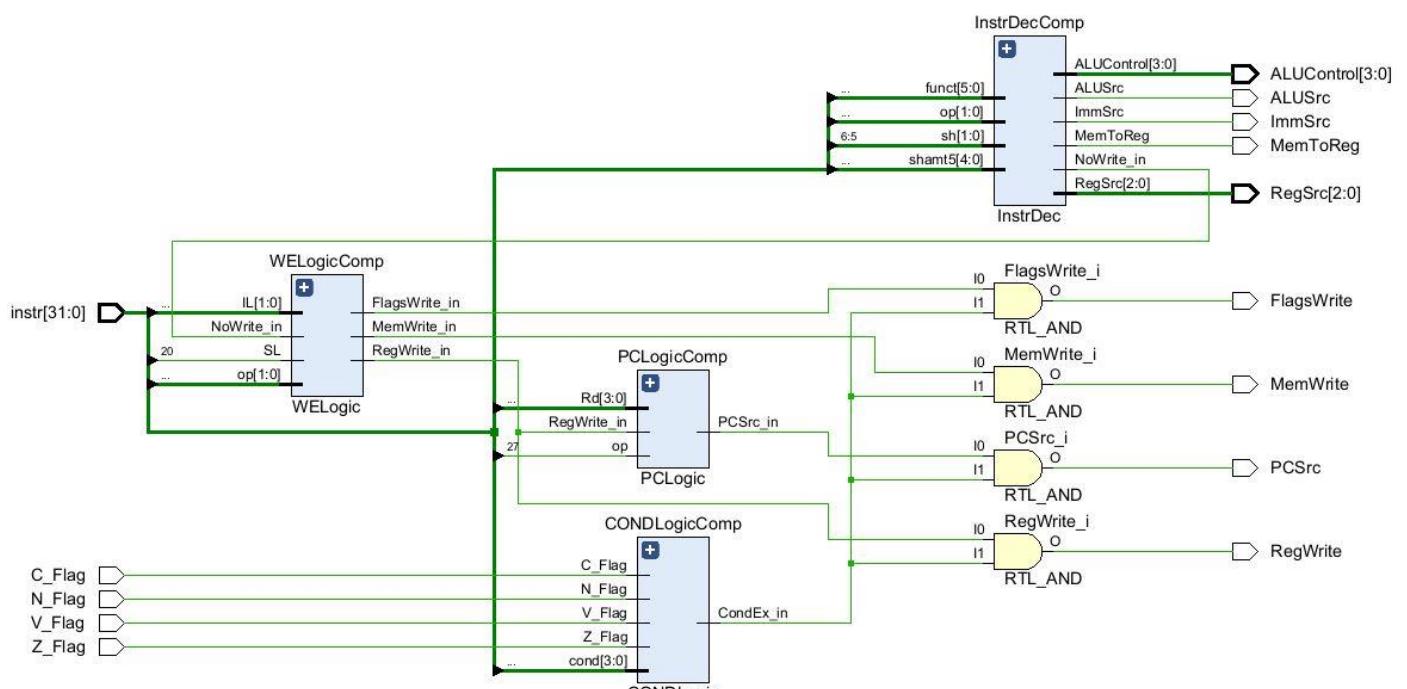
## 1.2 ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

Η μονάδα ελέγχου αποτελείται από τον αποκωδικοποιητή εντολής (InstrDec), τον αποκωδικοποιητή σημάτων έγκρισης εγγραφής (WELogic), την λογική επιλογής διεύθυνσης επόμενης εντολής (PCLogic), τη λογική ελέγχου συνθήκης (CONDLogic) και τις λογικές πύλες and. Τα σήματα για την εγγραφή στην μνήμη (MemWrite), την ενημέρωση των σημαιών(FlagsWrite), την εγγραφή στο αρχείο καταχωρητών (RegWrite) και επιλογής της επόμενης εντολής (PCSsrc) συνδυάζονται με το σήμα ελέγχου συνθήκης (CondEx\_in) μέσω των and πυλών. Στην ουσία αν δεν ικανοποιείται η συνθήη εκτέλεσης δε μπορεί να πραγματοποιηθεί, εγγραφή στη μνήμη ή τους καταχωρητές, ενημέρωση των σημαιών ή διακλάδωση.

Πέρα από τα σήματα εξόδου που αναφέρθηκαν η μονάδα ελέγχου έχει σαν έξοδο τα σήματα RegSrc, ALUSrc, MemtoReg, ALUControl και ImmSrc. Η ανάλυση τους θα γίνει στις επόμενες παραγράφους.

Είσοδοι της μονάδας είναι η εντολή που εκτελέσται (Instr), και οι σημαίες.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μνήμης δεδομένων.



**Εικόνα 18 – Διάγραμμα RTL της μονάδας ελέγχου**

### 1.2.1 ΣΧΕΔΙΑΣΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ ΕΝΤΟΛΩΝ (InstrDec)

Ο αποκωδικοποιητής εντολών έχει σαν εισόδους τα πεδία op, sh, shamt5 και funct της εντολής. Έξοδοι είναι τα σήματα ελέγχου RegSrc[1:0], ALUSrc, ImmSrc, ALUControl[1:0] και MemtoReg, καθώς και το εσωτερικό σήμα NoWrite\_in.

Η ανάλυση των εισόδων και των εξόδων είναι η εξής:

- **Είσοδος op:** Το συγκεκριμένο πεδίο είναι γνωστό ως operation code αποτελείται από τα bit 27:26 της εντολής και κωδικοποιεί τον τύπο της εντολής που θα εκτελεστεί. Στον παρακάτω πίνακα φαίνεται η κωδικοποίηση:

<b>Κωδικός</b>	<b>Τύπος εντολής</b>
00	Επεξεργασία δεδομένων
01	Μνήμης
10	Διακλάδωσης

**Πίνακας 1 – Είσοδος InstrDec - op**

- **Είσοδος funct:** το συγκεκριμένο πεδίο ονομάζεται και κωδικός λειτουργίας και ανάλογα με τον τύπο της εντολής τα bit που το αποτελούν έχουν διαφορετική σημασία. Στον παρακάτω πίνακα φαίνεται η κωδικοποίηση:

<b>Τύπος εντολών</b>	<b>Κωδικός</b>	<b>Θέση bit</b>	<b>Τιμή</b>
Επεξεργασίας δεδομένων	I	25	I = 1 για άμεσο τελεστέο I = 0 για καταχωρητή
	cmd	24:21	Ξεχωριστή τιμή ανάλογα την εντολή
	S	20	S = 1 για ενεργοποίηση σημαιών S = 0 για τη μη ενεργοποίηση σημαιών
Εντολές μνήμης	I	25	I = 0 για καταχωρητή I = 1 για άμεσο τελεστέο
	P	24	PW = 00 για μετα-αριθμοδεικτοδότηση
	W	21	PW = 10 Σχετική απόσταση PW = 11 για προ-αριθμοδεικτοδότηση
	B	22	LB = 00 STR LB = 01 STRB
	L	20	LB = 10 LDR LB = 11 LDRB
	U	23	U = 1 για πρόσθεση U = 0 για αφαίρεση
Εντολές διακλάδωσης	1L	25:24	L = 1 για BL L = 0 για B

**Πίνακας 2 – Είσοδος InstrDec - funct**

- **Είσοδος sh:** το συγκεκριμένο πεδίο χρησιμοποιήται για να καθοριστεί ποια shift εντολή θα υλοποιηθεί. Στον παρακάτω πίνακα φαίνεται η κωδικοποίηση:

<b>Τιμή</b>	<b>Εντολή</b>
sh = 00	LSL
sh = 01	LSR

sh = 10	ASR
sh = 11	ROR

**Πίνακας 3 – Εισόδος InstrDec - sh**

- **Εισόδος shamt5:** Το πεδίο shamt5 χρησιμοποιείται αποκλειστικά για να διακρίνεται αν η κωδικοποίηση funct=1101 αντιστοιχεί σε απλή MOV/NOP ή σε μία από τις εντολές ολίσθησης. Με αυτό τον τρόπο, όταν funct=1101 και το shamt5 δεν είναι μηδέν, η ALU εκτελεί την αντιστοιχη ολίσθηση (LSL, LSR, ASR, ROR)- διαφορετικά η εντολή θεωρείται MOV.

Τιμή	Εντολή
Shamt5 = 00000	MOV
Shamt5 = 00000	LSR

**Πίνακας 4 – Εισόδος InstrDec - shamt5**

- **Έξοδος RegSrc[2:0]:** η συγκεκριμένη έξοδος ελέγχει τους πολυπλέκτες μέσω των οποίων γίνεται η επιλογή των τιμών που θα λάβουν οι είσοδοι A1, A2, A3 στο Register File όπως φαίνεται στον παρακάτω πίνακα:

Τιμή	Εισόδος στο αρχείο καταχωρητών
RegSrc[0] = 0	RA1 = Rn
RegSrc[0] = 1	RA1 = 15
RegSrc[1] = 0	RA2 = Rm
RegSrc[1] = 1	RA2 = Rd
RegSrc[2] = 0	RA3 = Rd
RegSrc[2] = 1	RA2 = 14 (Link Register)

**Πίνακας 5 – Έξοδος InstrDec - RegSrc**

- **Έξοδος AluSrc:** ελέγχει τον πολυπλέκτη μέσω του οποίο γίνεται η επιλογή της SrcB στην ALU όπως φαίνεται στον παρακάτω πίνακα:

Τιμή	Εισόδος SrcB στην ALU
AluSrc = 0	SrcB = RD2
AluSrc = 1	SrcB = ExtImm

**Πίνακας 6 – Έξοδος InstrDec - AluSrc**

- **Έξοδος ImmSrc:** ελέγχει την μονάδα Extend όπως φαίνεται στον παρακάτω πίνακα:

Τιμή	Λειτουργία Extend
ImmSrc = 0	Επέκταση μηδενός από τα 12->32bit
ImmSrc = 1	Επέκταση προσημου από τα 26->32bit

**Πίνακας 7 – Έξοδος InstrDec - ImmSrc**

- Έξοδος ALUControl[3:0]: ελέγχει την εντολή που θα εκτελέσει η ALU.

<b>ALUControl(3:2)</b>	<b>ALUControl(1:0)</b>	<b>Κατηγορία</b>	<b>Εντολή</b>	<b>Περιγραφή</b>
00	00	Add/Sub	ADD	SrcA + SrcB
00	01	Add/Sub	SUB	SrcA - SrcB
01	00	Move	MOV	Rd $\leftarrow$ SrcB
01	01	Move	MVN	Rd $\leftarrow$ NOT SrcB
01	11	Move	NOP	Καμία αλλαγή (MOV R0,R0)
10	00	Logic	XOR	SrcA XOR SrcB
10	10	Logic	AND	SrcA AND SrcB
10	11	Logic	ORR	SrcA OR SrcB
11	00	Shift	ASR	Αριθμητική ολισθηση δεξιά
11	01	Shift	LSL	Λογική ολισθηση αριστερά
11	10	Shift	LSR	Λογική ολισθηση δεξιά
11	11	Shift	ROR	Κυκλική ολισθηση δεξιά

**Πίνακας 8 – Έξοδος InstrDec – ALUControl**

- Έξοδος MemtoReg: ελέγχει ποια τιμή θα εγγραφεί στο αρχείο καταχωρητών.

<b>Τιμή</b>	<b>Result</b>
MemtoReg = 0	Result = ALUResult
MemtoReg = 1	Result = RD

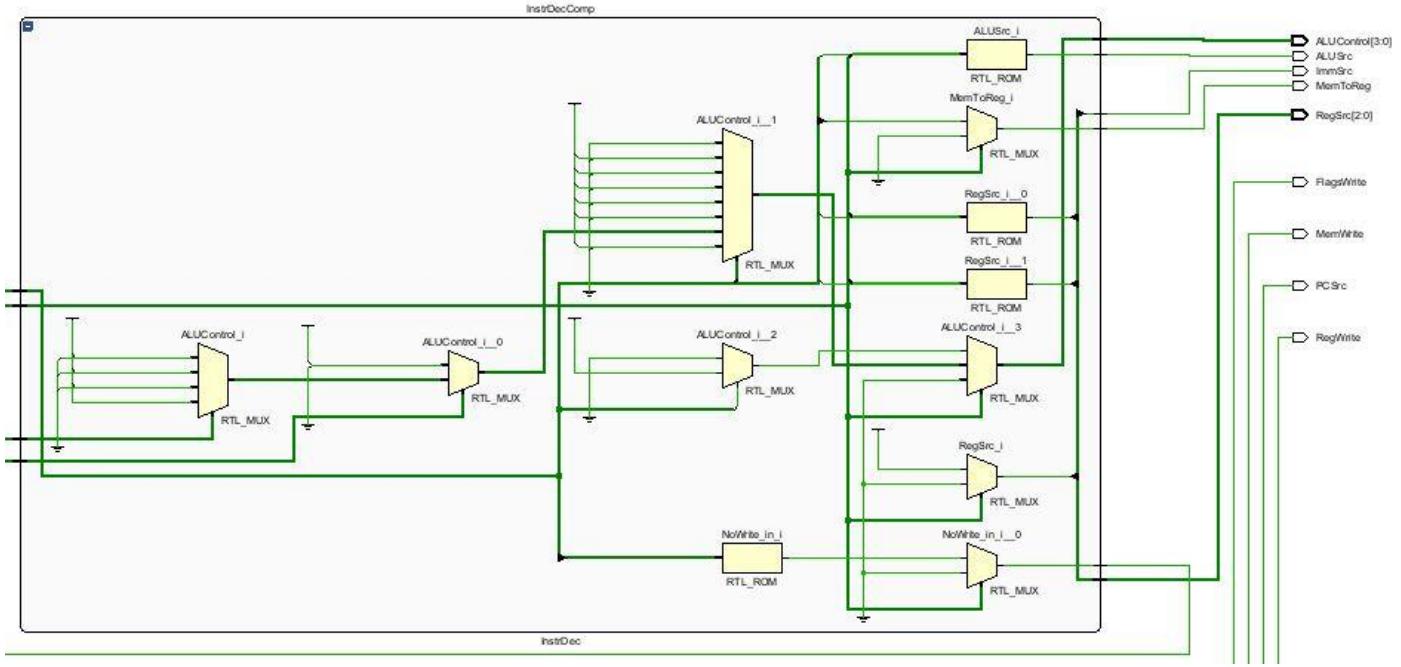
**Πίνακας 9 – Έξοδος InstrDec - MemtoReg**

- Εσωτερικό σήμα NoWrite\_in: χρησιμοποιείται στην εντολή CMP για να μη γινει εγγραφή στο αρχείο καταχωρητών ή τη μνήμη.

<b>Τιμή</b>	<b>Result</b>
NoWrite_in = 0	Κανονική λογική ελέγχου της RegWrite
NoWrite_in = 1	Το αποτέλεσμα της ALU δεν αποθηκεύεται σε κανέναν καταχωρητή

**Πίνακας 10 – Έξοδος InstrDec - NoWrite\_in**

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του αποκωδικοποιητή εντολών.



**Εικόνα 19 – Διάγραμμα RTL του αποκωδικοποιητή εντολών**

#### 1.2.1.1 Εντολές Επεξεργασίας Δεδομένων ( $op = "00"$ )

Πρόκειται για τις εντολές επεξεργασίας δεομένων, οι οποίες είτε χρησιμοποιούν δεύτερο τελεστέο από καταχωρητή είτε άμεση τιμή (immediate).

##### 1.2.1.1.1 Εντολή ADD

Η κωδικοποίηση funct 1 0100 X εκτελεί πρόσθεση.

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου  $RegSrc[1:0] = X0$ .
  - Για  $I=0$  χρειάζονται και οι δύο καταχωρητές άρα  $RegSrc[1:0] = 00$ .
  - $RegSrc[2] = 0$  γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:

  - Για  $I = 1$  είναι  $ImmSrc = 0$  γιατί έχουμε επέκταση μηδενός.
  - Για  $I=0$  η τιμή είναι αδιάφορη.

- ALUControl: 0000 ώστε η ALU να εκτελέσει ADD.
- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

##### 1.2.1.1.2 Εντολή SUB

Η κωδικοποίηση funct 1 0010 X εκτελεί αφαίρεση.

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου  $RegSrc[1:0] = X0$ .
  - Για  $I=0$  χρειάζονται και οι δύο καταχωρητές άρα  $RegSrc[1:0] = 00$ .
  - $RegSrc[2] = 0$  γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:

- Για  $I = 1$  είναι ImmSrc = 0 γιατί έχουμε επέκταση μηδενός.
- Για  $I=0$  η τιμή είναι αδιάφορη.
- ALUControl: 0001 ώστε η ALU να εκτελέσει SUB.
- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

### 1.2.1.1.3 Εντολή CMP

Η κωδικοποίηση funct 1 1010 1 εκτελεί σύγκριση (CMP).

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου RegSrc[1:0] = X0.
  - Για  $I=0$  χρειάζονται και οι δύο καταχωρητές άρα RegSrc[1:0] = 00.
  - Εκτελεί μόνο σύγκριση, δεν γίνεται εγγραφή σε καταχωρητή άρα RegSrc[2] = X.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:
  - Για  $I = 1$  είναι ImmSrc = 0 γιατί έχουμε επέκταση μηδενός.
  - Για  $I=0$  η τιμή είναι αδιάφορη.
- ALUControl: 0001 ώστε η ALU να εκτελέσει SUB (χρησιμοποιείται για τη σύγκριση).
- MemToReg: X γιατί δεν υπάρχει εγγραφή σε καταχωρητή.
- NoWrite\_in: 1 γιατί το αποτέλεσμα της ALU δεν πρέπει να γραφτεί σε κανέναν καταχωρητή, ενημερώνονται μόνο οι σημαίες.

### 1.2.1.1.4 Εντολή AND

Η κωδικοποίηση funct 1 0000 X εκτελεί λογικό AND.

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου RegSrc[1:0] = X0.
  - Για  $I=0$  χρειάζονται και οι δύο καταχωρητές άρα RegSrc[1:0] = 00.
  - RegSrc[2] = 0 γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:
  - Για  $I = 1$  είναι ImmSrc = 0 γιατί έχουμε επέκταση μηδενός.
  - Για  $I=0$  η τιμή είναι αδιάφορη.
- ALUControl: 1010 ώστε η ALU να εκτελέσει AND.
- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

### 1.2.1.1.5 Εντολή XOR

Η κωδικοποίηση funct 1 0001 X εκτελεί λογικό XOR.

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου RegSrc[1:0] = X0.
  - Για  $I=0$  χρειάζονται και οι δύο καταχωρητές άρα RegSrc[1:0] = 00.
  - RegSrc[2] = 0 γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:
  - Για  $I = 1$  είναι ImmSrc = 0 γιατί έχουμε επέκταση μηδενός.
  - Για  $I=0$  η τιμή είναι αδιάφορη.

- ALUControl: 1000 ώστε η ALU να εκτελέσει XOR.
- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

#### **1.2.1.1.6 Εντολή ORR**

Η κωδικοποίηση funct 1 1100 X εκτελεί λογικό ORR.

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου  $\text{RegSrc}[1:0] = X0$ .
  - Για  $I=0$  χρειάζονται και οι δύο καταχωρητές άρα  $\text{RegSrc}[1:0] = 00$ .
  - $\text{RegSrc}[2] = 0$  γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:
  - Για  $I = 1$  είναι ImmSrc = 0 γιατί έχουμε επέκταση μηδενός.
  - Για  $I=0$  η τιμή είναι αδιάφορη.
- ALUControl: 1011 ώστε η ALU να εκτελέσει ORR.
- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

#### **1.2.1.1.7 Εντολή MOV**

Η κωδικοποίηση funct 1 1101 0 εκτελεί μεταφορά (MOV).

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου  $\text{RegSrc}[1:0] = X0$ .
  - Για  $I=0$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου  $\text{RegSrc}[1:0] = X0$ .
  - $\text{RegSrc}[2] = 0$  γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:
  - Για  $I = 1$  είναι ImmSrc = 0 γιατί έχουμε επέκταση μηδενός.
  - Για  $I=0$  η τιμή είναι αδιάφορη.
- ALUControl: 0100 ώστε η ALU να εκτελέσει MOV.
- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

#### **1.2.1.1.8 Εντολή MVN**

Η κωδικοποίηση funct 1 1111 0 εκτελεί μεταφορά με αναστροφή (MVN).

- RegSrc:
  - Για  $I=1$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου  $\text{RegSrc}[1:0] = X0$ .
  - Για  $I=0$  δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn. Ως εκ τούτου  $\text{RegSrc}[1:0] = X0$ .
  - $\text{RegSrc}[2] = 0$  γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο, 0 όταν χρησιμοποιείται καταχωρητής.
- ImmSrc:
  - Για  $I = 1$  είναι ImmSrc = 0 γιατί έχουμε επέκταση μηδενός.
  - Για  $I=0$  η τιμή είναι αδιάφορη.
- ALUControl: 0101 ώστε η ALU να εκτελέσει MVN.

- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

### 1.2.1.1.9 Εντολή NOP

Η εντολή NOP υλοποιείται ως ειδική περίπτωση της MOV, δηλαδή ως MOV R0,R0. Έτσι δεν απαιτείται ξεχωριστή λογική αποκαδικοποίησης και η NOP χρησιμοποιεί ακριβώς τα ίδια σήματα ελέγχου με την MOV, χωρίς να επηρεάζει τους υπόλοιπους καταχωρητές.

### 1.2.1.1.10 Εντολές SHIFT

Η κωδικοποίηση funct 0 1101 0 εκτελεί ολίσθηση (LSL, LSR, ASR, ROR) ανάλογα με τα bits του πεδίου sh (δεξ ο σχετικό πίνακα).

- RegSrc: κρατάμε τον καταχωρητή Rm ἀρα RegSrc[1] = 0 και το Rn δεν χρησιμοποιείται αρά RegSrc[0] = X. Ως εκ τούτου RegSrc[1:0] = 0X. Το RegSrc[2] ισούται με 0 γιατί το αποτέλεσμα εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 0 γιατί χρησιμοποιείται καταχωρητής.
- ImmSrc: η τιμή είναι αδιάφορη γιατί δεν γίνεται επέκταση.
- ALUControl:
  - 1101 για LSL,
  - 1110 για LSR,
  - 1100 για ASR,
  - 1111 για ROR.
- MemToReg: 0 γιατί το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

### 1.2.1.2 Εντολές μνήμης ( $op = "01"$ )

Πρόκειται για τις εντολές store και load, οι οποίες χρησιμοποιούν μονο άμεση τιμή (immediate).

### 1.2.1.2.1 Εντολή LDR

Η κωδικοποίηση funct 0 1100 1 ή 0 1000 1 εκτελεί φόρτωση από τη μνήμη (LDR).

- RegSrc: Για LDR με άμεσο τελεστέο δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ RA1 παίρνει την τιμή Rn (τη διεύθυνση βάσης). Ως εκ τούτου RegSrc[1:0] = X0. RegSrc[2] = 0 γιατί η τιμή που διαβάζεται από την μνήμη εγγράφεται στον καταχωρητή Rd.
- ALUSrc: 1 για άμεσο τελεστέο γιατί το offset της διεύθυνσης είναι άμεσο.
- ImmSrc: 0 γιατί έχουμε επέκταση μηδενός του offset.
- ALUControl: 0000 ώστε η ALU να εκτελέσει ADD για να υπολογίσει τη διεύθυνση πρόσβασης.
- MemToReg: 1 γιατί το αποτέλεσμα που θα γραφτεί στον καταχωρητή προορισμού προέρχεται από τη μνήμη και όχι από την ALU.
- NoWrite\_in: 0 γιατί το αποτέλεσμα πρέπει να γραφτεί στον καταχωρητή προορισμού.

### 1.2.1.2.2 Εντολή STR

Η κωδικοποίηση funct 0 1100 0 ή 0 1000 0 εκτελεί αποθήκευση στη μνήμη (STR).

- RegSrc: Για STR χρειάζεται να δοθεί στον ALU και η διεύθυνση βάσης (Rn) και το περιεχόμενο του καταχωρητή που θα αποθηκευτεί. Για το RA2 πρέπει να επιλεγεί το καταχωρητής που περιέχει το δεδομένο που θα γραφτεί, άρα RegSrc[1:0] = 10 ώστε ο πολυπλέκτης να κατευθύνει στον RA2 το Rd (πεδίο της εντολής που περιέχει τον καταχωρητή των δεδομένων). Δεν γίνεται εγγραφή σε καταχωρητή άρα RegSrc[2] = X.
- ALUSrc: 1 για άμεσο τελεστέο γιατί το offset της διεύθυνσης είναι άμεσο.

- ImmSrc: 0 γιατί έχουμε επέκταση μηδενός του offset.
- ALUControl: 0000 ώστε η ALU να εκτελέσει ADD για να υπολογίσει τη διεύθυνση πρόσβασης.
- MemToReg: X γιατί δεν υπάρχει εγγραφή σε καταχωρητή από τη μνήμη.
- NoWrite\_in: 0 γιατί, παρότι δεν γράφουμε σε καταχωρητή, το σήμα αυτό δεν απενεργοποιεί την εγγραφή (η λογική ελέγχου RegWrite είναι ήδη 0).

#### 1.2.1.3 Εντολές διακλάδωσης ( $op = "10"$ )

Πρόκειται για τις εντολές διακλάδωσης με link (BL) είτε χωρίς (B).

##### 1.2.1.3.1 Εντολή B

Η κωδικοποιηση funct 1 0XXX X εκτελεί διακλάδωση χωρίς αποθήκευση διεύθυνσης επιστροφής (B).

- RegSrc: Για διακλάδωση δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ για RA1 χρειάζεται να πάρουμε την τιμή του PC. Ως εκ τούτου RegSrc[1:0] = X1. Δεν γίνεται εγγραφή σε καταχωρητή άρα RegSrc[2] = X.
- ALUSrc: 1 γιατί ο offset της διακλάδωσης είναι άμεσος.
- ImmSrc: 1 γιατί το offset πρέπει να μετατραπεί σε σχετική διεύθυνση με ολίσθηση αριστερά κατά δύο bit και στη συνέχεια να γίνει πρόσημη επέκταση.
- ALUControl: 0000 ώστε η ALU να εκτελέσει ADD για τον υπολογισμό της νέας διεύθυνσης.
- MemToReg: 0 γιατί δεν γίνεται εγγραφή στο αρχείο καταχωρητών από τη μνήμη.
- NoWrite\_in: 0 γιατί δεν απαιτείται ειδική απενεργοποίηση της εγγραφής — η λογική RegWrite είναι ήδη 0.

##### 1.2.1.3.2 Εντολή BL

Η κωδικοποιηση funct 1 1XXX X εκτελεί διακλάδωση με αποθήκευση διεύθυνσης επιστροφής (BL).

- RegSrc: Για BL δεν χρειάζεται δεύτερος καταχωρητής για τον δεύτερο τελεστέο, γι' αυτό η τιμή RA2 είναι αδιάφορη (X) ενώ για RA1 χρειάζεται να πάρουμε την τιμή του PC. Ως εκ τούτου RegSrc[1:0] = X1. Διακλάδωση με αποθήκευση του PC+4 στο R14 (link register) άρα RegSrc[2] = 1.
- ALUSrc: 1 γιατί ο offset της διακλάδωσης είναι άμεσος.
- ImmSrc: 1 γιατί το offset πρέπει να μετατραπεί σε σχετική διεύθυνση με ολίσθηση αριστερά κατά δύο bit και στη συνέχεια να γίνει πρόσημη επέκταση.
- ALUControl: 0000 ώστε η ALU να εκτελέσει ADD για τον υπολογισμό της νέας διεύθυνσης.
- MemToReg: 0 γιατί η αποθήκευση της διεύθυνσης επιστροφής γίνεται απευθείας στο R14 (link register) και όχι από ανάγνωση της μνήμης.
- NoWrite\_in: 0 γιατί πρέπει να γραφτεί η διεύθυνση επιστροφής στον καταχωρητή R14.

#### 1.2.1.4 Πίνακας αληθείας αποκωδικοποιητή εντολών (InstrDec)

Εντολή	Εντολές επεξεργασίας δεδομένων										
	Instr op	Instr funct	shamt5	sh	Tύπος	RegSrc	ALUSrc	ImmSrc	ALUControl	MemToReg	NoWrite in
ADD	00	1 0100 X	XXXXXX	XX	DP Imm	0X0	1	0	0000	0	0
ADD	00	0 0100 X	XXXXXX	XX	DP Reg	000	0	X	0000	0	0
SUB	00	1 0010 X	XXXXXX	XX	DP Imm	0X0	1	0	0001	0	0
SUB	00	0 0010 X	XXXXXX	XX	DP Reg	000	0	X	0001	0	0
CMP	00	1 1010 1	XXXXXX	XX	DP Imm	XX0	1	0	0001	X	1
CMP	00	0 1010 1	XXXXXX	XX	DP Reg	X00	0	X	0001	X	1
AND	00	1 0000 X	XXXXXX	XX	DP Imm	0X0	1	0	1010	0	0
AND	00	0 0000 X	XXXXXX	XX	DP Reg	000	0	X	1010	0	0
XORR	00	1 0001 X	XXXXXX	XX	DP Imm	0X0	1	0	1000	0	0
XORR	00	0 0001 X	XXXXXX	XX	DP Reg	000	0	X	1000	0	0
ORR	00	1 1100 X	XXXXXX	XX	DP Imm	0X0	1	0	1011	0	0
ORR	00	0 1100 X	XXXXXX	XX	DP Reg	000	0	X	1011	0	0
MOV	00	1 1101 0	XXXXXX	XX	DP Imm	0XX	1	0	0100	0	0
MOV-NOP	00	0 1101 0	00000	XX	DP Reg	0X0	0	X	0100	0	0
MVN	00	1 1111 0	XXXXXX	XX	DP Imm	0XX	1	0	0101	0	0
MVN	00	0 1111 0	XXXXXX	XX	DP Reg	0X0	0	X	0101	0	0
LSL	00	0 1101 0	XXXXXX	00	DP Reg	00X	0	X	1101	0	0
LSR	00	0 1101 0	XXXXXX	01	DP Reg	00X	0	X	1110	0	0
ASR	00	0 1101 0	XXXXXX	10	DP Reg	00X	0	X	1100	0	0
ROR	00	0 1101 0	XXXXXX	11	DP Reg	00X	0	X	1111	0	0

Εντολή	Εντολές μνήμης										
	Instr op	Instr funct	shamt5	sh	Tύπος	RegSrc	ALUSrc	Imm Src	ALUControl	MemToReg	NoWrite in
LDR	01	0 1100 1	XXXXXX	XX	M Imm +	0X0	1	0	0000	1	0
LDR	01	0 1000 1	XXXXXX	XX	M Imm -	0X0	1	0	0001	1	0

STR	01	0 1100 0	xxxxx	xx	M Imm +	X10	1	0	0000	x	0
STR	01	0 1000 0	xxxxx	xx	M Imm -	X10	1	0	0001	x	0

Εντολές διακλάδωσης

Εντολή	Instruction				Τύπος	RegSrc	ALUSrc	Imm Src	ALUControl	MemToReg	NoWrite_in
	Instr op	Instr funct	shamt5	sh							
B	10	1 0XXX X	xxxxx	xx	B Imm +	XX1	1	1	0000	0	0
BL	10	1 1XXX X	xxxxx	xx	B Imm +	1X1	1	1	0000	0	0

Πίνακας 11 – Πίνακας αποκωδικοποιητή εντολών

### **1.2.2 ΣΧΕΔΙΑΣΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ ΣΗΜΑΤΩΝ ΕΓΚΡΙΣΗΣ ΕΙΤΡΑΦΗΣ (WELOGIC)**

Ο αποκωδικοποιητής σημάτων έγκρισης εγγραφής χειρίζεται τα σήματα που επιτρέπουν την εγγραφή στο αρχείο καταχωρητών, στη μνήμη και στους καταχωρητές κατάστασης. Παιρνει σαν είσοδο τα πεδία op, S/L και 1L των εντολών καθώς και το εσωτερικό σήμα NoWrite\_in. Οι έξοδοι που παράγει είναι τα εσωτερικά σήματα RegWrite\_in, FlagsWrite\_in και MemWrite\_in.

- **Είσοδος op:** Το συγκεκριμένο πεδίο είναι γνωστό ως operation code αποτελείται από τα bit 27:26 της εντολής και κωδικοποιεί τον τύπο της εντολής που θα εκτελεστεί. Στον παρακάτω πίνακα φαίνεται η κωδικοποίηση:

<b>Κωδικός</b>	<b>Τύπος εντολής</b>
00	Επεξεργασία δεδομένων
01	Μνήμης
10	Διακλάδωσης

**Πίνακας 12 – Είσοδος WELogic - op**

- **Είσοδος S/L:** Το συγκεκριμένο σήμα αποτελείται από το bit 20 της εντολής και η σημασία του φαίνεται στον παρακάτω πίνακα:

Operation code	Σημασία
00	S = 1 για ενεργοποίηση σημαιών S = 0 για τη μη ενεργοποίηση σημαιών
01	L = 1 εντολή LDR L = 0 εντολή STR
10	Αδιάφορο

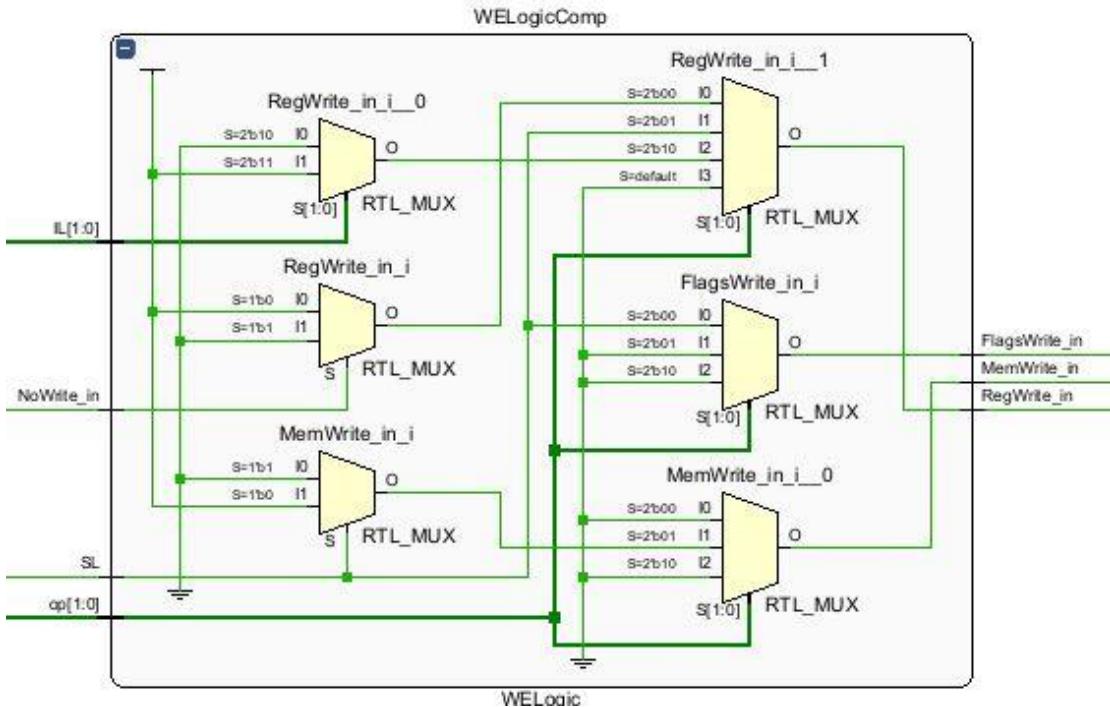
**Πίνακας 13 – Είσοδος WELogic - S/L**

- **Είσοδος 1L:** Το συγκεκριμένο σήμα αποτελείται από τα bit 25:24 της εντολής και αφορά μόνο εντολές διακλάδωσης. Η σημασία του φαίνεται στον παρακάτω πίνακα:

Operation code	Σημασία
10	Εντολή B
11	Εντολή BL

**Πίνακας 14 – Είσοδος WELogic - 1L**

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του αποκωδικοποιητή σημάτων έγκρισης εγγραφής.



**Εικόνα 20 – Διάγραμμα RTL του αποκωδικοποιητή σημάτων έγκρισης εγγραφής**

#### 1.2.2.1 Εντολές Επεξεργασίας Δεδομένων ( $op = "00"$ )

Στις εντολές επεξεργασίας δεδομένων έχουμε πάντα εγγραφή σε καταχωρητή και ως εκ τούτου το μόνο ενεργό σήμα είναι το `RegWrite_in`. Επίσης όταν η είσοδος `S` της εντολής ισούται με '1' ενεργοποιείται το σήμα `FlagsWrite_in`.

Εξαίρεση αποτελεί η εντολή CMP η οποία δεν κάνει εγγραφή κάποιας τιμής στην μνήμη ή σε καταχωρητή αλλά ενεργοποιεί το σήμα εγγραφής για τις σημαίες.

#### 1.2.2.2 Εντολές μνήμης ( $op = "01"$ )

Η εντολή LDR ενεργοποιεί μόνο το σήμα `RegWrite_in` για εγγραφή από τη μνήμη σε register. Η εντολή STR ενεργοποιεί μόνο το σήμα `MemWrite_in` για εγγραφή στη μνήμη.

#### 1.2.2.3 Εντολές διακλάδωσης ( $op = "10"$ )

Η εντολή B δεν κάνει καμία εγγραφή ενώ η BL κάνει εγγραφή στον link register(14) και ως εκ τούτου ενεργοποιεί το σήμα `RegWrite_in`.

#### 1.2.2.4 Πίνακας αληθείας αποκωδικοποιητή σημάτων έγκρισης εγγραφής (WELogic)

Τύπος / εντολή	Instruction			NoWrite_in	RegWrite_in	MemWrite_in	FlagsWrite_in
	Instr op	Instr S/L	Instr 1L				
DP	00	0	XX	0	1	0	0
DP	00	1	XX	0	1	0	1
CMP	00	1	XX	1	0	0	1
LDR	01	1	XX	0	1	0	0
STR	01	0	XX	0	0	1	0
B	10	X	10	0	0	0	0
BL	10	X	11	0	1	0	0

**Πίνακας 15 – Πίνακας αληθείας αποκωδικοποιητή σημάτων έγκρισης εγγραφής**

### 1.2.3 ΣΧΕΔΙΑΣΗ ΛΟΓΙΚΗΣ ΕΠΙΛΟΓΗΣ ΔΙΕΥΘΥΝΣΗΣ ΕΠΟΜΕΝΗΣ ΕΝΤΟΛΗΣ (PCLogic)

Η επιλογή της επόμενης εντολής γίνεται μεταξύ του PC + 4 ή της εξόδου της ALU. Όταν η διεύθυνση προορισμού Rd της εντολής είναι ο καταχωρητής 15 (πχ για εντολή επιστροφής από υπορουτίνα ή εξαίρεση), ή έχουμε εντολή διακλάδωσης η επόμενη εντολή έρχεται από την ALU. Στις υπόλοιπες περιπτώσεις από την έξοδο του PCPlus4.

Οι είσοδοι του PCLogic είναι τα πεδία Rd και op[1] της εντολής καθώς και το εσωτερικό σήμα RegWrite\_in. Έξοδος, το εσωτερικό σήμα PCSrc\_in που ελέγχεται από το εσωτερικό σήμα CondEx\_in, το οποίο παίρνει την τιμή 0, όταν CondEx\_in = 0.

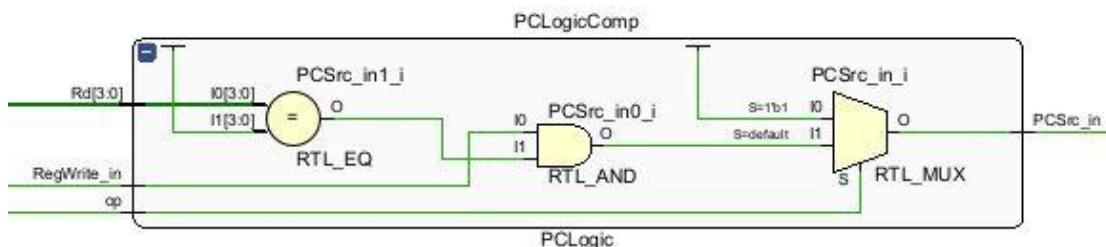
- **Είσοδος Rd:** Η συγκεκριμένη είσοδος είναι ο καταχωρητής προορισμού της εντολής αποτελεί τα bit 12-15 της εντολής για όλου τους τύπους εντολών εκτός των εντολών διακλάδωσης.
- **Είσοδος op:** Το συγκεκριμένο πεδίο είναι γνωστό ως operation code αποτελείται από τα bit 27:26 της εντολής και κωδικοποιεί τον τύπο της εντολής που θα εκτελεστεί. Στον παρακάτω πίνακα φαίνεται η κωδικοποίηση:

<b>Κωδικός</b>	<b>Τύπος εντολής</b>
00	Επεξεργασία δεδομένων
01	Μνήμης
10	Διακλάδωσης

**Πίνακας 16 – Είσοδος PCLogic - op**

- **Είσοδος RegWrite\_in:** το συγκεκριμένο σήμα είναι ενεργό όταν η εντολή πραγματοποιεί εγγραφή στο αρχείο καταχωρητών.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της λογικής επιλογής διεύθυνσης επόμενης εντολής.



**Εικόνα 21 – Διάγραμμα RTL της λογικής επιλογής διεύθυνσης επόμενης εντολής**

#### 1.2.3.1 Εντολές Επεξεργασίας Δεδομένων (op = "00")

Στις εντολές επεξεργασίες δεδομένων το σήμα PCSrc\_in ενεργοποιείται όταν η διεύθυνση προορισμού rd είναι ίση με 15 ( $1111_2$ ).

#### 1.2.3.2 Εντολές μνήμης (op = "01")

Η εντολή LDR ενεργοποιεί το σήμα PCSrc\_in όταν η διεύθυνση προορισμού rd είναι ίση με 15 ( $1111_2$ ). Η εντολή STR δεν ενεργοποιεί ποτέ το συγκεκριμένο σήμα.

#### 1.2.3.3 Εντολές διακλάδωσης (op = "10")

Οι εντολές διακλάδωσης ενεργοποιούν πάντα το σήμα PCSrc\_in.

#### 1.2.3.4 Πίνακας αληθείας λογικής επιλογής διεύθυνσης επόμενης εντολής (PCLogic)

Τύπος	Instruction		RegWrite_in	PCSrc_in
	Instr op	Instr Rd		
DP	00	0000 - 1110	1	0
DP	00	1111	1	1
CMP	00	XXXX	0	0
LDR	01	0000 - 1110	1	0
LDR	01	1111	1	1
STR	01	XXXX	0	0
B	10	XXXX	0	1

Πίνακας 17 – Πίνακας αληθείας λογικής επιλογής διεύθυνσης επόμενης εντολής

#### 1.2.4 ΣΧΕΔΙΑΣΗ ΛΟΓΙΚΗΣ ΕΛΕΓΧΟΥ ΣΥΝΘΗΚΗΣ (CONDLOGIC)

Η λογική ελέγχου συνθήκης ελέγχει εάν ικανοποιείται η συνθήκη που ορίζεται στο πεδίου cond της εντολής με βάση τις τρέχουσες τιμές σημαίες N, Z, C, V. Οι είσοδοι είναι το πεδίο cond της εντολής και η έξοδος flags του καταχωρητή καταστάσεων. Η έξοδος είναι το σήμα CondEx\_in που εγκρίνει την εκτέλεση της εντολής, όταν παίρνει την τιμή 1.

- **Είσοδος cond:** Το πεδίο cond κωδικοποιεί την υπό συνθήκη εκτέλεση της εντολής με βάση τις σημαίες που περιγράφονται στον επόμενο πίνακα. Το πεδίο cond αποτελείται από τα bits 31:28 της εντολής

Κωδικός	Μνημονικό	Όνομα
0000	EQ	Equal
0001	NE	Not equal
0010	CS/HS	Carry set / unsigned higher or same
0011	CC/LO	Carry clear / unsigned lower
0100	MI	Minus / negative
0101	PL	Plus / positive or zero
0110	VS	Overflow / overflow set
0111	VC	No overflow / overflow clear
1000	HL	Unsigned higher
1001	LS	Unsigned lower or same
1010	GE	Signed greater or equal
1011	LT	Signed less
1100	GT	Signed greater
1101	LE	Signed less or equal
1110	AL(ή none)	Always / unconditional
1111	none	For unconditional instructions

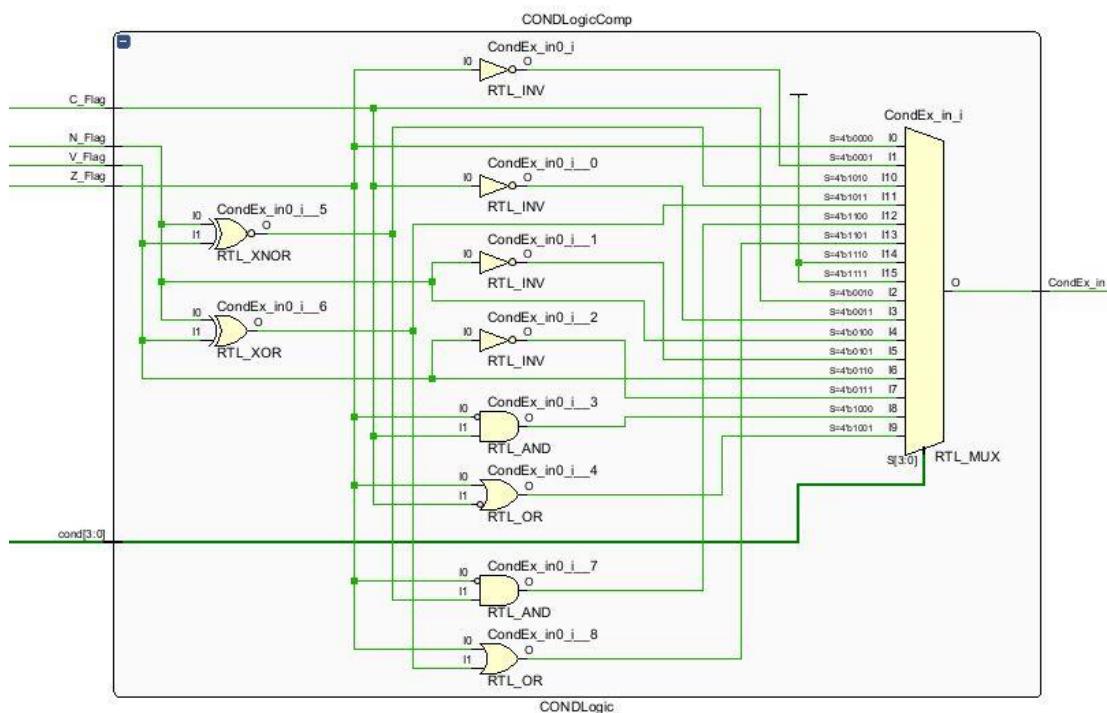
Πίνακας 18 – Είσοδος CONDLogic - cond

- **Είσοδος σημαίες:** Οι σημαίες κωδικοποιούνται ως εξής:

<b>Σημαία</b>	<b>Όνομα</b>	<b>Περιγραφή</b>
N	Αρνητικό	Το αποτέλεσμα της εντολής είναι αρνητικό δηλαδή το περισσότερο σημαντικό bit (bit 31) του αποτελέσματος έχει τιμή 1.
Z	Μηδέν	Το αποτέλεσμα της εντολής είναι ίσο με το μηδέν.
C	Κρατούμενο	Η εντολή παράγει κρατούμενο εξόδου.
V	Υπερχείληση	Η εντολή προκαλεί υπερχείληση.

**Πίνακας 19 – Είσοδος CONDLogic - σημαίες**

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της λογικής επιλογής διεύθυνσης επόμενης εντολής.



**Εικόνα 22 – Διάγραμμα RTL της λογικής ελέγχου συνθήκης**

#### 1.2.4.1 Πίνακας αληθείας λογικής ελέγχου συνθήκης (CONDLogic)

<b>cond</b>	<b>Μνημονικό</b>	<b>Όνομα</b>	<b>CondEx</b>	<b>CondEx in</b>
0000	EQ	Equal	Z	1
0001	NE	Not equal	Z̄	1
0010	CS/HS	Carry set / unsigned higher or same	C	1
0011	CC/LO	Carry clear / unsigned lower	C̄	1
0100	MI	Minus / negative	N	1
0101	PL	Plus / positive or zero	N̄	1
0110	VS	Overflow / overflow set	V	1
0111	VC	No overflow / overflow clear	V̄	1
1000	HI	Unsigned higher	ZC	1
1001	LS	Unsigned lower or same	Z+C	1

1010	GE	Signed greater or equal	ΝΟΥ	1
1011	LT	Signed less	ΝΘΥ	1
1100	GT	Signed greater	Ζ ΝΟΥ	1
1101	LE	Signed less or equal	Ζ+(ΝΘΥ)	1
1110	AL(ή none)	Always / unconditional	1	1
1111	none	For unconditional instructions	1	1

Πίνακας 20 – Πίνακας αληθείας λογικής ελέγχου συνθήκης (CONDLogic)

### 1.3 ΕΠΕΞΕΡΓΑΣΤΗΣ

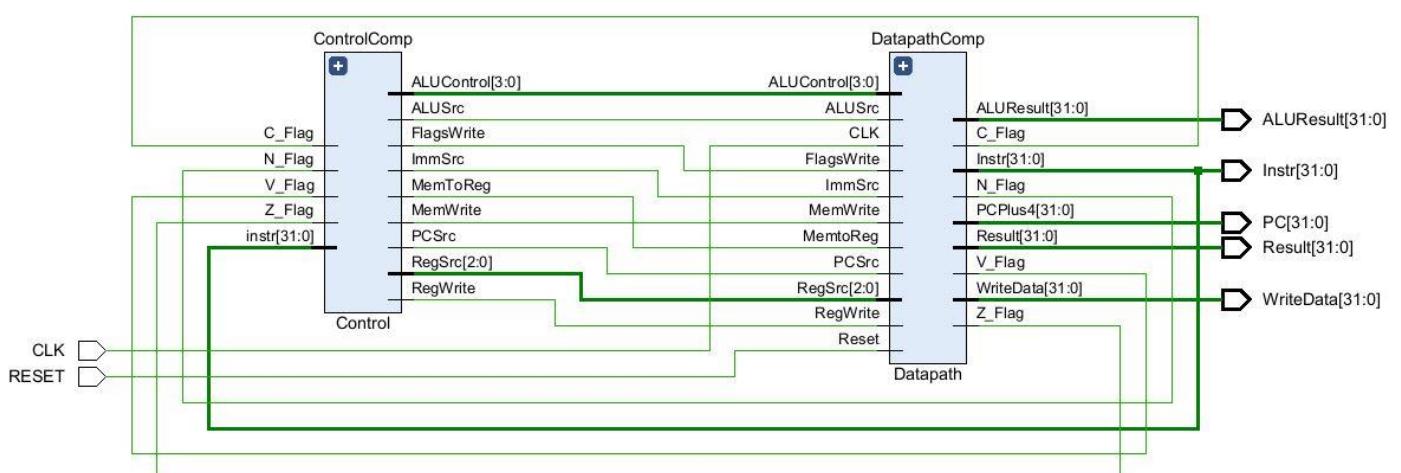
Ο επεξεργαστής, ως ολοκληρωμένη λειτουργική μονάδα, στο ανώτερο ιεραρχικό του επίπεδο, αποτελείται από δύο κύρια τμήματα: τη διαδρομή δεδομένων (Datapath) και τη μονάδα ελέγχου (Control Unit). Η διαδρομή δεδομένων περιλαμβάνει το αρχείο καταχωρητών, την μνήμη δεδομένων, την μνήμη εντολών την αριθμητική και λογική μονάδα (ALU), καθώς και όλα τα απαραίτητα κυκλώματα διασύνδεσης, όπως πολυπλέκτες επιλογής σημάτων και βοηθητικούς (μη αρχιτεκτονικούς) καταχωρητές, οι οποίοι χρησιμοποιούνται για την εκτέλεση των εντολών.

Οι εντολές βρίσκονται αποθηκευμένες στη μνήμη εντολών και, ανάλογα με το περιεχόμενό τους, δεδομένα μεταφέρονται ή αποθηκεύονται στο Αρχείο Καταχωρητών και επεξεργάζονται από την ALU. Η διαδικασία αυτή καθοδηγείται από τα σήματα ελέγχου που παράγει η μονάδα ελέγχου, η οποία λαμβάνει ως είσοδο τόσο την εντολή που πρόκειται να εκτελεστεί όσο και την τιμή του καταχωρητή κατάστασης προγράμματος.

Η βασική είσοδος του επεξεργαστή είναι το σήμα ρολογιού (clock), το οποίο καθορίζει τη συχνότητα λειτουργίας του συστήματος και επηρεάζει άμεσα την ταχύτητα εκτέλεσης των εντολών. Το σήμα αυτό επιτρέπει την ενημέρωση των καταχωρητών και τη μετάβαση του συστήματος από μία κατάσταση στην επόμενη. Επιπλέον, υπάρχει και το σήμα επαναφοράς (reset), το οποίο, αν και σε πραγματικές υλοποιήσεις FPGA δεν είναι πάντοτε απαραίτητο (καθώς οι καταχωρητές και τα flip-flops επανέρχονται αυτόματα σε αρχική κατάσταση κατά την ενεργοποίηση του συστήματος), προστίθεται εδώ για να δίνεται η δυνατότητα χειροκίνητης επαναφοράς του επεξεργαστή στην αρχική του κατάσταση.

Τέλος, οι έξοδοι που έχουν σχεδιαστεί σε αυτή την άσκηση εξυπηρετούν κυρίως τον έλεγχο της σωστής λειτουργίας του επεξεργαστή και δεν είναι απαραίτητες για την κανονική λειτουργία του, καθώς το σύστημα μπορεί να λειτουργήσει πλήρως ως κλειστό σύστημα χωρίς αυτές.

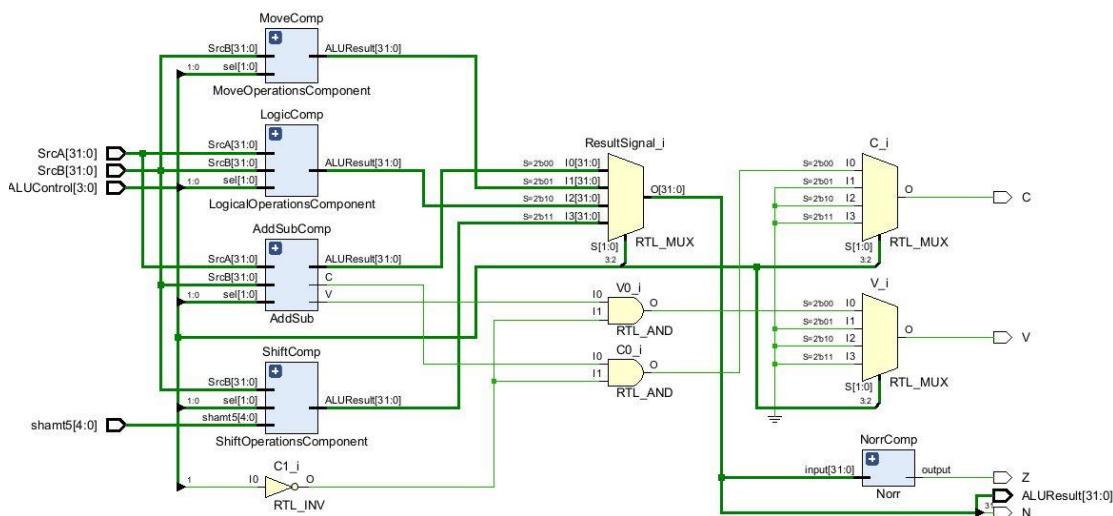
Όλα τα σήματα εξόδου που παράγει η μονάδα ελέγχου χρησιμοποιούνται ως είσοδοι στη διαδρομή δεδομένων. Αντίστροφα, οι είσοδοι της μονάδας ελέγχου προέρχονται από το σήμα Instr, το οποίο αντιστοιχεί στην εντολή που πρόκειται να εκτελεστεί, καθώς και από τις σημαίες. Τα δύο αυτά σήματα αποτελούν εξόδους του Datapath, διαμορφώνοντας έτσι μια αμφίδρομη σχέση επικοινωνίας μεταξύ των δύο υποσυστημάτων. Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL του επεξεργαστή.



Εικόνα 23 – Διάγραμμα RTL επεξεργαστή

## 1.4 ALU

Στην μονάδα ALU εκτελούνται όλες οι πράξεις που εκτελεί ο επεξεργαστής. Η είσοδος **SrcA** της μονάδας είναι ο πρώτος τελεστέος της πράξης και έρχεται από το αρχείο καταχωρητών. Η είσοδος **SrcB** της μονάδας είναι ο δεύτερος τελεστέος και έρχεται από τον πολυπλέκτη SrcB ο οποίος περιεγράφη σε προηγούμενη παράγραφο. Υπογραμμίζεται πράξεις στην αρχιτεκτονική ARM πράξεις εκτελούνται μόνο μεταξύ καταχωρητών ή καταχωρητών και σταθερών τιμών. Η είσοδος **shamt5** δείχνει την ποσότητα ολίσθησης όταν εκτελούνται πράξεις ολίσθησης (ASR, LSL, LSR, ROR). Τέλος υπάρχει και η είσοδος **ALUControl** η οποία έρχεται από τη μονάδα ελέγχου και μέσω της οποία γίνεται η επιλογή της πράξης που θα υλοποιηθεί. Η ανάλυση της τιμής που παίρνει η είσοδος ALUControl θα γίνει στο κεφάλαιο που αφορά τη μονάδα ελέγχου. Η έξοδος της μονάδας ALU είναι το αποτέλεσμα της πράξης, δηλαδή το **ALUResult** και οι **σημαίες** C,Z,N,V. Αναλυτική περιγραφή της μονάδας ALU και των λειτουργιών της θα γίνει στο αντίστοιχο κεφάλαιο. Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας ALU.



**Εικόνα 24 – Διάγραμμα RTL της μονάδας ALU**

Όπως φαίνεται και από το RTL διάγραμμα η υλοποίηση της ALU έγινε με αρχιτεκτονική Structural. Για την υλοποίηση της ALU χρησιμοποιήθηκε η μονάδα AddSub η οποία υλοποιεί πράξεις πρόσθεσης και αφαίρεσης, η μονάδα LogicalOperationsComponent η οποία υλοποιεί λογικές πράξεις, η μονάδα MoveOperationsComponent η οποία υλοποιεί τις πράξεις MOV, MVN και NOP, η μονάδα ShiftOperationsComponent που υλοποιεί πράξεις ολίσθησης και η μονάδα Norr η οποία υλοποιεί την πράξη nor.

Η επιλογή της μονάδας που θα χρησιμοποιείται γίνεται βάσει του σήματος ALUControl και πιο συγκεκριμένα των δυο πιο σημαντικών bits του. Έτσι για:

- ALUControl (3,2) ίσο με '00' χρησιμοποιείται η μονάδα AddSub.
- ALUControl (3,2) ίσο με '01' χρησιμοποιείται η μονάδα MoveOperations.
- ALUControl (3,2) ίσο με '10' χρησιμοποιείται η μονάδα LogicalOperations.
- ALUControl (3,2) ίσο με '11' χρησιμοποιείται η μονάδα ShiftOperations.

Η μονάδα Norr χρησιμοποιείται για τον υπολογισμό της σημαίας Z.

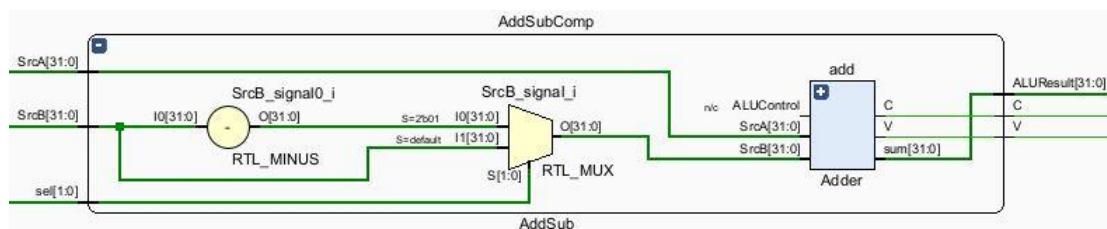
### 1.4.1 ΥΠΟΜΟΝΑΔΑ ADDSUB

Η μονάδα AddSub χρησιμοποιείται για την εκτέλεση πράξεων αφαίρεσης και πρόσθεσης. Σαν είσοδο δέχεται τους αριθμούς μεταξύ των οποίων θα πραγματοποιηθεί η πράξη καθώς και την επιλογή το σήμα sel το οποίο είναι τα δύο λιγότερα σημαντικά ψηφία του σήματος ALUControl. Έτσι όταν:

- sel είναι ίσο με '00' γίνεται πρόσθεση.

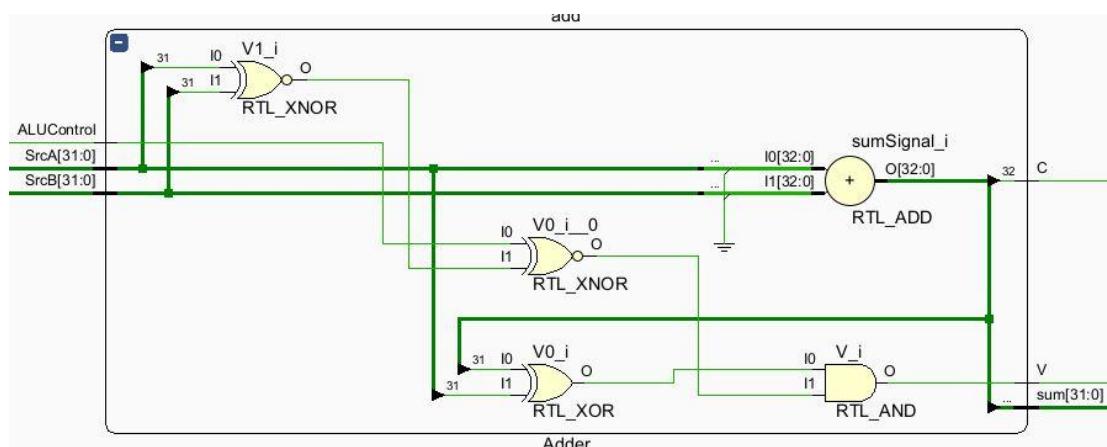
- sel είναι ίσο με '01' γίνεται αφαίρεση.

Οι έξοδοι της μονάδας AddSub είναι το ALUResult το οποίο είναι το αποτέλεσμα της πράξης και οι σημαίες C και V. Υπογραμμίζεται ότι οι σημαίες carry και overflow αλλάζουν κατάσταση μόνο από τις πράξεις αυτές. Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας AddSub.



**Εικόνα 25 – Διάγραμμα RTL της μονάδας AddSub**

Όπως φαίνεται και από το διάγραμμα η μονάδα χρησιμοποιεί το component add το οποίο πραγματοποιεί την πράξη της πρόσθεσης. Στην ουσία όταν έχουμε αφαίρεση κάνουμε πρόσθεση με το συμπλήρωμα ως προς δύο του αριθμού που θέλουμε να αφαιρέσουμε. Η μονάδα add έχει ως εισόδους τους αριθμούς μεταξύ των οποίων θα πραγματοποιηθεί η πρόσθεση καθώς και την επιλογή το σήμα ALUControl το οποίο είναι το λιγότερο σημαντικό ψηφίο του σήματος ALUControl που έρχεται στην ALU και το οποίο χρησιμεύει στον υπολογισμό της σημαίας V. Έξοδος είναι το άθροισμα sum, και οι σημαίες C,V. Υπογραμμίζεται ότι όταν έχουμε αφαίρεση ο αριθμός SrcB γίνεται συμπλήρωμα ως προς 2 στη μονάδα AddSub πριν περαστεί στην add. Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας add.



**Εικόνα 26 – Διάγραμμα RTL της μονάδας add**

#### 1.4.2 ΥΠΟΜΟΝΑΔΑ MoveOperations

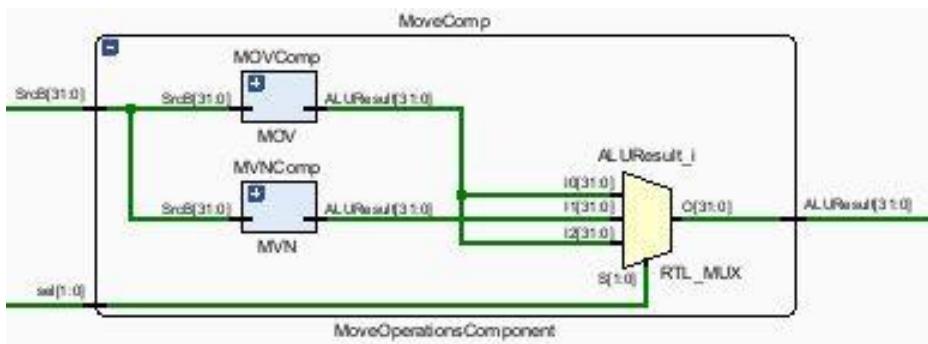
Η μονάδα MoveOperations χρησιμοποιείται για την εκτέλεση πράξεων μεταφοράς (MOV-related operations) μέσα στην ALU.

Σαν έξοδο δέχεται τον αριθμό SrcB, πάνω στον οποίο θα πραγματοποιηθεί η επιλεγμένη πράξη, καθώς και το σήμα επιλογής sel, το οποίο αποτελείται από τα δύο λιγότερα σημαντικά ψηφία του σήματος ALUControl. Ανάλογα με την τιμή του σήματος sel, η μονάδα εκτελεί μία από τις παρακάτω λειτουργίες:

- Όταν  $sel = "00"$ , εκτελείται η εντολή MOV, δηλαδή αντιγράφεται η τιμή του SrcB στην έξοδο ALUResult.
- Όταν  $sel = "01"$ , εκτελείται η εντολή MVN, δηλαδή η έξοδος παίρνει τη συμπληρωματική (bitwise NOT) μορφή του SrcB.
- Όταν  $sel = "11"$ , εκτελείται η εντολή NOP, η οποία πρακτικά ισοδυναμεί με την MOV R0, R0, δηλαδή δεν μεταβάλλει την τιμή του SrcB (περνάει αμετάβλητη στην έξοδο).

Η έξοδος της μονάδας είναι το σήμα ALUResult, το οποίο αντιστοιχεί στο αποτέλεσμα της επιλεγμένης λειτουργίας.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας MoveOperationsComponent.



**Εικόνα 27 – Διάγραμμα RTL της μονάδας MoveOperationsComponent**

Στο σχηματικό διάγραμμα RTL της μονάδας φαίνεται ότι η MoveOperationsComponent χρησιμοποιεί εσωτερικά δύο υπομονάδες:

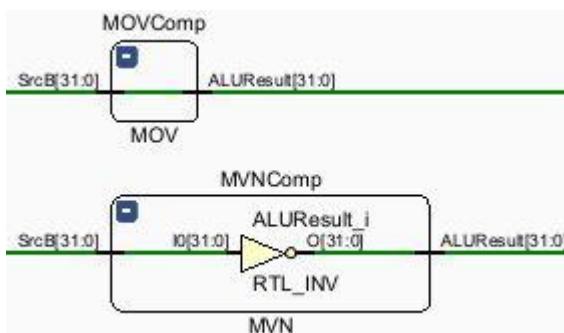
- τη μονάδα MOV, η οποία υλοποιεί την απλή μεταφορά της τιμής του SrcB στην έξοδο, και
- τη μονάδα MVN, η οποία υλοποιεί την πράξη της λογικής αντιστροφής (NOT) πάνω στα bits του SrcB.

Η επιλογή της κατάλληλης εξόδου γίνεται μέσω πολυπλέκτη (with-select statement), ο οποίος κατευθύνει στην έξοδο ALUResult είτε το αποτέλεσμα της MOV, είτε της MVN, είτε πάλι της MOV στην περίπτωση της NOP.

Συνεπώς, η μονάδα MoveOperationsComponent αναλαμβάνει όλες τις λειτουργίες που σχετίζονται με τις εντολές MOV, MVN και NOP, παρέχοντας στη μονάδα ALU ένα ενοποιημένο υποσύστημα μεταφοράς δεδομένων.

Η μονάδα δεν επηρεάζει τις σημαίες κατάστασης (C, V, Z, N), καθώς οι πράξεις αυτές δεν προκαλούν αριθμητική μεταβολή.

Στην παρακάτω εικόνα φαίνονται τα σχηματικά διαγράμματα σε επίπεδο RTL των μονάδων MOV και MVN.



**Εικόνα 28 – Διάγραμμα RTL των μονάδων MOV και MVN**

#### 1.4.3 ΥΠΟΜΟΝΑΔΑ LOGICAL OPERATIONS

Η μονάδα LogicalOperationsComponent χρησιμοποιείται για την εκτέλεση λογικών πράξεων μεταξύ δύο αριθμητικών λέξεων SrcA και SrcB. Η μονάδα αυτή αποτελεί τμήμα της ALU και αναλαμβάνει να πραγματοποιήσει τις βασικές λογικές εντολές AND, OR και XOR, ανάλογα με την τιμή του σήματος ελέγχου sel, το οποίο αντιστοιχεί στα δύο λιγότερα σημαντικά bits του σήματος ALUControl.

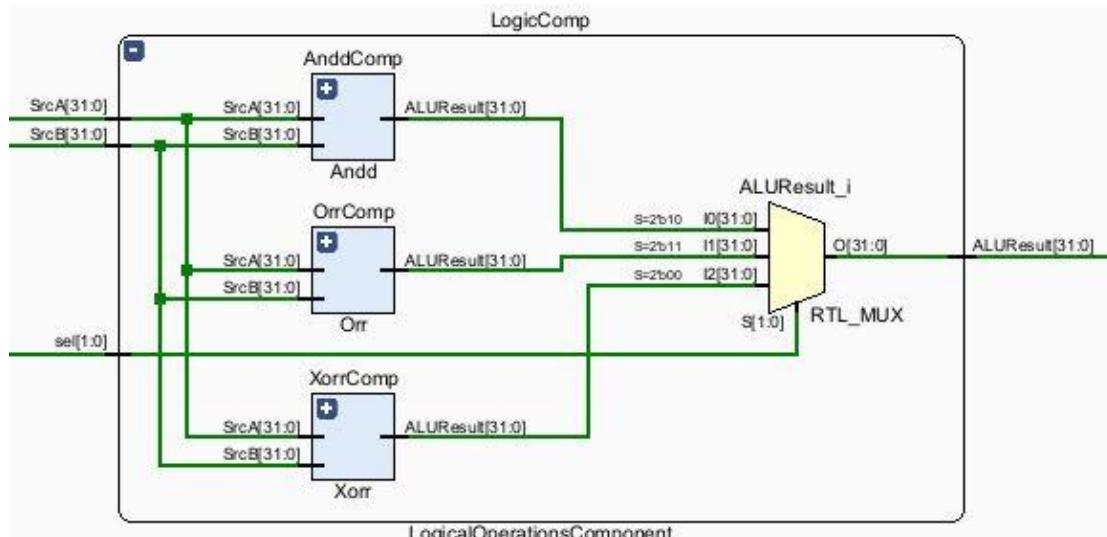
Αναλυτικά, η λειτουργία της μονάδας καθορίζεται ως εξής:

- Όταν  $sel = "10"$ , εκτελείται η εντολή AND, δηλαδή πραγματοποιείται λογικό ΚΑΙ ανάμεσα στα αντίστοιχα bits των SrcA και SrcB.

- Όταν  $sel = "11"$ , εκτελείται η εντολή OR, δηλαδή πραγματοποιείται λογικό 'Η ανάμεσα στα αντίστοιχα bits των δύο εισόδων.
- Όταν  $sel = "00"$ , εκτελείται η εντολή XOR, δηλαδή πραγματοποιείται λογικό ΑΠΟΚΛΕΙΣΤΙΚΟ 'Η (exclusive OR) μεταξύ των SrcA και SrcB.

Η έξοδος της μονάδας είναι το σήμα ALUResult, το οποίο περιέχει το αποτέλεσμα της επιλεγμένης λογικής πράξης.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας LogicalOperations.



**Εικόνα 29 – Διάγραμμα RTL των μονάδας LogicalOperations**

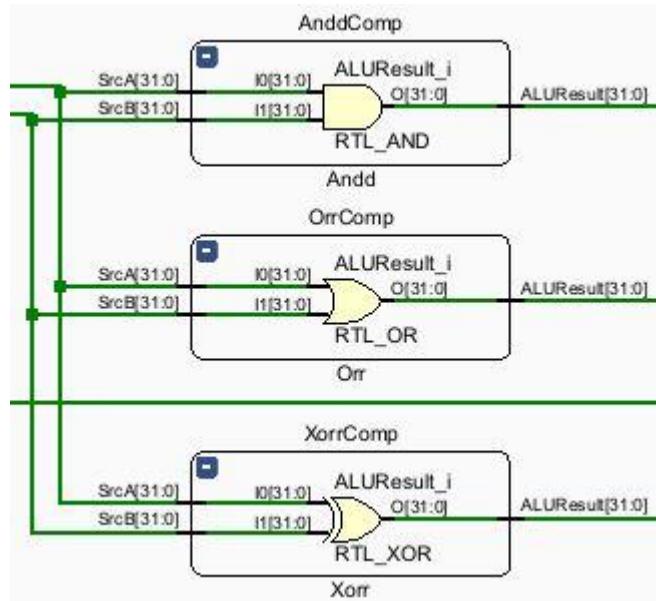
Στο διάγραμμα RTL της μονάδας LogicalOperationsComponent, φαίνεται ότι η μονάδα χρησιμοποιεί εσωτερικά τρεις επιμέρους υπομονάδες:

- Τη μονάδα Anddd, η οποία υλοποιεί την πράξη AND.
- Τη μονάδα Orr, η οποία υλοποιεί την πράξη OR.
- Τη μονάδα Xorrr, η οποία υλοποιεί την πράξη XOR.

Κάθε μία από αυτές τις υπομονάδες δέχεται ως εισόδους τα SrcA και SrcB και παράγει το αντίστοιχο αποτέλεσμα λογικής πράξης.

Η επιλογή της εξόδου που θα σταλεί στην ALUResult γίνεται μέσω πολυπλέκτη (with-select statement), ο οποίος, βάσει του σήματος sel, προωθεί την έξοδο της κατάλληλης υπομονάδας στο τελικό αποτέλεσμα.

Στην παρακάτω εικόνα φαίνονται τα σχηματικά διαγράμματα σε επίπεδο RTL των μονάδων Anddd, Orr και Xorrr.



**Εικόνα 30 – Διάγραμμα RTL των μονάδων *Andd*, *Orr* και *Xorr***

Η μονάδα LogicalOperations είναι υπεύθυνη αποκλειστικά για λογικές πράξεις και δεν επηρεάζει τις σημαίες αριθμητικής κατάστασης (όπως Carry ή Overflow), αφού οι λειτουργίες της δεν περιλαμβάνουν αριθμητική πράξη.

#### 1.4.4 ΥΠΟΜΟΝΑΔΑ SHIFT OPERATIONS

Η μονάδα ShiftOperations είναι υπεύθυνη για την εκτέλεση πράξεων ολίσθησης (shift operations) πάνω στην είσοδο SrcB.

Χρησιμοποιείται από την ALU για την υλοποίηση των εντολών που αφορούν λογική, αριθμητική και κυκλική ολίσθηση των bits ενός δεδομένου.

Η μονάδα δέχεται ως εισόδους:

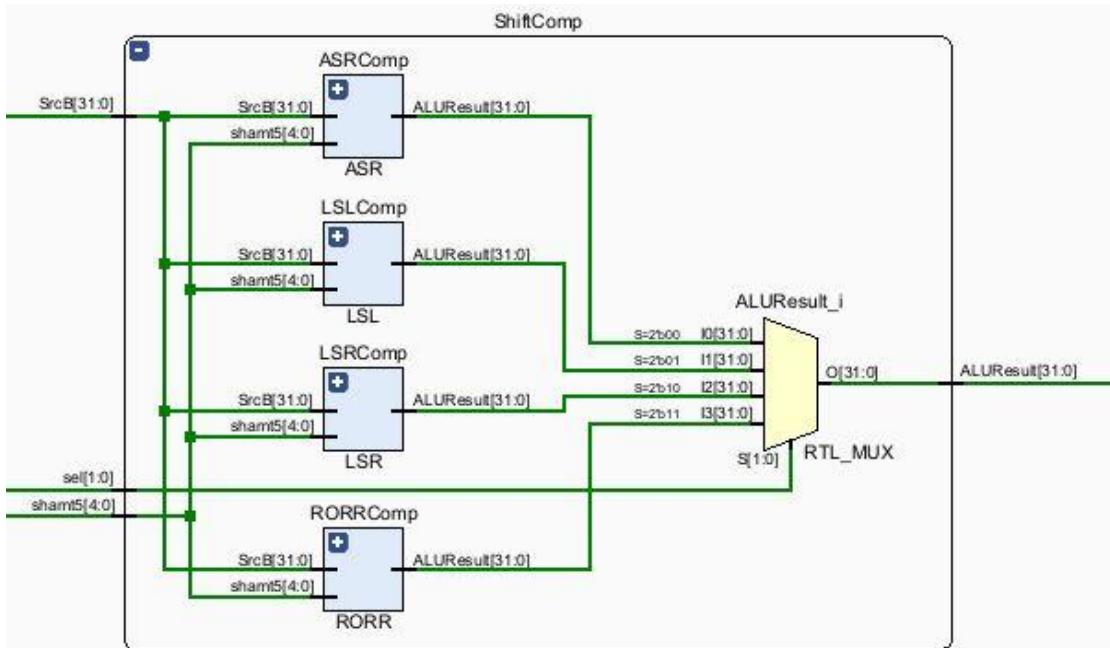
- το σήμα SrcB, το οποίο είναι ο αριθμός προς ολίσθηση,
- το σήμα shamt5, που καθορίζει το πλήθος των θέσεων ολίσθησης (5-bit), και
- το σήμα επιλογής sel, το οποίο αποτελείται από τα δύο λιγότερα σημαντικά bits του σήματος ALUControl και καθορίζει ποια μορφή ολίσθησης θα εκτελεστεί.

Οι λειτουργίες της μονάδας περιγράφονται ως εξής:

- 'Όταν  $sel = "00"$ , εκτελείται Αριθμητική Ολίσθηση Δεξιά (ASR), κατά την οποία το πρόσημο του αριθμού (MSB) διατηρείται σταθερό καθώς τα bits ολισθαίνουν προς τα δεξιά.
- 'Όταν  $sel = "01"$ , εκτελείται Λογική Ολίσθηση Αριστερά (LSL), κατά την οποία τα bits μετακινούνται προς τα αριστερά και τα κενά συμπληρώνονται με μηδενικά.
- 'Όταν  $sel = "10"$ , εκτελείται Λογική Ολίσθηση Δεξιά (LSR), κατά την οποία τα bits μετακινούνται προς τα δεξιά και τα κενά συμπληρώνονται με μηδενικά.
- 'Όταν  $sel = "11"$ , εκτελείται Κυκλική Ολίσθηση Δεξιά (ROR), όπου τα bits που εξέρχονται από το λιγότερο σημαντικό άκρο επανεισάγονται στο περισσότερο σημαντικό.

Η έξοδος της μονάδας είναι το σήμα ALUResult, το οποίο περιέχει το αποτέλεσμα της επιλεγμένης πράξης ολίσθησης.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας LogicalOperations.

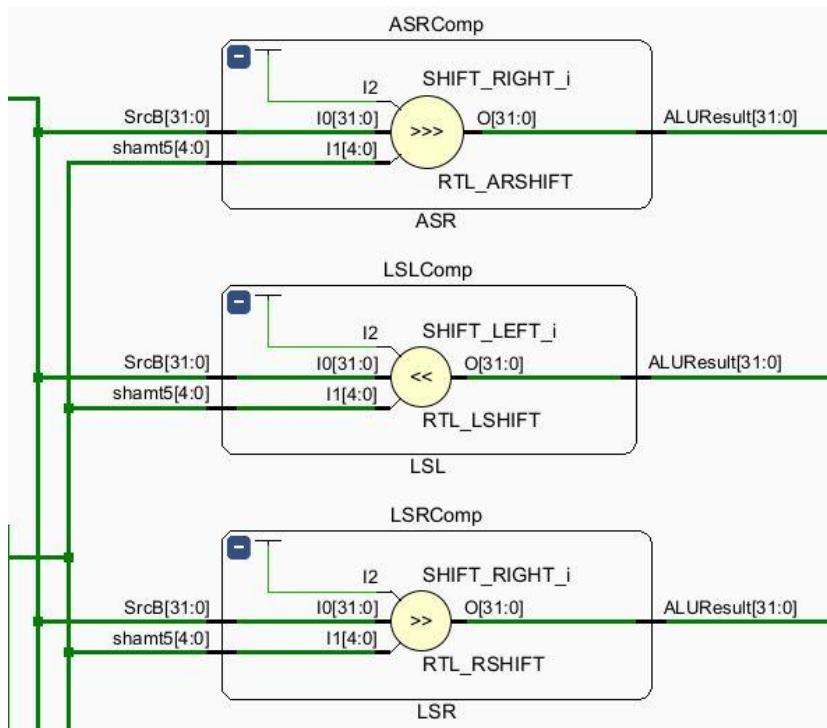


**Εικόνα 31 – Διάγραμμα RTL των μονάδας LogicalOperations**

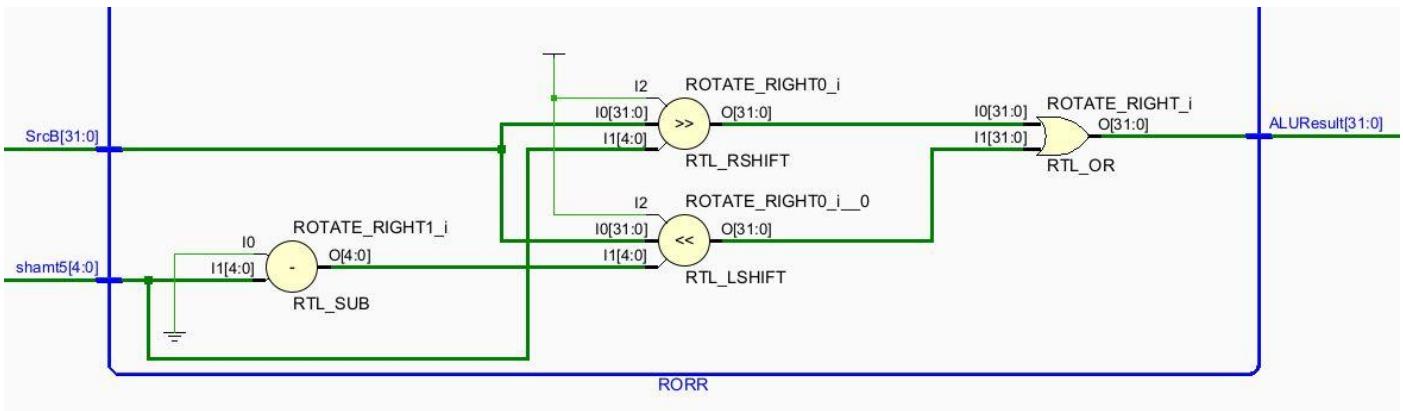
Στο σχηματικό διάγραμμα RTL της μονάδας ShiftOperationsComponent, φαίνεται ότι η σχεδίαση αποτελείται από τέσσερις επιμέρους υπομονάδες:

- ASR, η οποία υλοποιεί την αριθμητική ολίσθηση δεξιά.
- LSL, η οποία υλοποιεί τη λογική ολίσθηση αριστερά.
- LSR, η οποία υλοποιεί τη λογική ολίσθηση δεξιά.
- RORR, η οποία υλοποιεί την κυκλική ολίσθηση δεξιά.

Κάθε υπομονάδα δέχεται ως εισόδους τα σήματα SrcB και shamt5, και επιστρέφει ένα ενδιάμεσο αποτέλεσμα. Στις παρακάτω εικόνες φαίνονται τα σχηματικά διαγράμματα σε επίπεδο RTL των μονάδων ASR, LSL, LSR και RORR.



**Εικόνα 32 – Διάγραμμα RTL των μονάδων ASR, LSL, LSR**



**Εικόνα 33 – Διάγραμμα RTL των μονάδων ROR**

Η επιλογή του τελικού αποτελέσματος πραγματοποιείται μέσω πολυπλέκτη (with-select statement), ο οποίος, ανάλογα με την τιμή του σήματος sel, κατευθύνει στην έξοδο ALUResult την έξοδο της αντίστοιχης υπομονάδας.

Η μονάδα ShiftOperationsComponent προσφέρει μια παραμετροποιήσιμη και ευέλικτη υλοποίηση των πράξεων ολίσθησης, καθώς μέσω των γενικευμένων παραμέτρων N (πλάτος λέξης) και S (πλάτος σήματος ολίσθησης) μπορεί να προσαρμοστεί σε διαφορετικά μεγέθη επεξεργαστών.

Οι πράξεις ολίσθησης που εκτελούνται δεν επηρεάζουν τις σημαίες αριθμητικής κατάστασης (C και V), καθώς δεν αποτελούν αριθμητικές πράξεις, αλλά λειτουργίες σε επίπεδο bit.

#### 1.4.5 ΥΠΟΜΟΝΑΔΑ NORR

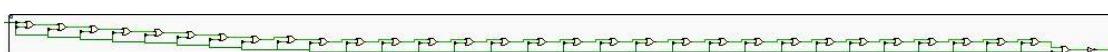
Η μονάδα Norr χρησιμοποιείται για την εκτέλεση της λογικής πράξης NOR σε έναν δυαδικό αριθμό N bits. Πρόκειται για μια απλή αλλά ουσιαστική βοηθητική μονάδα της ALU, η οποία ελέγχει αν όλα τα bits ενός σήματος είναι μηδενικά.

Η λειτουργία της μονάδας βασίζεται στη λογική πράξη NOR (NOT-OR), σύμφωνα με την οποία η έξοδος παίρνει την τιμή '1' μόνο αν όλα τα bits της εισόδου είναι '0'. Σε οποιαδήποτε άλλη περίπτωση (δηλαδή αν έστω ένα bit της εισόδου είναι '1'), η έξοδος γίνεται '0'.

Η μονάδα δέχεται ως είσοδο το σήμα input, που αποτελείται από N bits και αντιστοιχεί στα δεδομένα που θα ελεγχθούν και παράγει ως έξοδο το σήμα output, το οποίο είναι μονό bit και εκφράζει το αποτέλεσμα της λογικής πράξης NOR.

Στο διάγραμμα RTL, η λειτουργία της μονάδας Norr φαίνεται να βασίζεται σε έναν συνδυαστικό βρόχο, ο οποίος πραγματοποιεί διαδοχικό OR μεταξύ όλων των bits της εισόδου. Το αποτέλεσμα των OR στο τέλος αντιστρέφεται (NOT) ώστε να προκύψει η τελική έξοδος.

Στην παρακάτω εικόνα φαίνεται το σχηματικό διάγραμμα σε επίπεδο RTL της μονάδας NORR.



**Εικόνα 34 – Διάγραμμα RTL των μονάδων NORR**

#### 1.4.6 ΧΡΗΣΗ ΠΟΡΩΝ

Η χρήση πόρων από την ALU στο στάδιο του Implementation , ανα component αλλά και συνολικά, φαίνεται στις παρακάτω εικόνες.

Name	1	Slice LUTs (53200)	Slice (13300)	LUT as Logic (53200)	Bonded IOB (200)	
▼ N ALU		601	176	601	109	
▼ I AddSubComp (AddSub)		96	35	96	0	
I add (Adder)		33	10	33	0	
▼ I LogicComp (LogicalOperationsComponent)		120	35	120	0	
I AnddComp (Andd)		32	27	32	0	
I OrrComp (Orr)		32	27	32	0	
I XorrComp (Xorr)		32	27	32	0	
I MoveComp (MoveOperationsComponent)		16	15	16	0	
I NorrComp (Norr)		7	6	7	0	
▼ I ShiftComp (ShiftOperationsComponent)		330	92	330	0	
I ASRComp (ASR)		76	31	76	0	
I LSLComp (LSL)		71	28	71	0	
I LSRCComp (LSR)		71	33	71	0	
I RORRComp (RORR)		80	34	80	0	

**Εικόνα 35 –Χρήση πόρων από την ALU**

Name	1	Slice LUTs (53200)	Slice (13300)	LUT as Logic (53200)	Bonded IOB (200)	
▼ N ALU		1.13%	1.32%	1.13%	54.50%	
▼ I AddSubComp (AddSub)		0.18%	0.26%	0.18%	0.00%	
I add (Adder)		0.06%	0.08%	0.06%	0.00%	
▼ I LogicComp (LogicalOperationsComponent)		0.23%	0.26%	0.23%	0.00%	
I AnddComp (Andd)		0.06%	0.20%	0.06%	0.00%	
I OrrComp (Orr)		0.06%	0.20%	0.06%	0.00%	
I XorrComp (Xorr)		0.06%	0.20%	0.06%	0.00%	
I MoveComp (MoveOperationsComponent)		0.03%	0.11%	0.03%	0.00%	
I NorrComp (Norr)		0.01%	0.05%	0.01%	0.00%	
▼ I ShiftComp (ShiftOperationsComponent)		0.62%	0.69%	0.62%	0.00%	
I ASRComp (ASR)		0.14%	0.23%	0.14%	0.00%	
I LSLComp (LSL)		0.13%	0.21%	0.13%	0.00%	
I LSRCComp (LSR)		0.13%	0.25%	0.13%	0.00%	
I RORRComp (RORR)		0.15%	0.26%	0.15%	0.00%	

**Εικόνα 36 – Ποσοστό χρήσης πόρων από την ALU**

Όπως φαίνεται συνολικά χρησιμοποιείται το 1.13% των LUT του FPGA με την περισσότερη χρήση να γίνεται από την υπομονάδα πράξεων ολίσθησης και τη λιγότερη όπως ήταν αναμενόμενο από την υπομονάδα πορ η οποία υλοποιεί μία απλή λογική πράξη ελέγχου μηδενισμού του αποτελέσματος.

Χρησιμοποιούνται 1.32% των slice με την μεγαλύτερη χρήση να γίνεται από την μονάδα πράξεων ολίσθησης (0.69%) και την μικρότερη από τη μονάδα πρόσθεσης (0.08%).

Όλη η χρήση των LUT προορίζεται για την εκτέλεση λογικών πράξεων μιας και το ποσοστό των LUT as Logic και Slice LUTs είναι ακριβώς ίδιο. Αυτό σημαίνει ότι η ALU περιορίζεται αποκλειστικά σε συνδυαστική λογική. Τέλος φαίνεται ότι χρησιμοποιούνται περίπου οι μισές είσοδοι και έξοδοι της πλακέτας Zynq 7000 ZC702 (xc7z020clg484-1). Αυτό είναι αναμενόμενο, καθώς η ALU διαθέτει πολλαπλές γραμμές εισόδου και εξόδου

(SrcA, SrcB, ALUResult, σημαίες N, Z, C, V, κ.ά.), οι οποίες απαιτούν σημαντικό αριθμό φυσικών pins του FPGA για τη διασύνδεση με το υπόλοιπο σύστημα.

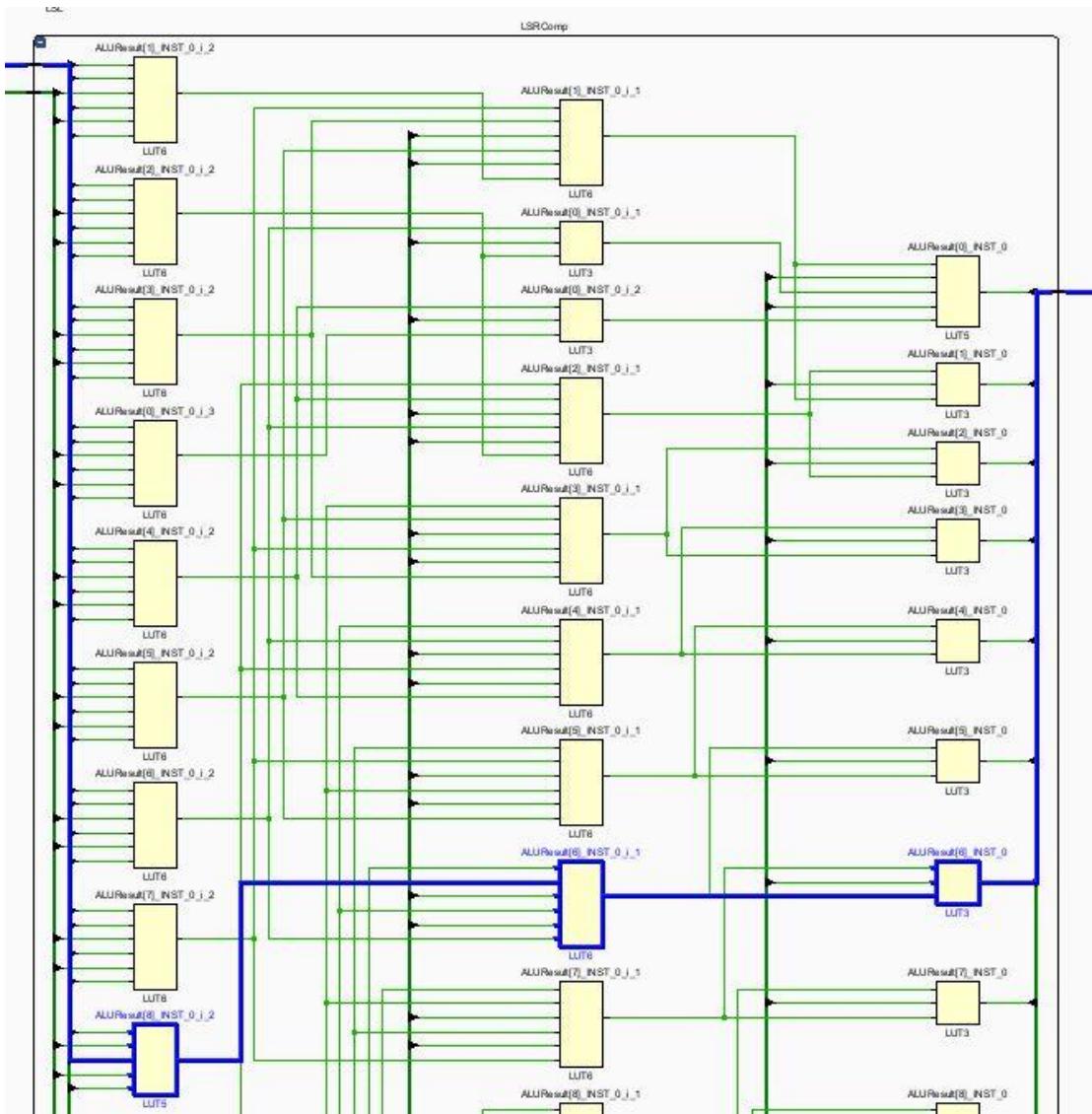
#### 1.4.7 TAXYTHTA LEITOYRGIAΣ ALU

Για να βρούμε την ταχύτητα λειτουργίας της ALU θα πρέπει να βρούμε τον χρόνο διάδοσης του κυκλώματος. Ο χρόνος διάδοσης φαίνεται στα timing reports του Vivado. Στην παρακάτω εικόνα φαίνεται ο χρόνος διάδοσης για Την ALU.

Name	Slack	^ 1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞		9	8	21	SrcB[24]	Z	19.050	4.931	14.119	∞	input port clock			0.000
Path 2	∞		7	6	21	SrcB[24]	ALUResult[4]	18.319	4.646	13.673	∞	input port clock			0.000
Path 3	∞		7	6	18	SrcB[19]	ALUResult[30]	18.294	4.506	13.788	∞	input port clock			0.000
Path 4	∞		7	6	21	SrcB[24]	ALUResult[2]	18.165	4.673	13.492	∞	input port clock			0.000
Path 5	∞		7	6	21	SrcB[24]	ALUResult[6]	18.119	4.633	13.486	∞	input port clock			0.000
Path 6	∞		15	5	19	SrcB[1]	ALUResult[31]	18.109	6.615	11.494	∞	input port clock			0.000
Path 7	∞		7	6	138	shamt5[2]	ALUResult[13]	18.073	4.224	13.850	∞	input port clock			0.000
Path 8	∞		7	6	21	SrcB[24]	ALUResult[14]	18.072	4.425	13.647	∞	input port clock			0.000
Path 9	∞		7	6	21	SrcB[24]	ALUResult[3]	17.966	4.414	13.553	∞	input port clock			0.000
Path 10	∞		15	5	19	SrcB[1]	ALUResult[29]	17.836	6.644	11.192	∞	input port clock			0.000

**Εικόνα 37 – Χρόνος διάδοσης ALU**

Ο μέγιστος χρόνος είναι διάδοσης είναι 19.05ns στο Path 1 το οποίο ξεκινάει από την είσοδο SrcB και καταλήγει στην έξοδο στης σημαίας Z. Από αυτόν τον χρόνο τα 14.119ns δαπανώνται στην καλωδίωση (net delay) και τα 4.931ns στη λογική (logic delay). Το συγκεκριμένο μονοπάτι αφορά την υλοποίηση της εντολής LSR και λεπτομέρεια αυτού φαίνεται στην επόμενη εικόνα.



**Εικόνα 38 – Μονοπάτι μέγιστου χρόνου διάδοσης στην ALU**

Ο ελάχιστος χρόνος μόλυνσης είναι 2.406ns στο Path 11 το οποίο ξεκινάει από την είσοδο ALUControl(1) και καταλήγει στην έξοδο στης σημαίας V. Από αυτόν τον χρόνο τα 1.032ns δαπανώνται στην καλωδίωση (net delay) και τα 1.374ns στη λογική (logic delay). Στην παρακάτω εικόνα φαίνεται ο χρόνος μόλυνσης για την ALU.

Name	Slack	^ 1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 11	∞		3	2	129	ALUControl[1]	V	2.406	1.374	1.032	∞	input port clock			0.000
Path 12	∞		3	2	129	ALUControl[1]	C	2.415	1.441	0.974	∞	input port clock			0.000
Path 13	∞		4	3	128	ALUControl[0]	N	2.669	1.418	1.251	∞	input port clock			0.000
Path 14	∞		6	5	18	SrcB[10]	Z	4.022	1.530	2.492	∞	input port clock			0.000
Path 15	∞		4	3	18	SrcB[10]	ALUResult[10]	4.176	1.426	2.750	∞	input port clock			0.000
Path 16	∞		3	2	34	ALUControl[2]	ALUResult[28]	4.176	1.428	2.748	∞	input port clock			0.000
Path 17	∞		3	2	34	ALUControl[2]	ALUResult[29]	4.185	1.445	2.739	∞	input port clock			0.000
Path 18	∞		4	3	21	SrcB[24]	ALUResult[24]	4.217	1.437	2.780	∞	input port clock			0.000
Path 19	∞		4	3	17	SrcB[5]	ALUResult[5]	4.257	1.439	2.818	∞	input port clock			0.000
Path 20	∞		3	2	34	ALUControl[2]	ALUResult[27]	4.261	1.424	2.837	∞	input port clock			0.000

**Εικόνα 39 – Χρόνος μόλυνσης ALU**

Η ALU είναι συνδυαστικό κύκλωμα και δεν διαθέτει είσοδο ρολογιού. Η μέγιστη καθυστέρηση διάδοσης εκφράζει και την μέγιστη ταχύτητα λειτουργίας του κυκλώματος. Η μέγιστη ταχύτητα διάδοσης είναι 19.05ns και η μέγιστη ταχύτητα προκύπτει από το αντίστροφο του χρόνου αυτού, δηλαδή:  $F_{max} = 1/T_p = 1/19.05\text{ns} = 52.5 \text{ MHz}$ .

Αυτό σημαίνει ότι η ALU μπορεί θεωρητικά να ολοκληρώνει μια πράξη κάθε 19.05 ns, δηλαδή να λειτουργεί με μέγιστη συχνότητα περίπου 52.5 MHz, εφόσον ενταχθεί σε χρονισμένο σύστημα.

Η τιμή αυτή ορίζει τη μέγιστη ταχύτητα λειτουργίας του κυκλώματος, καθώς αντιστοιχεί στο κρίσιμο μονοπάτι (critical path) του σχεδίου — το πιο αργό μονοπάτι μεταξύ εισόδου και εξόδου.

## 1.5 ΣΥΧΝΟΤΗΤΑ ΛΕΙΤΟΥΡΓΙΑΣ ΕΠΕΞΕΡΓΑΣΤΗ

Για την εύρεση της συχνότητας λειτουργίας του επεξεργαστή και την εύρεση της χειρότερης κρίσιμης διαδρομής και της χειρότερης σύντομης διαδρομής θα δουλέψουμε στο επίπεδο Implementation. Στο Implementation επίπεδο το Vivado τρέχει τον κώδικα στο πραγματικό επίπεδο της συγκεκριμένης FPGA πλακέτας που χρησιμοποιούμε έτσι ξέρουμε ακριβώς ποιοι συγκεκριμένα πόροι θα χρησιμοποιηθούν οπότε η ανάλυση χρονισμού είναι ακριβής και αντιπροσωπευτική της πραγματικής λειτουργίας του επεξεργαστή στο υλικό.

Σε επίπεδο Synthesis γίνεται εκτίμηση των πόρων που θα χρειαστούν για την υλοποίηση του επεξεργαστή χωρίς όμως να λαμβάνεται υπόψη η πραγματική φυσική τοποθέτηση των στοιχείων μέσα στο FPGA. Στο στάδιο Synthesis, το Vivado μεταφράζει τον κώδικα VHDL σε ένα λογικό σχέδιο, υπολογίζοντας θεωρητικές καθυστερήσεις βασισμένες μόνο στη λογική συνδεσμολογία και όχι στις πραγματικές αποστάσεις ή στα καλώδια του chip. Έτσι, οι χρόνοι διάδοσης που προκύπτουν στο Synthesis αποτελούν προσεγγιστικές εκτιμήσεις, χρήσιμες για έναν πρώτο έλεγχο, αλλά όχι ρεαλιστικές για την τελική συχνότητα λειτουργίας.

Κατά το στάδιο Implementation, και συγκεκριμένα μέσα από το timing report, το Vivado παρουσιάζει όλα τα χρονικά χαρακτηριστικά του κυκλώματος. Οι πληροφορίες αυτές προκύπτουν με βάση το synthesized design και το αρχείο constrains, στο οποίο ορίζουμε τις απαιτήσεις που θέλουμε να τηρηθούν κατά την υλοποίηση. Στην περίπτωσή μας ο περιορισμός που θέτουμε είναι η περίοδος του ρολογιού. Με βάση αυτά, το Vivado εκτελεί τη διαδικασία place and route, δηλαδή τοποθετεί τα στοιχεία του κυκλώματος σε φυσικούς πόρους του FPGA και τα συνδέει με τέτοιο τρόπο ώστε να ικανοποιούνται οι χρονικοί περιορισμοί, χωρίς να επηρεάζεται η ορθή λειτουργία του συστήματος.

Στόχος της ανάλυσης αυτής είναι να βρεθεί η μέγιστη συχνότητα λειτουργίας του επεξεργαστή, δηλαδή η ελάχιστη περίοδος ρολογιού που μπορούμε να ορίσουμε στα constraints χωρίς να εμφανιστούν χρονικές παραβιάσεις. Οι παραβιάσεις αυτές εμφανίζονται όταν κάποια μονοπάτια δεδομένων δεν προλαβαίνουν να σταθεροποιηθούν μέσα σε έναν κύκλο του ρολογιού, με αποτέλεσμα πιθανές λανθασμένες εγγραφές σε καταχωρητές.

Για να εντοπίσουμε τέτοιες περιπτώσεις, το Vivado υπολογίζει για κάθε διαδρομή μεταξύ καταχωρητών δύο βασικά μεγέθη: τον χρόνο setup και τον χρόνο hold.

Ο χρόνος setup είναι το χρονικό διάστημα που πρέπει η είσοδος ενός καταχωρητή να παραμένει σταθερή πριν από την ακμή του ρολογιού, ώστε η τιμή να καταγραφεί σωστά.

Ο χρόνος hold, αντίθετα, είναι το διάστημα που πρέπει η είσοδος να παραμείνει σταθερή μετά την ακμή του ρολογιού, για να ολοκληρωθεί σωστά η εγγραφή.

Αν η καθυστέρηση διάδοσης ενός μονοπατίου είναι πολύ μεγάλη, μπορεί να μην επαρκεί ο διαθέσιμος χρόνος πριν την ακμή του ρολογιού — αυτό αποτελεί παραβίαση setup. Αντίστοιχα, αν το σήμα αλλάξει πολύ γρήγορα μετά την ακμή, έχουμε παραβίαση hold.

Το Vivado συνοψίζει αυτά τα δεδομένα με δύο δείκτες:

- WNS (Worst Negative Slack): δείχνει το μονοπάτι με τη μικρότερη χρονική "άνεση" ως προς τον setup χρόνο. Αν είναι θετικός ή μηδενικός, ο χρονισμός είναι εντάξει. Αν γίνει αρνητικός, υπάρχει παραβίαση setup.
- WHS (Worst Hold Slack): δείχνει το μονοπάτι με το μικρότερο χρονικό περιθώριο σταθερότητας μετά την ακμή του ρολογιού. Και εδώ, αρνητική τιμή σημαίνει παραβίαση hold.

Συνολικά, για να θεωρείται το κύκλωμα χρονιστικά ορθό, πρέπει και οι δύο τιμές (WNS και WHS) να είναι μη αρνητικές, πράγμα που σημαίνει ότι όλοι οι καταχωρητές λαμβάνουν σωστά τις τιμές τους μέσα στους χρονικούς περιορισμούς που έχουν τεθεί από το constraint του ρολογιού.

Για να βρούμε την βέλτιστη συχνότητα λειτουργίας δοκιμάστηκαν διάφοροι χρονικοί περιορισμοί έως ότου οι δείκτες WNS και WHS να γίνουν οριακά θετικοί. Εν τέλη η μέγιστη συχνότητα προσδιορίστηκε στα 82.305MHz το οποίο αντιστοιχεί σε περίοδο ρολογιού ίση με 12.150ns όπως φαίνεται και στην επόμενη εικόνα.

Name	Waveform	Period (ns)	Frequency (MHz)
CLK	{0.000 6.075}	12.150	82.305

**Εικόνα 40 – Μέγιστη συχνότητα λειτουργίας**

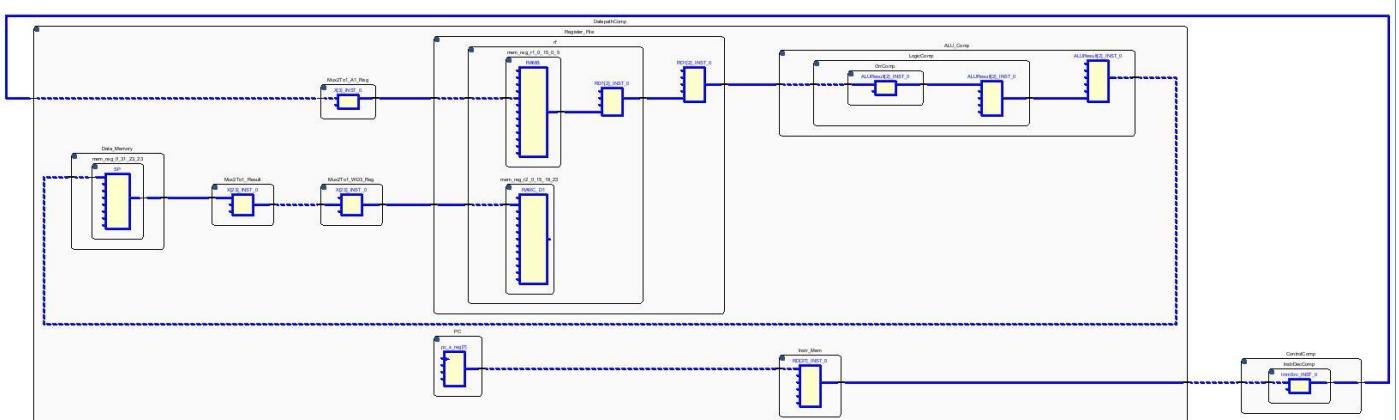
Στην παρακάτω εικόνα φαίνεται το Design Timing Summary.

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0,027 ns	Worst Hold Slack (WHS): 0,608 ns	Worst Pulse Width Slack (WPWS): 4,825 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 772	Total Number of Endpoints: 772	Total Number of Endpoints: 157

All user specified timing constraints are met.

**Εικόνα 41 – Design Timing Summary**

Το Worst Negative Slack υπολογίστηκε στα 0.027 ns και το Worst Hold Slack υπολογίστηκε στα 0.608 ns. Περεταίρω μείωση της περιόδου του clock έδινε αρνητικό WNS. Άρα έχει γίνει βελτιστοποίηση από το Vivado στην κατανομή των πόρων με τέτοιον τρόπο ώστε να δίνει τη μέγιστη δυνατή συχνότητα, δίχως παραβίαση του Setup Time. Το μονοπάτι από το οποίο προκύπτει το WNS, με άλλα λόγια η χειρότερη κρίσιμη διαδρομή φαίνεται παρακάτω.



**Εικόνα 42 – Διάγραμμα χειρότερης κρίσιμης διαδρομής**

Ο μέγιστος χρόνος είναι διάδοσης είναι 11.773ns στο Path 1. Από αυτόν τον χρόνο τα 9.261ns δαπανώνται στην καλωδίωση (net delay) και τα 2.512ns στη λογική (logic delay). Από το μονοπάτι στο επίπεδο Implementation είναι δύσκολο να βγει συμπέρασμα ποια διαδρομή θα μπορούσε να αντιστοιχηθεί σε επίπεδο RTL καθότι σε αυτή τη φάση δίνεται έμφαση στην εξοικονόμηση πόρων με σκοπό τη μείωση του κύκλου. Όπως φαίνεται από το διάγραμμα της χειρότερης διαδρομής αλλά και την παρακάτω εικόνα η διαδρομή ξεκινάει από τον μετρητή προγράμματος και καταλήγει στο αρχείο καταχωρητών παιρνόντα μέσα από την Data Memory. Ως εκ τούτου η κρίσιμη διαδρομή είναι στην εντολή load η οποία παίρνει εγγράφει στο αρχείο καταχωρητών δεδομένων από τη μνήμη δεδομένων.

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.027	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/...23/RAMC_D1/I	11.773	2.512	9.261	12.2
Path 2	0.047	13	96	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/..._5/RAMC_D1/I	11.792	2.190	9.602	12.2
Path 3	0.055	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/...23/RAMC_D1/I	11.744	2.512	9.232	12.2
Path 4	0.080	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/..._12_17/RAMA/I	11.842	2.512	9.330	12.2
Path 5	0.095	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/..._24_29/RAMB/I	11.767	2.527	9.240	12.2
Path 6	0.096	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/...23/RAMB_D1/I	11.725	2.662	9.063	12.2
Path 7	0.145	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/...15_0_5/RAMA/I	11.783	2.531	9.252	12.2
Path 8	0.155	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/..._6_11/RAMA/I	11.734	2.531	9.203	12.2
Path 9	0.194	12	128	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/...15_0_5/RAMA/I	11.733	2.531	9.202	12.2
Path 10	0.202	13	96	DatapathComp/PC/pc_s_reg[7]/C	DatapathComp/...29/RAMC_D1/I	11.596	2.242	9.354	12.2

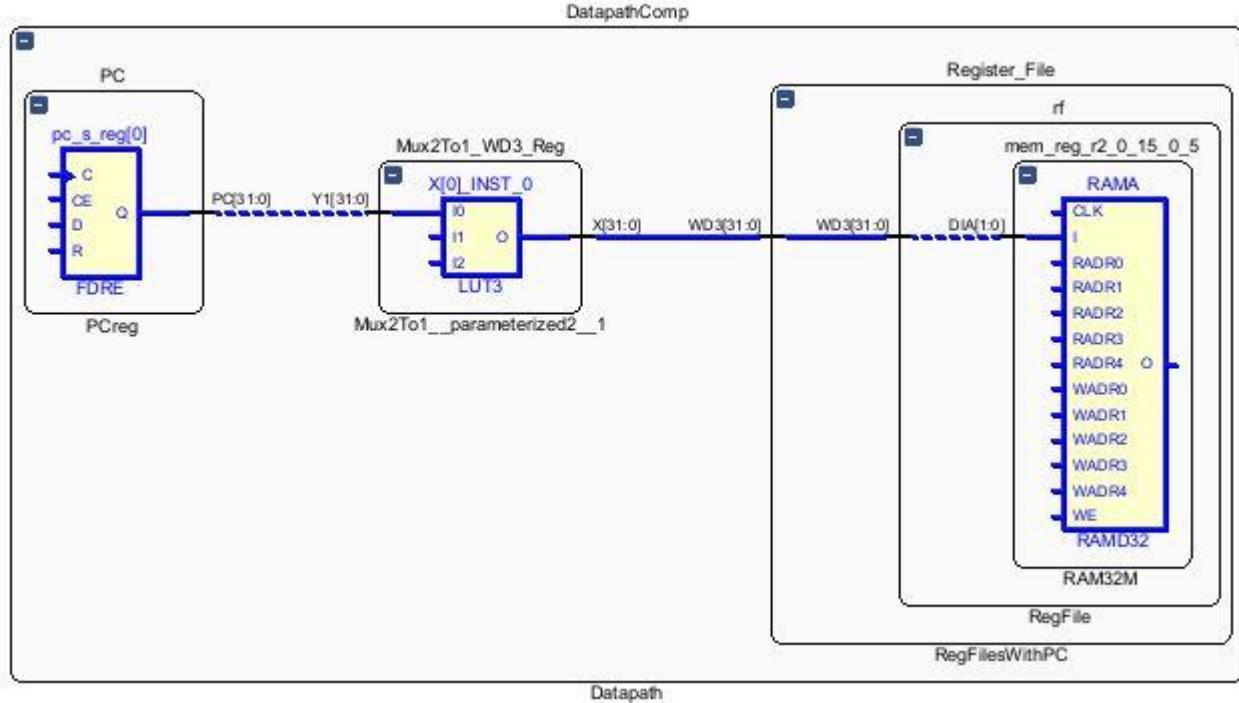
### Εικόνα 43 – Μέγιστοι χρόνοι διάδοσης επεξεργαστή

Το slack που εμφανίζεται στο timing report του Vivado εκφράζει το χρονικό περιθώριο που έχει το κύκλωμα ώστε τα δεδομένα να φτάσουν εγκαίρως στον επόμενο καταχωρητή, σε σχέση με την περίοδο του ρολογιού. Ουσιαστικά δείχνει τη διαφορά ανάμεσα στον χρόνο που έχουν τα δεδομένα διαθέσιμο και στον χρόνο που πραγματικά χρειάζονται για να διαδοθούν μέσα στο κύκλωμα. Όταν το slack είναι θετικό σημαίνει ότι το κύκλωμα προλαβαίνει να ολοκληρώσει τη διάδοση των σημάτων πριν από την επόμενη ακμή του ρολογιού, άρα λειτουργεί χρονιστικά σωστά. Αντίθετα, αρνητική τιμή slack σημαίνει ότι κάποια διαδρομή καθυστερεί περισσότερο από όσο επιτρέπεται και συνεπώς υπάρχει χρονιστική παραβίαση. Συνολικά, το slack δείχνει το περιθώριο ασφαλείας του κυκλώματος ως προς τη συχνότητα λειτουργίας του και χρησιμοποιείται για να αξιολογηθεί αν η υλοποίηση είναι χρονιστικά ορθή.

Ο υπολογισμός του slack στο Vivado γίνεται μέσα από τη ανάλυση χρονισμού όπου εξετάζονται όλα τα μονοπάτια μετάδοσης δεδομένων μεταξύ των καταχωρητών του κυκλώματος. Για κάθε μονοπάτι το εργαλείο υπολογίζει δύο χρόνους: τον arrival time, δηλαδή τον πραγματικό χρόνο που χρειάζονται τα δεδομένα για να διαδοθούν από τον έναν καταχωρητή στον επόμενο, και τον required time, που είναι ο μέγιστος επιτρεπόμενος χρόνος για να φτάσουν τα δεδομένα πριν από την επόμενη ακμή του ρολογιού. Το slack προκύπτει ως η διαφορά αυτών των δύο μεγεθών, σύμφωνα με τη σχέση Slack = Required Time – Arrival Time.

Στη συγκεκριμένη περίπτωση ο μικρότερο slack εμφανίζεται στο Path 1 και ισούται με 0.027ns το οποίο είναι και το Worst Negative Slack.

Ο ελάχιστος χρόνος διάδοσης είναι 0.774ns στο Path 11. Από αυτόν τον χρόνο τα 0.588ns δαπανώνται στην καλωδίωση (net delay) και τα 0.186ns στη λογική (logic delay). Η συγκεκριμένη διαδρομή είναι πιο σύντομη και ξεκινά από τον καταχωρητή του μετρητή προγράμματος περνάει από τον πολυπλέκτη WD3 και καταλήγει στο αρχείο καταχωρητών. Βάσει της διαδρομής η εντολής που εκτελείται στη διαδρομή με τον ελάχιστο χρόνο διάδοσης είναι η BL η οποία δεν περνάει από την ALU, ούτε από τη μνήμη δεδομένων.



**Εικόνα 44 – Διαδρομή με τον ελάχιστο χρόνο διάδοσης επεξεργαστή**

Name	Slack ^ 1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 11	0.608	1	5	DatapathComp/PC/pc_s_reg[0]/C	DatapathComp/...15_0_5/RAMA/I	0.774	0.186	0.588
Path 12	0.617	1	5	DatapathComp/PC/pc_s_reg[0]/C	DatapathComp/...15_0_5/RAMA/I	0.782	0.186	0.596
Path 13	0.618	1	5	DatapathComp/PC/pc_s_reg[0]/C	DatapathComp/PC/pc_s_reg[0]/D	0.709	0.186	0.523
Path 14	0.643	2	3	DatapathComp/...31_8_8/SP/CLK	DatapathComp/...6_11/RAMB/I	0.824	0.478	0.346
Path 15	0.651	2	4	DatapathComp/...30_30/SP/CLK	DatapathComp/...15_30_31/SP/I	0.833	0.476	0.357
Path 16	0.681	2	4	DatapathComp/...31_31/SP/CLK	DatapathComp/...0_31_0/SP/I	0.819	0.483	0.336
Path 17	0.685	2	4	DatapathComp/...31_31/SP/CLK	DatapathComp/...0_31_0/DP/I	0.819	0.483	0.336
Path 18	0.687	2	4	DatapathComp/...31_31/SP/CLK	DatapathComp/...0_31_0/SP/I	0.845	0.483	0.362
Path 19	0.699	2	3	DatapathComp/...13_13/SP/CLK	DatapathComp/...17/RAMA_D1/I	0.832	0.483	0.349
Path 20	0.705	2	3	DatapathComp/...12_12/SP/CLK	DatapathComp/...c_s_reg[12]/D	0.829	0.476	0.353

**Εικόνα 45 – Ελάχιστοι χρόνοι διάδοσης επεξεργαστή**

## 2. ΕΠΑΛΗΘΕΥΣΗ ΤΗΣ ΟΡΘΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑΣ

Η επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή είναι απαραίτητη για να διασφαλιστεί ότι το κύκλωμα εκτελεί σωστά όλες τις λειτουργίες για τις οποίες σχεδιάστηκε. Μέσω προσομοιώσεων και ελέγχων, επιβεβαιώνεται τόσο η λειτουργική συμπεριφορά του επεξεργαστή όσο και η χρονική του ακρίβεια, πριν αυτός υλοποιηθεί στο FPGA.

### 2.1 ΔΗΜΙΟΥΡΓΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ ΣΕ ASSEMBLY

Για την επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή δημιουργησα ένα πρόγραμμα το οποίο να δοκιμάζει όλες τις εντολές καθώς και την εκτέλεση υπό συνθήκη. Το πρόγραμμα είναι το εξής:

MAIN:

```
MOV R0, #20
MOV R1, R0
NOP
STR R1, [R0, #2]
STR R1, [R0, #-2]
LDR R2, [R0, #2]
LDR R3, [R0, #-2]
SUB R0, R1, #19
ADD R4, R1, #30
SUB R2, R0, R1
ADD R1, R0, R3
AND R3, R0, R1
AND R3, R1, #15
EOR R3, R0, R1
EOR R3, R1, #15
ORR R3, R0, R1
ORR R3, R1, #15
LSL R3, R1, #2
LSR R1, R3, #2
ASR R1, R2, #2
ROR R1, R3, #2
MVN R0, #2
MVN R4, R3
CMP R2, #15
BEQ COROUTINE_EQ
BLNE COROUTINE_NE
B MAIN
```

COROUTINE\_NE:

```
ADDS R0, R1, #-1
BCC COROUTINE_CC
BCS COROUTINE_CS
```

COROUTINE\_CS:

```
SUBS R0, R2, R1
BPL COROUTINE_PL
BMI COROUTINE_MI
```

#### *COROUTINE\_MI:*

```
SUBS R0, R1, #21
BVS COROUTINE_VS
BVC COROUTINE_VC
```

#### *COROUTINE\_VC:*

```
BHI COROUTINE_HI
BLS COROUTINE_LS
```

#### *COROUTINE\_LS:*

```
ADdge R4, R1, #30
ADDGT R4, R1, #30
ADDLT R4, R1, #30
MOVE PC, R14
```

#### *COROUTINE\_EQ:*

```
ADDS R0, R1, #-1
```

#### *COROUTINE\_CC:*

```
ADDS R0, R1, #-1
```

#### *COROUTINE\_PL:*

```
ADDS R0, R1, #-1
```

#### *COROUTINE\_VS:*

```
ADDS R0, R1, #-1
```

#### *COROUTINE\_HI:*

```
ADDS R0, R1, #-1
```

Η δομή της εφαρμογής είναι τέτοια ώστε να ελέγχει αρχικά όλες τις εντολές επεξεργασίας δεδομένων και μνήμης, οι οποίες εκτελούνται σειριακά και χωρίς συνθήκη, και στη συνέχεια να δοκιμάζει τις εντολές διακλάδωσης σε συνδυασμό με εκτέλεση υπό συνθήκη. Με αυτόν τον τρόπο ελέγχονται όλα τα μνημονικά του επεξεργαστή — κάποια εκτελούνται, ενώ άλλα παραλείπονται ανάλογα με την κατάσταση των σημαιών, εξασφαλίζοντας έτσι τον έλεγχο της σωστής λειτουργίας του μηχανισμού συνθηκών.

Αρχικά, στη ρουτίνα MAIN, εκτελούνται εντολές MOV για την αρχικοποίηση των καταχωρητών με συγκεκριμένες τιμές, ώστε να μπορούν στη συνέχεια να πραγματοποιηθούν πράξεις μεταξύ τους. Έπειτα εκτελούνται εντολές STR και LDR, οι οποίες ελέγχουν τη σωστή λειτουργία της μνήμης δεδομένων, αλλά και την ορθή αποθήκευση και ανάκτηση τιμών από αυτή. Στο σημείο αυτό πραγματοποιείται δοκιμή τόσο με θετικό όσο και με αρνητικό offset, ώστε να επιβεβαιωθεί η σωστή λειτουργία της διευθυνσιοδότησης μνήμης και στις δύο περιπτώσεις.

Ακολουθούν οι υπόλοιπες εντολές επεξεργασίας δεδομένων μεταξύ καταχωρητών και μεταξύ καταχωρητών και σταθερών τιμών (immediates), όπως οι ADD, SUB, AND, EOR, ORR και οι εντολές ολίσθησης (LSL, LSR, ASR, ROR). Στη συνέχεια εκτελούνται εντολές υπό συνθήκη, οι οποίες δοκιμάζουν τη σωστή ενεργοποίηση και απενεργοποίηση των σημαιών κατάστασης (flags), καθώς και εντολές διακλάδωσης που οδηγούν στην εκτέλεση διαφορετικών υπορουτινών.

Τέλος, με την εντολή διακλάδωσης στο τέλος της εφαρμογής, το πρόγραμμα επανέρχεται στην αρχή (MAIN), επιτρέποντας τη συνεχή επανάληψη της εκτέλεσης και τη διαρκή επαλήθευση της λειτουργίας του επεξεργαστή.

H εκτέλεση ξεκινάει από τον κλάδο MAIN.

Η πρώτη εντολή αποθηκεύει στον R0 την τιμή 20, ενώ η δεύτερη αντιγράφει την τιμή αυτή στον R1, άρα και οι δύο καταχωρητές περιέχουν την ίδια τιμή. Η εντολή NOP δεν επηρεάζει το αποτέλεσμα, καθώς δεν πραγματοποιεί καμία ενέργεια και χρησιμοποιείται για λόγους χρονισμού.

Στη συνέχεια εκτελούνται οι εντολές STR, οι οποίες αποθηκεύουν το περιεχόμενο του R1 (20) στη μνήμη, με βάση τον R0 και offset +2 και -2 αντίστοιχα. Με τον τρόπο αυτό ελέγχεται η σωστή λειτουργία της μνήμης δεδομένων τόσο με θετικό όσο και με αρνητικό offset.

Ακολουθούν οι εντολές LDR, οι οποίες διαβάζουν ξανά από τις ίδιες θέσεις μνήμης και φορτώνουν τις τιμές στους R2 και R3, επαληθεύοντας ότι οι προηγούμενες εγγραφές πραγματοποιήθηκαν σωστά.

Η επόμενη ομάδα εντολών εκτελεί αριθμητικές πράξεις. Η εντολή SUB R0, R1, #19 υπολογίζει  $20 - 19 = 1$ , οπότε ο R0 παίρνει την τιμή 1. Στη συνέχεια η ADD R4, R1, #30 αποθηκεύει στον R4 την τιμή 50 ( $20 + 30$ ). Η SUB R2, R0, R1 δίνει αποτέλεσμα -19, ενώ η ADD R1, R0, R3 ενημερώνει τον R1 με το άθροισμα των δύο τιμών, επιβεβαιώνοντας ότι η ALU εκτελεί σωστά πρόσθεση και αφαίρεση με καταχωρητές και άμεσες τιμές. Ακολουθούν οι λογικές πράξεις AND, EOR και ORR, οι οποίες ελέγχουν τη σωστή λειτουργία των bitwize πράξεων τόσο μεταξύ δύο καταχωρητών όσο και μεταξύ καταχωρητή και σταθερής τιμής. Οι πράξεις αυτές επιβεβαιώνουν τη σωστή διαχείριση των bits στην ALU.

Έπειτα, εκτελούνται οι εντολές ολίσθησης LSL, LSR, ASR και ROR, οι οποίες ελέγχουν τη λειτουργία της μονάδας μετατοπίσεων. Για παράδειγμα, η LSL R3, R1, #2 μετατοπίζει αριστερά τα bits του R1 κατά δύο θέσεις, ενώ η ASR και η ROR επιβεβαιώνουν τη σωστή λειτουργία ολίσθησης με διατήρηση ή περιστροφή των bits.

Στη συνέχεια οι εντολές MVN R0, #2 και MVN R4, R3 ελέγχουν τη σωστή λειτουργία του συμπληρώματος ως προς ένα (bitwise NOT), αντιστρέφοντας όλα τα bits του τελεστή.

Η εντολή CMP R2, #15 συγκρίνει το περιεχόμενο του R2 με την τιμή 15 και ενημερώνει τις σημαίες συνθήκης, ανάλογα με το αποτέλεσμα της σύγκρισης. Ανάλογα με τις σημαίες, ακολουθεί σειρά εντολών διακλάδωσης υπό συνθήκη (BEQ, BLNE, BCC, BCS, κ.λπ.) που ελέγχουν την ορθή λειτουργία του μηχανισμού εκτέλεσης υπό συνθήκη.

Στον κλάδο COROUTINE\_NE, εκτελείται η εντολή ADDS, η οποία ενημερώνει τις σημαίες, και ακολουθούν εντολές διακλάδωσης που ελέγχουν τις σημαίες Carry (C) και Zero (Z). Αντίστοιχα, στους υπόλοιπους κλάδους (COROUTINE\_CS, MI, VC, LS), εκτελούνται αριθμητικές εντολές με πρόθεμα S, ώστε να ενημερώνονται οι σημαίες συνθήκης, και ακολουθούν διακλαδώσεις που ελέγχουν συνθήκες όπως Positive/Negative (PL, MI), Overflow (VS, VC), Greater/Less (GE, LT) και Higher/Lower (HI, LS).

Με αυτόν τον τρόπο ελέγχεται η σωστή λειτουργία όλων των σημαιών (N, Z, C, V) καθώς και η ακρίβεια της εκτέλεσης υπό συνθήκη. Η τελευταία εντολή της ρουτίνας LS επαναφέρει τον PC στην κύρια ρουτίνα (MAIN), επανεκκινώντας το πρόγραμμα και επιτρέποντας τη συνεχή λειτουργική επαλήθευση του επεξεργαστή.

Στον παρακάτω πίνακα παρουσιάζονται οι εντολές που εκτελούνται, η σειρά εκτέλεσής τους (a.a.), καθώς και οι τιμές ορισμένων βασικών σημάτων του επεξεργαστή, όπως:

- PC (Program Counter),
- Instr (τρέχουσα εντολή),
- ALUResult (αποτέλεσμα της ALU),
- WriteData (δεδομένα προς εγγραφή),
- Result (τελικό αποτέλεσμα),
- και οι σημαίες N, V, Z, C.

Για την παρακολούθηση αυτών των σημάτων κατά τη φάση δοκιμής του επεξεργαστή, προστέθηκαν πρόσθετες έξοδοι στο top-level του κυκλώματος πέραν αυτών που ζητούνταν στην εκφώνηση της άσκησης.

Ο επιπλέον κώδικας που υλοποιεί αυτές τις εξόδους έχει συμπεριληφθεί με σχολιασμό ώστε να είναι σαφής η χρήση του και να μην επηρεάζει τη βασική λειτουργία του επεξεργαστή.

Επιπλέον στο πίνακα έχουν προστεθεί οι τιμές των καταχωρητών που χρησιμοποιούνται σε κάθε εντολή.  
Οι εντολές οι οποίες δε εκτελούνται επειδή δεν ικανοποιείται η συνθήκη εκτέλεσης δεν φαίνονται στον πίνακα.

<u>α.α.</u>	<u>Εντολη</u>	<u>PC</u>	<u>Instr</u>	<u>ALUResult</u>	<u>WriteData</u>	<u>Result</u>	<u>Σχόλια</u>	<u>R0</u>	<u>R1</u>	<u>R2</u>	<u>R3</u>	<u>R4</u>	<u>N</u>	<u>Z</u>	<u>V</u>	<u>C</u>
<b>MAIN</b>																
1	MOV R0, #20;	0	e3a00014	20	U	20		20	-	-	-	-	-	-	-	-
2	MOV R1, R0;	4	e1a01000	20	20	20		20	20	-	-	-	-	-	-	-
3	NOP	8	e1a01000	20	20	20		20	20	-	-	-	-	-	-	-
4	STR R1, [R0,2]	12	e5801002	22	20	22		20	20	-	-	-	-	-	-	-
5	STR R1, [R0,-2]	16	e5001002	18	20	18		20	20	-	-	-	-	-	-	-
6	LDR R2, [R0,2]	20	e5902002	22	U	20		20	20	20	-	-	-	-	-	-
7	LDR R3, [R0,-2]	24	e5103002	18	20	20		20	20	20	20	-	-	-	-	-
8	SUB R0, R1, 19	28	e2410013	1	20	1		1	20	20	20	-	-	-	-	-
9	ADD R4, R1, 30	32	e281401e	50	U	50		1	20	20	20	50	-	-	-	-
10	SUB R2, R0, R1	36	e0402001	-19	20	-19		1	20	-19	20	50	-	-	-	-
11	ADD R1, R0, R3	40	e0801003	21	20	21		1	21	-19	20	50	-	-	-	-
12	AND R3, R0, R1	44	e0003001	1	21	1		1	21	-19	1	50	-	-	-	-
13	AND R3, R1, #15	48	e201300f	5	56	5		1	21	-19	5	50	-	-	-	-
14	XOR R3, R0, R1	52	e0203001	20	21	20		1	21	-19	20	50	-	-	-	-
15	XOR R3, R1, #15	56	e221300f	26	64	26		1	21	-19	26	50	-	-	-	-
16	OR R3, R0, R1	60	e1803001	21	21	21		1	21	-19	21	50	-	-	-	-
17	OR R3, R1, #15	64	e381300f	31	72	31		1	21	-19	31	50	-	-	-	-
18	LSL R3, R1, #2	68	e1a03101	84	21	84		1	21	-19	84	50	-	-	-	-
19	LSR R1, R3, #2	72	e1a01123	21	84	21		1	21	-19	84	50	-	-	-	-
20	ASR R1, R2, #2	76	e1a01142	-5	-19	-5		1	-5	-19	84	50	-	-	-	-
21	ROR R1, R3, #2	80	e1a01163	21	84	21		1	21	-19	84	50	-	-	-	-
22	MVN R0, #2	84	e3e00002	-3	-19	-3		-3	21	-19	84	50	-	-	-	-
23	MVN R4, R3	88	e1e04003	-85	84	-85		-3	21	-19	84	-85	-	-	-	-
24	CMP R2, #15	92	e352000f	-34	100	-34		-3	21	-19	84	-85	1	0	0	1
25	BEQ COROUTINE_EQ	96	0a000010	168	-3	168		-3	21	-19	84	-85	1	0	0	1
26	BLNE COROUTINE_NE	100	1b000000	108	-3	108		-3	21	-19	84	-85	1	0	0	1
42	B MAIN	104	eaffffe4	0	51	0		0	21	-19	84	51	0	0	1	1
<b>COROUTINE_NE</b>																
27	ADDS R0, R1, #-1	108	e2510001	20	21	20		20	21	-19	84	-85	0	0	0	1

28	BCC COROUTINE_CC	112	3a00000d	172	U	172		20	21	-19	84	-85	0	0	0	1
29	BCS COROUTINE_CS	116	2affffff	120	124	120		20	21	-19	84	-85	0	0	0	1
<b>COROUTINE CS</b>																
30	SUBS R0, R2, R1	120	e0520001	-40	21	-40		-40	21	-19	84	-85	1	0	0	1
31	BPL COROUTINE_PL	124	5a00000b	176	U	176		-40	21	-19	84	-85	1	0	0	1
32	BMI COROUTINE_MI	128	4affffff	132	136	132		-40	21	-19	84	-85	1	0	0	1
<b>COROUTINE MI</b>																
33	SUBS R0, R1, #21	132	e2510015	0	U	0		0	21	-19	84	-85	0	1	0	1
34	BVS COROUTINE_VS	136	6a000009	180	U	180		0	21	-19	84	-85	0	1	0	1
35	BVCCOROUTINE_VC	140	7affffff	144	148	144		0	21	-19	84	-85	0	1	0	1
<b>COROUTINE VC</b>																
36	BLS COROUTINE_HI	144	8a000008	184	U	184		0	21	-19	84	-85	0	1	0	1
37	BHI COROUTINE_LS	148	9affffff	152	156	152		0	21	-19	84	-85	0	1	0	1
<b>COROUTINE LS</b>																
38	ADDGE R4, R1, 30	152	a281401e	51	104	51		0	21	-19	84	-85	0	1	0	1
39	ADDDT R4, R1, 30	156	c281401e	51	104	51		0	21	-19	84	-85	0	1	0	1
40	ADDLT R4, R1, 30	160	b281401e	51	104	51		0	21	-19	84	51	0	1	0	1
41	MOVLE PC, R14	164	d1a0f00e	104	104	104		0	21	-19	84	51	0	1	0	1

**Πίνακας 21 - Εντολές και αντίστοιχες τιμές σημάτων κατά τη δοκιμή του επεξεργαστή**

## 2.2 ΔΙΑΓΡΑΜΜΑΤΑ ΧΡΟΝΙΣΜΟΥ ALU

Για την επαλήθευση της ορθής σχεδίασης και λειτουργίας της ALU, δημιουργήθηκε ένα testbench που δοκιμάζει όλες τις λειτουργίες της αριθμητικής-λογικής μονάδας, καθώς και τις σημαίες κατάστασης (N, Z, C, V). Το testbench περιλαμβάνει δοκιμές για αριθμητικές πράξεις, λογικές πράξεις, πράξεις μετατόπισης και πράξεις μεταφοράς, με διαφορετικές τιμές εισόδων για να επαληθευτεί η σωστή συμπεριφορά σε όλες τις περιπτώσεις.

Η δομή του testbench είναι τέτοια ώστε να ελέγχει αρχικά τις πράξεις μετατόπισης (ROR, ASR, LSR, LSL), τις οποίες εκτελεί με διάφορες τιμές και ποσότητες μετατόπισης, και στη συνέχεια προχωρά στις λογικές πράξεις (AND, OR, XOR) και στις αριθμητικές πράξεις (ADD, SUB). Τέλος, ελέγχονται οι πράξεις μεταφοράς (MOV, MVN) και η NOP. Με αυτόν τον τρόπο, ελέγχονται όλες οι λειτουργίες της ALU — κάθε πράξη εκτελείται με συγκεκριμένες εισόδους και αναμένονται συγκεκριμένες έξοδοι και σημαίες.

Το κυρίως τμήμα κώδικα του testbench είναι ο κάτωθι:

```
report "TESTING ROR";
-- Expected result: X"78123456"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"12345678";
shamt5_in    <= "01000";
ALUControl_in <= "1111";
wait for 1*CLK_period;
-- Expected result: X"80000000"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000001";
shamt5_in    <= "00001";
ALUControl_in <= "1111";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000000";
shamt5_in    <= "00001";
ALUControl_in <= "1111";
wait for 1*CLK_period;

report "TESTING ASR";
-- Expected result: X"38000000"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"70000000";
shamt5_in    <= "00001";
ALUControl_in <= "1100";
wait for 1*CLK_period;
-- Expected result: X"FFFFFFFF"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"80000000";
```

```

shamt5_in    <= "1111";
ALUControl_in <= "1100";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000001";
shamt5_in    <= "00001";
ALUControl_in <= "1100";
wait for 1*CLK_period;

report "TESTING LSR";
-- Expected result: X"78000000"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"F0000000";
shamt5_in    <= "00001";
ALUControl_in <= "1110";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000001";
shamt5_in    <= "00001";
ALUControl_in <= "1110";
wait for 1*CLK_period;

report "TESTING LSL";
-- Expected result: X"00000002"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000001";
shamt5_in    <= "00001";
ALUControl_in <= "1101";
wait for 1*CLK_period;
-- Expected result X"80000000"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"40000000";
shamt5_in    <= "00001";
ALUControl_in <= "1101";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000000";
shamt5_in    <= "00100";
ALUControl_in <= "1101";

```

```

wait for 1*CLK_period;

report "TESTING AND";
-- Expected result: X"0A0A0A0A"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= X"AAAAAAA";
SrcB_in      <= X"OF0F0F0F";
ALUControl_in <= "1010";
wait for 1*CLK_period;
-- Expected result: X"80000000"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= X"80000000";
SrcB_in      <= X"F0000000";
ALUControl_in <= "1010";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= X"12345678";
SrcB_in      <= X"00000000";
ALUControl_in <= "1010";
wait for 1*CLK_period;

report "TESTING OR";
-- Expected result: X"1F1F1F1F"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= X"OF0F0F0F";
SrcB_in      <= X"10101010";
ALUControl_in <= "1011";
wait for 1*CLK_period;
-- Expected result: X"80000000"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= X"80000000";
SrcB_in      <= X"00000000";
ALUControl_in <= "1011";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= X"00000000";
SrcB_in      <= X"00000000";
ALUControl_in <= "1011";
wait for 1*CLK_period;

report "TESTING XOR";
-- Expected result: X"1F1F1F1F"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= X"OF0F0F0F";
SrcB_in      <= X"10101010";
ALUControl_in <= "1000";

```

```

wait for 1*CLK_period;
-- Expected result: X"80000000"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= X"80000000";
SrcB_in      <= X"00000000";
ALUControl_in <= "1000";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= X"00000000";
SrcB_in      <= X"00000000";
ALUControl_in <= "1000";
wait for 1*CLK_period;

report "TESTING ADD";
-- Expected result: X"00000015"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= X"00000005";
SrcB_in      <= X"00000010";
ALUControl_in <= "0000";
wait for 1*CLK_period;
-- Expected result: X"80000000"
-- Expected flags. N:1 , Z:0, V:1, C:0
SrcA_in      <= X"40000000";
SrcB_in      <= X"40000000";
ALUControl_in <= "0000";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:1
SrcA_in      <= X"FFFFFFF";
SrcB_in      <= X"00000001";
ALUControl_in <= "0000";
wait for 1*CLK_period;

report "TESTING SUB";
-- Expected result: X"00000008"
-- Expected flags. N:0 , Z:0, V:0, C:1
SrcA_in      <= X"0000000A";
SrcB_in      <= X"00000002";
ALUControl_in <= "0001"; -- SUB
wait for 1*CLK_period;
-- Expected result: X"80000000"
-- Expected flags. N:1 , Z:0, V:1, C:0
SrcA_in      <= X"00000000";
SrcB_in      <= X"80000000";
ALUControl_in <= "0001";
wait for 1*CLK_period;
-- Expected result: X"00000000"

```

```

-- Expected flags. N:0 , Z:1, V:0, C:1
SrcA_in      <= X"00000000";
SrcB_in      <= X"00000000";
ALUControl_in <= "0001";
wait for 1*CLK_period;

report "TESTING MOV";
-- Expected result: X"2AAAAAAA"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"2AAAAAAA";
ALUControl_in <= "0100";
wait for 1*CLK_period;
-- Expected result: X"80000000"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"80000000";
ALUControl_in <= "0100";
wait for 1*CLK_period;
-- Expected result: X"00000000"
-- Expected flags. N:0 , Z:1, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000000";
ALUControl_in <= "0100";
wait for 1*CLK_period;

report "TESTING MVN";
-- Expected result: X"55555555"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"AAAAAAA";
ALUControl_in <= "0101";
wait for 1*CLK_period;
-- Expected result: X"7FFFFFFF"
-- Expected flags. N:0 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"80000000";
ALUControl_in <= "0101";
wait for 1*CLK_period;
-- Expected result: X"FFFFFFF"
-- Expected flags. N:1 , Z:0, V:0, C:0
SrcA_in      <= (others => 'X');
SrcB_in      <= X"00000000";
ALUControl_in <= "0101";
wait for 1*CLK_period;

report "TESTING NOP";
-- Expected result: X"00000001"

```

-- Expected flags. N:0 , Z:0, V:0, C:0

*SrcA\_in* <= (others => 'X');

*SrcB\_in* <= X"00000001";

*ALUControl\_in* <= "0111";

wait for 1\*CLK\_period;

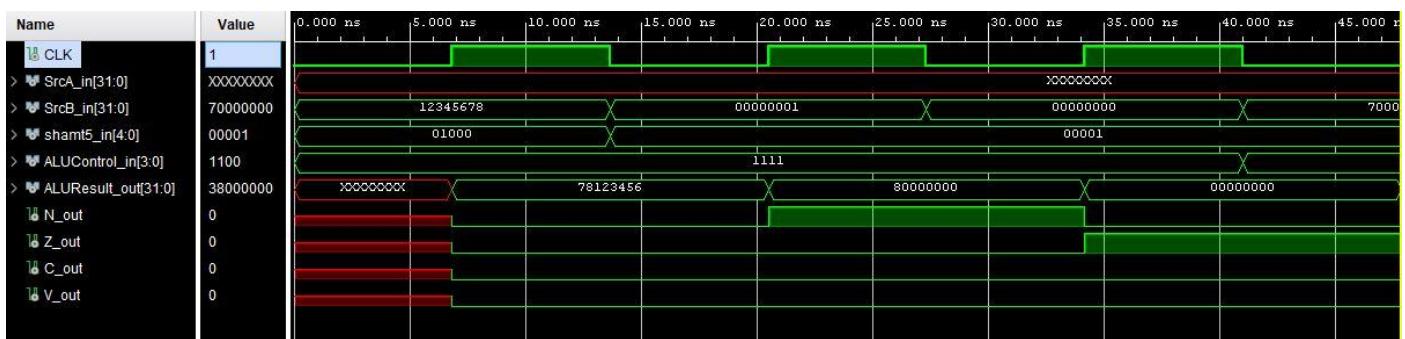
report "TESTS COMPLETED";

## 2.2.1 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ ROR

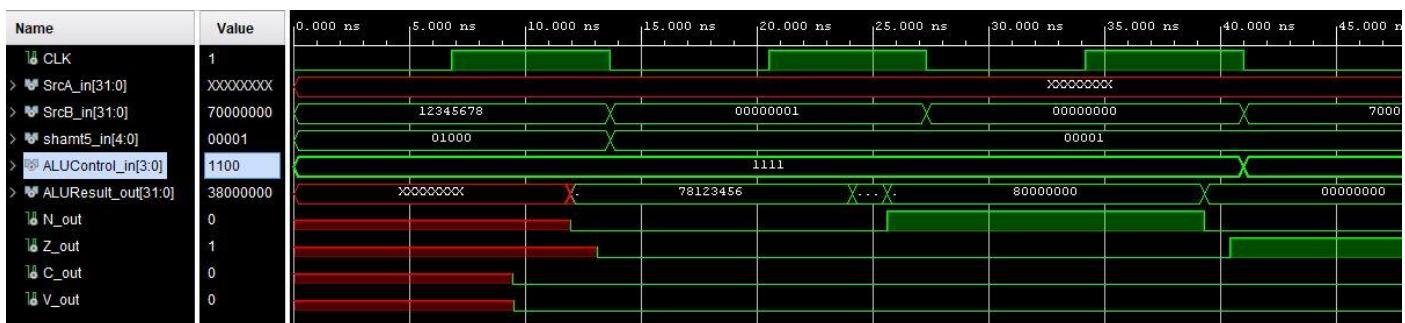
Η εντολή ROR αντιστοιχεί σε τιμή σήματος ALUControl = "1111" και πραγματοποιεί περιστροφή των bits της τιμής του SrcB προς τα δεξιά, κατά αριθμό θέσεων που ορίζεται από το σήμα shamt5. Στο testbench εκτελούνται τρεις διαφορετικές δοκιμές της εντολής.

Στην πρώτη περίπτωση δεν αναμένεται να ενεργοποιηθεί καμία σημαία, στη δεύτερη αναμένεται η ενεργοποίηση της σημαίας N (Negative), καθώς το αποτέλεσμα έχει MSB ίσο με 1, ενώ στην τρίτη περίπτωση αναμένεται να ενεργοποιηθεί η σημαία Z (Zero), αφού το αποτέλεσμα της πράξης είναι μηδενικό.

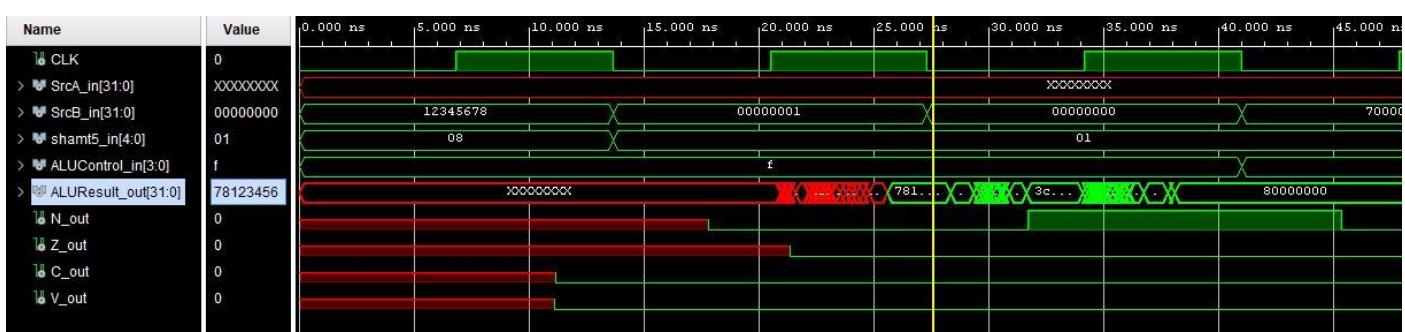
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



Εικόνα 46 – Αποτέλεσμα εντολής ROR σε Behavioral Simulation



Εικόνα 47 – Αποτέλεσμα εντολής ROR σε Synthesis Simulation

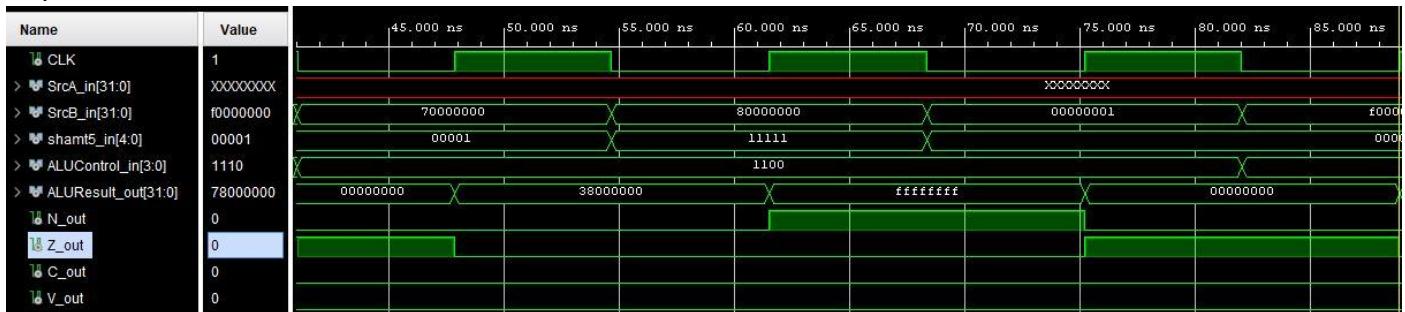


Εικόνα 48 – Αποτέλεσμα εντολής ROR σε Implementation Simulation

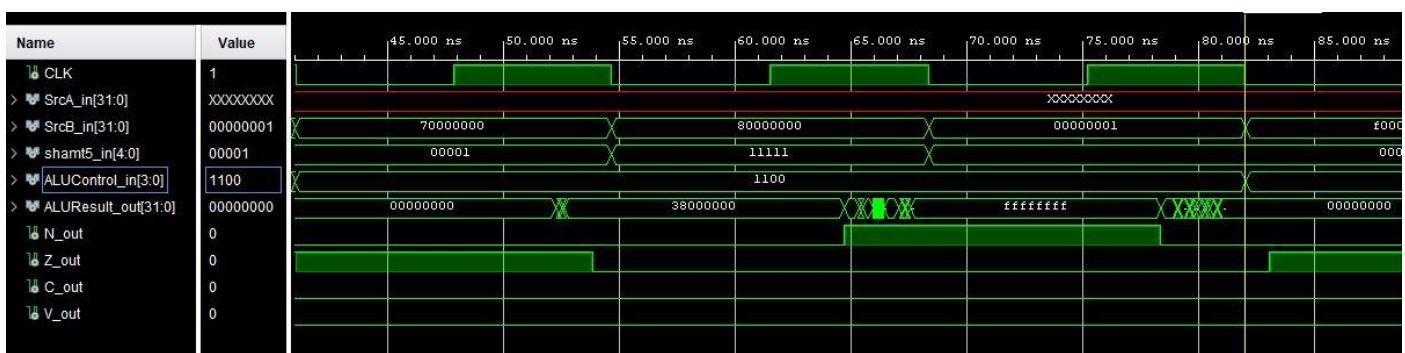
## 2.2.2 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ ASR

Η εντολή ASR αντιστοιχεί σε τιμή σήματος ALUControl = "1100" και εκτελεί αριθμητική ολίσθηση προς τα δεξιά (Arithmetic Shift Right) της τιμής του SrcB κατά αριθμό θέσεων που καθορίζεται από το shamt5. Κατά τη διάρκεια της ολίσθησης, το πιο σημαντικό bit (MSB) διατηρείται ώστε να διαφυλάσσεται το πρόσημο του αριθμού. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη περίπτωση δεν αναμένεται να ενεργοποιηθεί καμία σημαία, στη δεύτερη ενεργοποιείται η σημαία N, καθώς το αποτέλεσμα είναι αρνητικό, ενώ στην τρίτη ενεργοποιείται η σημαία Z, καθώς το αποτέλεσμα της πράξης είναι μηδενικό.

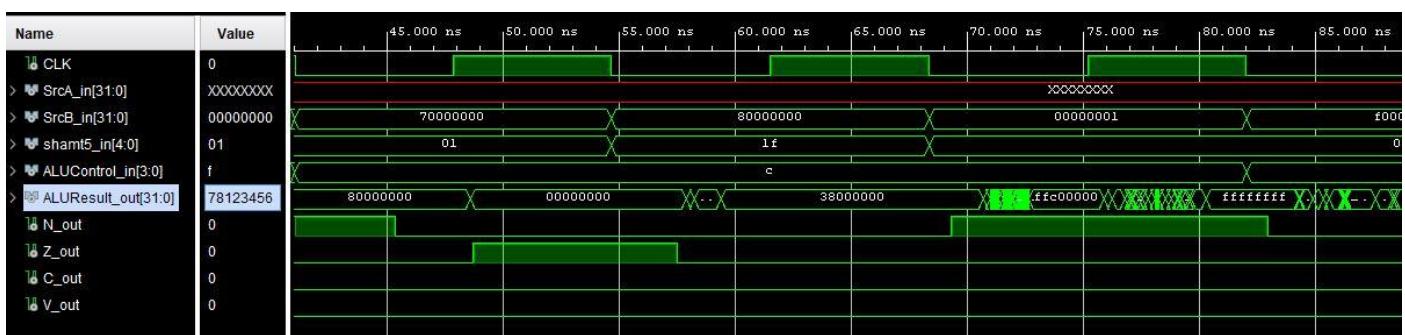
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



Εικόνα 49 – Αποτέλεσμα εντολής ASR σε Behavioral Simulation



Εικόνα 50 – Αποτέλεσμα εντολής ASR σε Synthesis Simulation

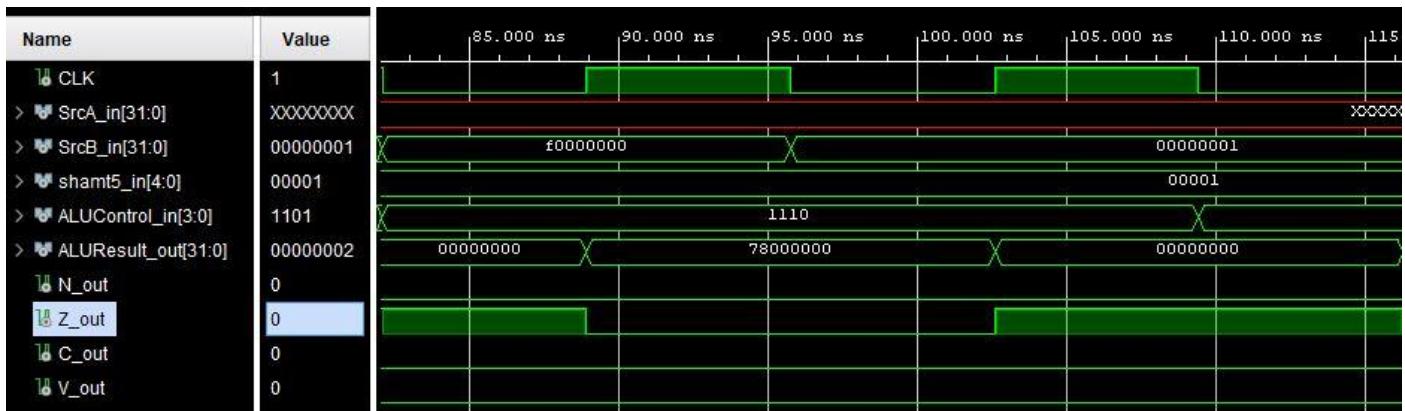


Εικόνα 51 – Αποτέλεσμα εντολής ASR σε Implementation Simulation

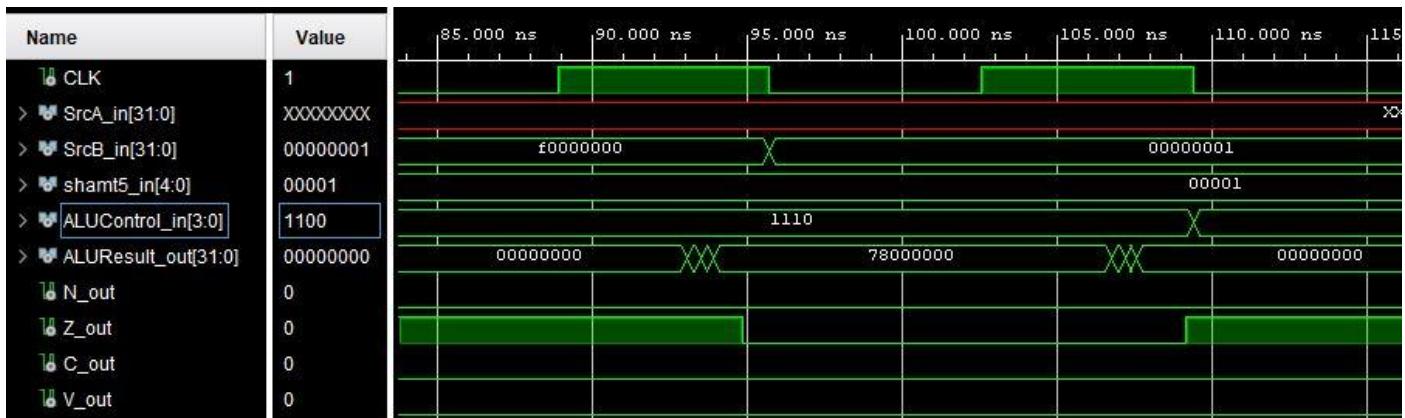
## 2.2.3 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ LSR

Η εντολή LSR αντιστοιχεί σε τιμή σήματος ALUControl = "1110" και εκτελεί λογική ολίσθηση προς τα δεξιά (Logical Shift Right) της τιμής του SrcB κατά αριθμό θέσεων που καθορίζεται από το shamt5. Κατά την ολίσθηση, τα κενά που δημιουργούνται στα αριστερά συμπληρώνονται με μηδενικά. Στο testbench πραγματοποιούνται δύο δοκιμές. Στην πρώτη περίπτωση δεν αναμένεται να ενεργοποιηθεί καμία σημαία, ενώ στη δεύτερη ενεργοποιείται η σημαία Z, καθώς το αποτέλεσμα της πράξης είναι μηδενικό.

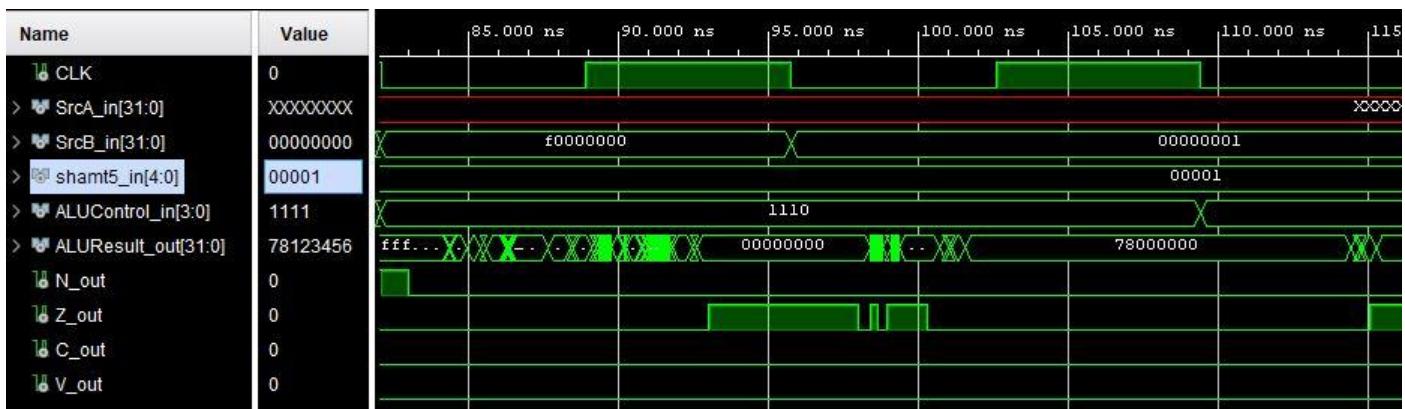
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 52 – Αποτέλεσμα εντολής LSR σε Behavioral Simulation**



**Εικόνα 53 – Αποτέλεσμα εντολής LSR σε Synthesis Simulation**

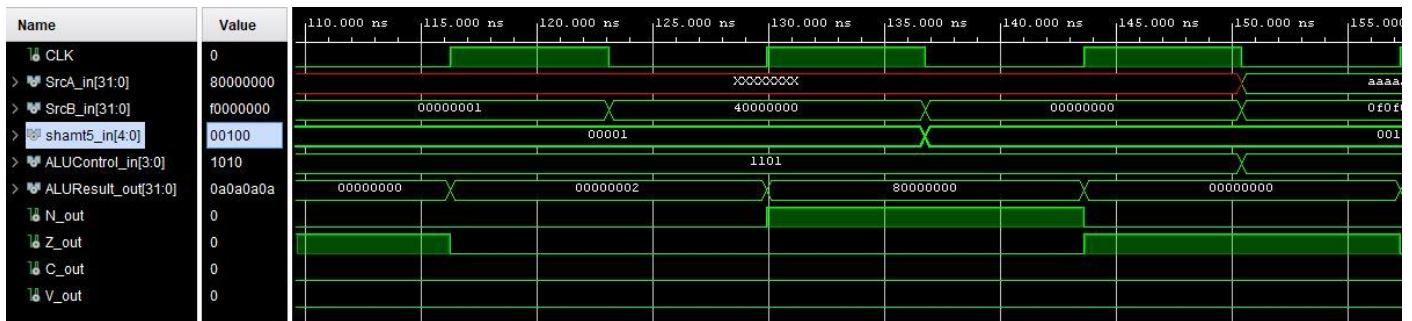


**Εικόνα 54 – Αποτέλεσμα εντολής LSR σε Implementation Simulation**

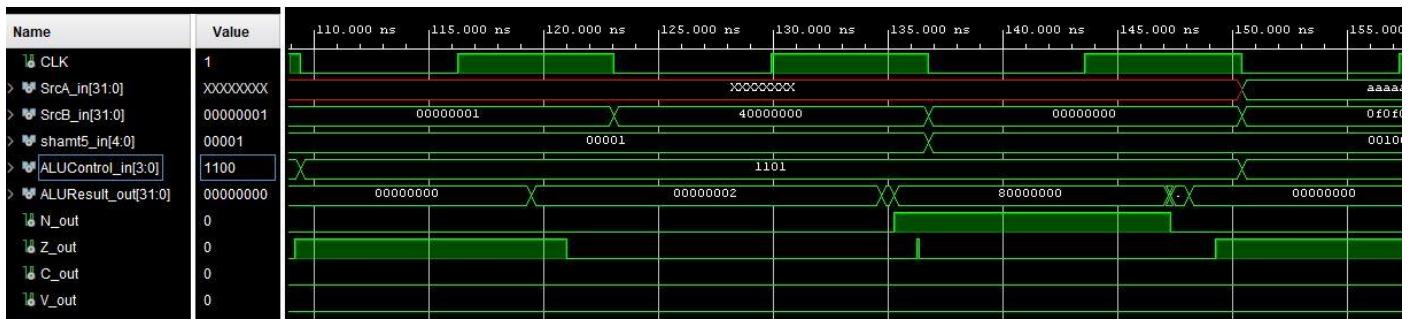
## 2.2.4 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ LSL

Η εντολή LSL αντιστοιχεί σε τιμή σήματος ALUControl = "1101" και εκτελεί λογική ολίσθηση προς τα αριστερά (Logical Shift Left) της τιμής του SrcB κατά αριθμό θέσεων που καθορίζεται από το shamt5. Κατά την ολίσθηση, τα λιγότερο σημαντικά bits (LSBs) συμπληρώνονται με μηδενικά. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη περίπτωση δεν αναμένεται να ενεργοποιηθεί καμία σημαία, στη δεύτερη ενεργοποιείται η σημαία N, καθώς το αποτέλεσμα είναι αρνητικό (MSB = 1), ενώ στην τρίτη ενεργοποιείται η σημαία Z, καθώς το αποτέλεσμα της πράξης είναι μηδενικό.

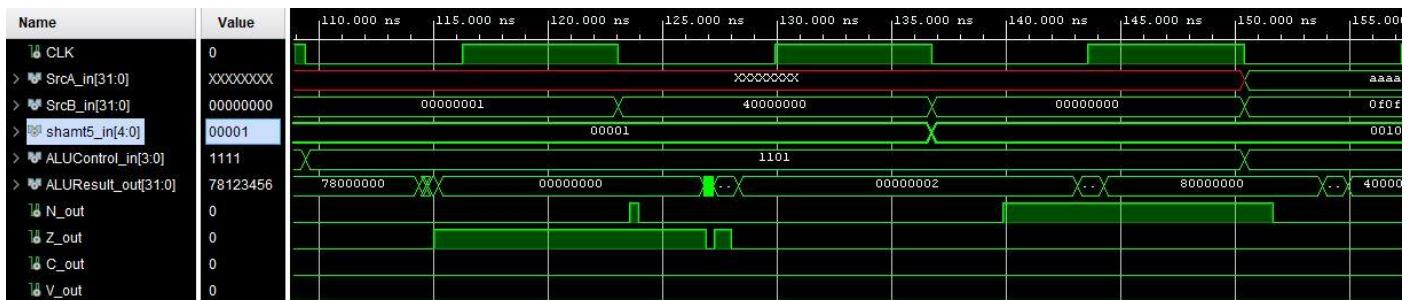
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 55 – Αποτέλεσμα εντολής LSL σε Behavioral Simulation**



**Εικόνα 56 – Αποτέλεσμα εντολής LSL σε Synthesis Simulation**

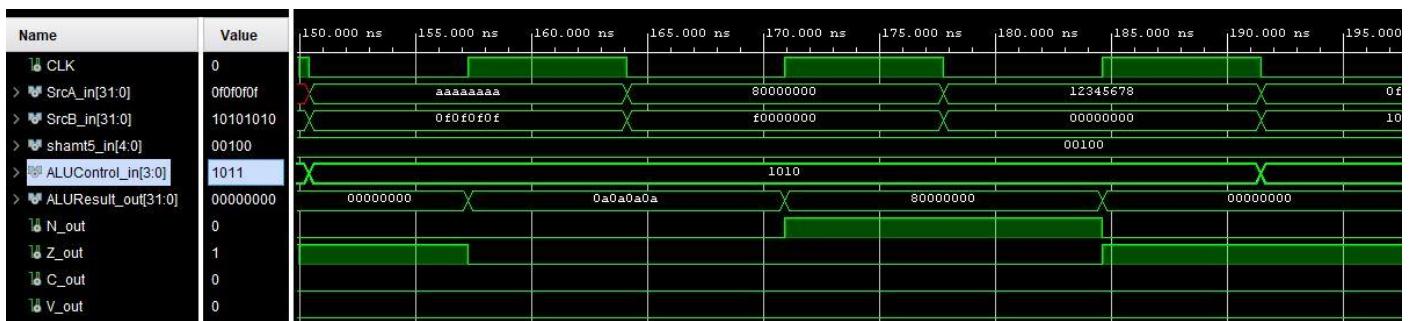


**Εικόνα 57 – Αποτέλεσμα εντολής LSL σε Implementation Simulation**

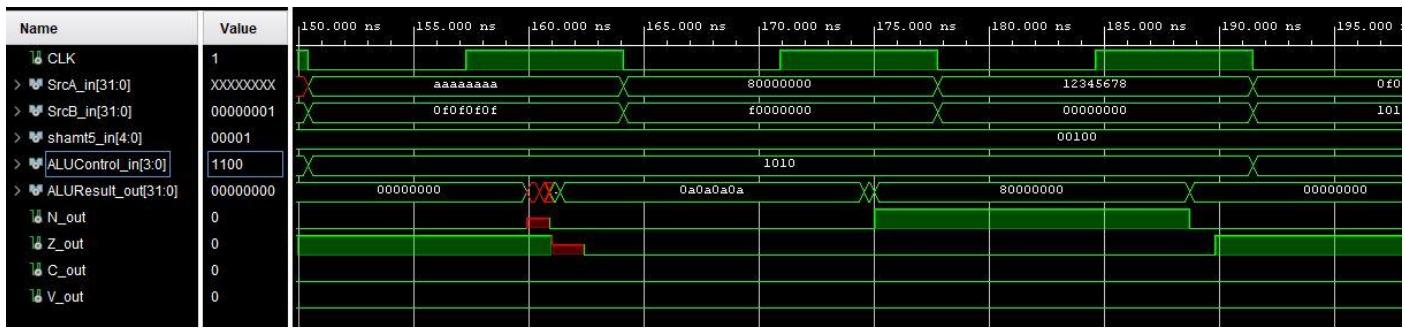
## 2.2.5 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ AND

Η εντολή AND αντιστοιχεί σε τιμή σήματος ALUControl = "1010" και εκτελεί λογική πράξη AND μεταξύ των τιμών των σημάτων SrcA και SrcB, συγκρίνοντας τα bits ένα προς ένα. Το αποτέλεσμα περιέχει '1' μόνο στις θέσεις όπου και τα δύο αντίστοιχα bits των εισόδων είναι '1'. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη περίπτωση δεν αναμένεται να ενεργοποιηθεί καμία σημαία, στη δεύτερη ενεργοποιείται η σημαία N, καθώς το αποτέλεσμα είναι αρνητικό (MSB = 1), ενώ στην τρίτη ενεργοποιείται η σημαία Z, καθώς το αποτέλεσμα είναι μηδενικό.

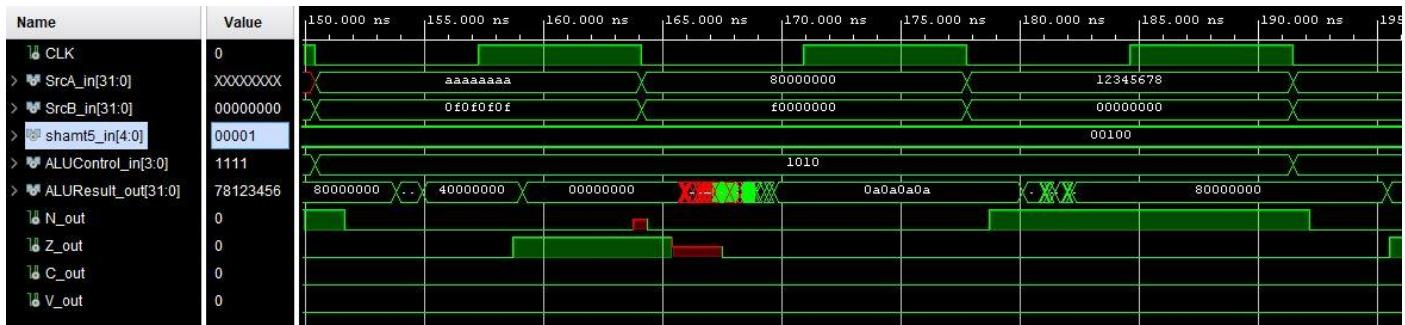
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 58 – Αποτέλεσμα εντολής AND σε Behavioral Simulation**



**Εικόνα 59 – Αποτέλεσμα εντολής AND σε Synthesis Simulation**

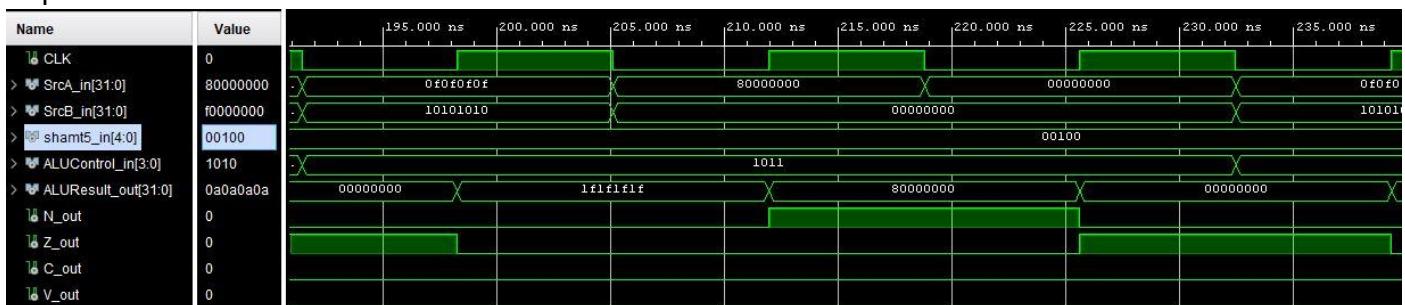


**Εικόνα 60 – Αποτέλεσμα εντολής AND σε Implementation Simulation**

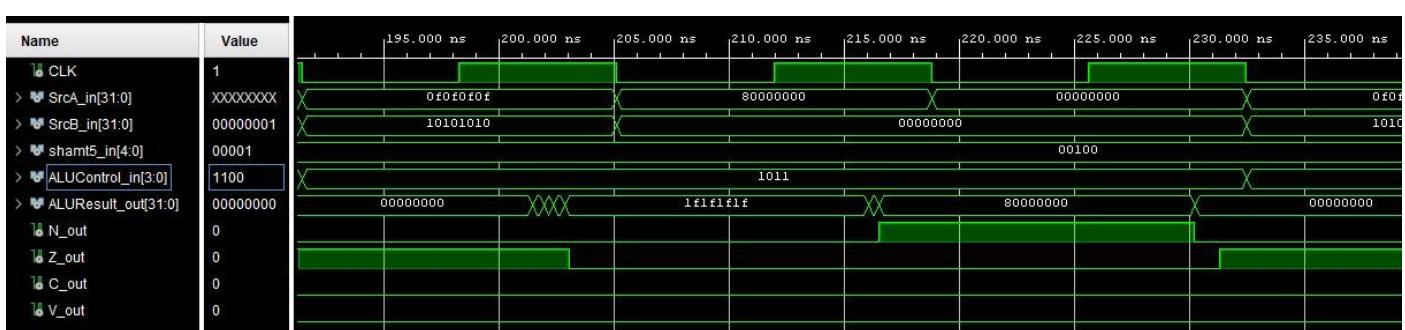
## 2.2.6 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ OR

Η εντολή OR αντιστοιχεί σε τιμή σήματος ALUControl = "1011" και εκτελεί λογική πράξη OR μεταξύ των εισόδων SrcA και SrcB, συγκρίνοντας τα bits ένα προς ένα. Το αποτέλεσμα περιέχει '1' σε κάθε θέση όπου τουλάχιστον ένα από τα δύο αντίστοιχα bits των εισόδων είναι '1'. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη περίπτωση δεν ενεργοποιείται καμία σημαία, στη δεύτερη ενεργοποιείται η σημαία N, καθώς το αποτέλεσμα είναι αρνητικό (MSB = 1), ενώ στην τρίτη ενεργοποιείται η σημαία Z, καθώς το αποτέλεσμα είναι μηδενικό.

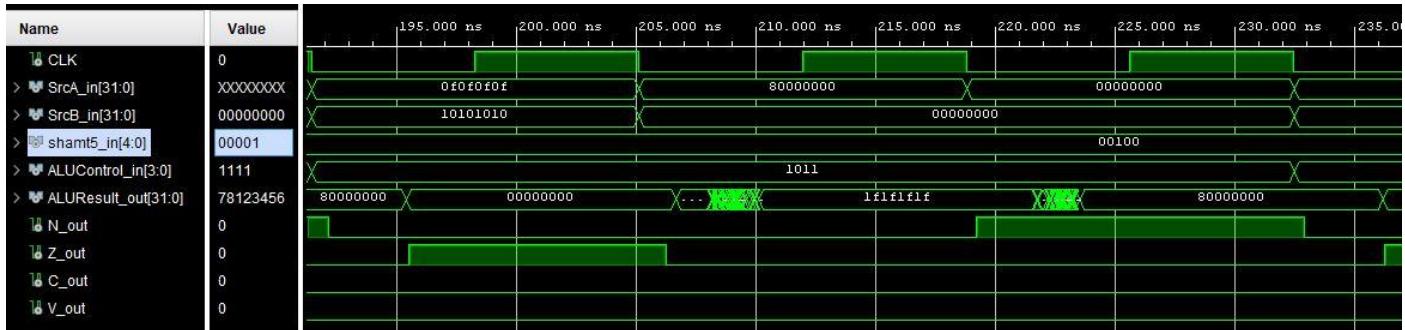
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 61 – Αποτέλεσμα εντολής OR σε Behavioral Simulation**



**Εικόνα 62 – Αποτέλεσμα εντολής OR σε Synthesis Simulation**

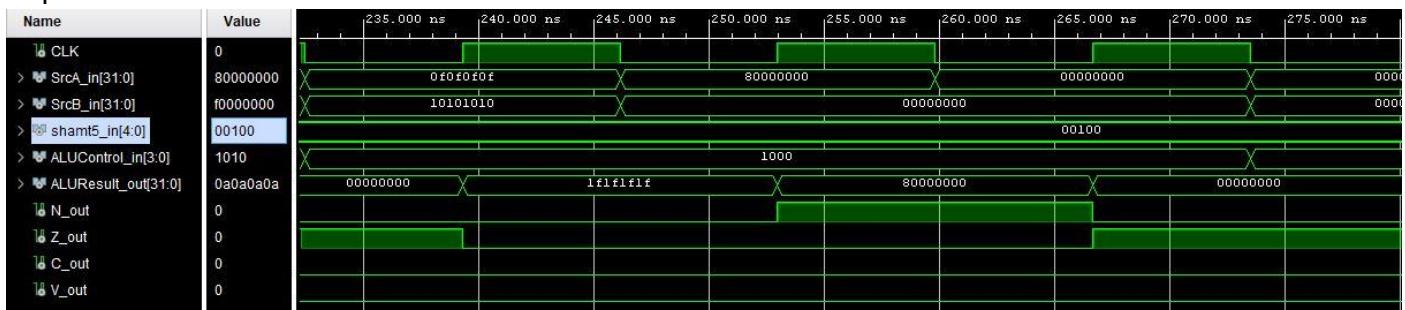


**Εικόνα 63 – Αποτέλεσμα εντολής OR σε Implementation Simulation**

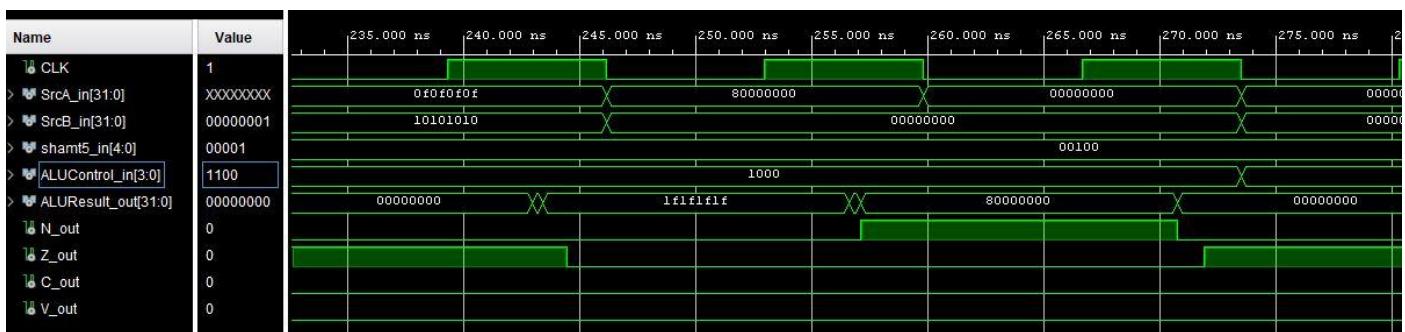
### 2.2.7 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ XOR

Η εντολή XOR αντιστοιχεί σε τιμή σήματος ALUControl = "1000" και εκτελεί λογική πράξη XOR (αποκλειστικό OR) μεταξύ των εισόδων SrcA και SrcB. Το αποτέλεσμα περιέχει '1' μόνο στις θέσεις όπου τα αντίστοιχα bits των δύο εισόδων διαφέρουν. Στο testbench εκτελούνται τρεις δοκιμές. Στην πρώτη περίπτωση δεν ενεργοποιείται καμία σημαία, στη δεύτερη ενεργοποιείται η σημαία N, καθώς το αποτέλεσμα είναι αρνητικό (MSB = 1), ενώ στην τρίτη ενεργοποιείται η σημαία Z, καθώς το αποτέλεσμα είναι μηδενικό.

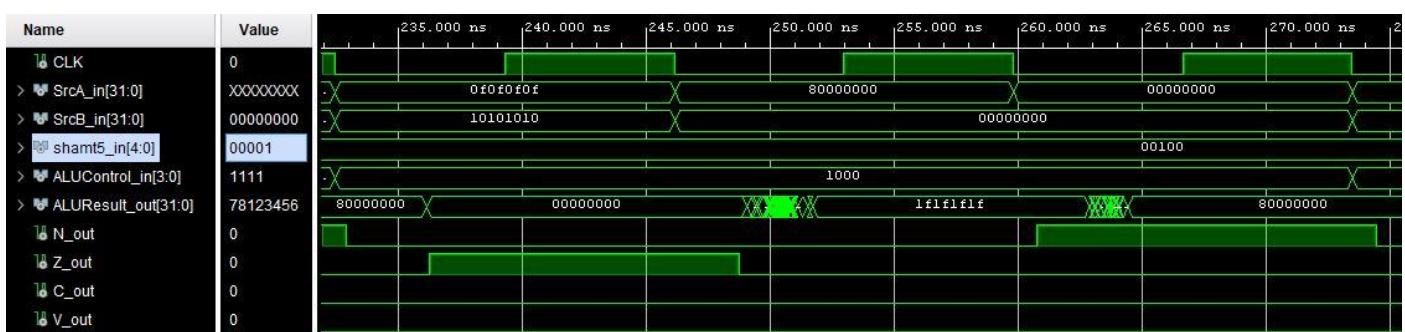
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 64 – Αποτέλεσμα εντολής XOR σε Behavioral Simulation**



**Εικόνα 65 – Αποτέλεσμα εντολής XOR σε Synthesis Simulation**

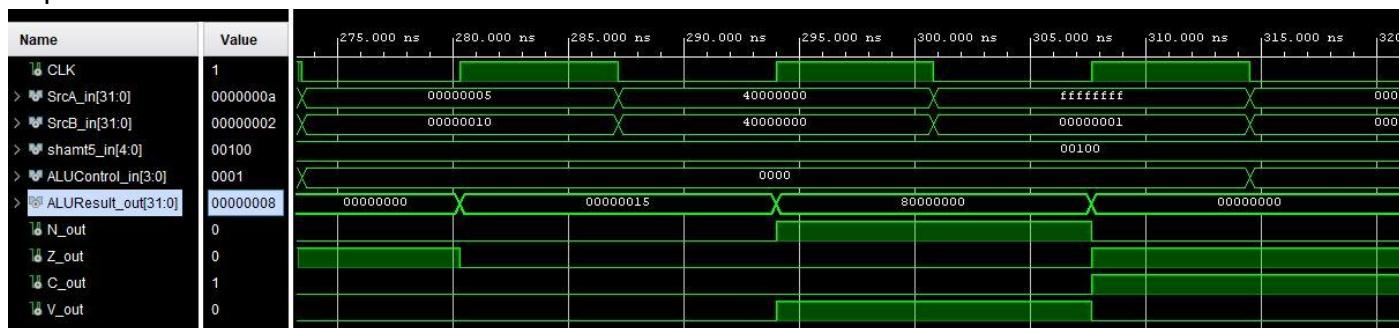


**Εικόνα 66 – Αποτέλεσμα εντολής XOR σε Implementation Simulation**

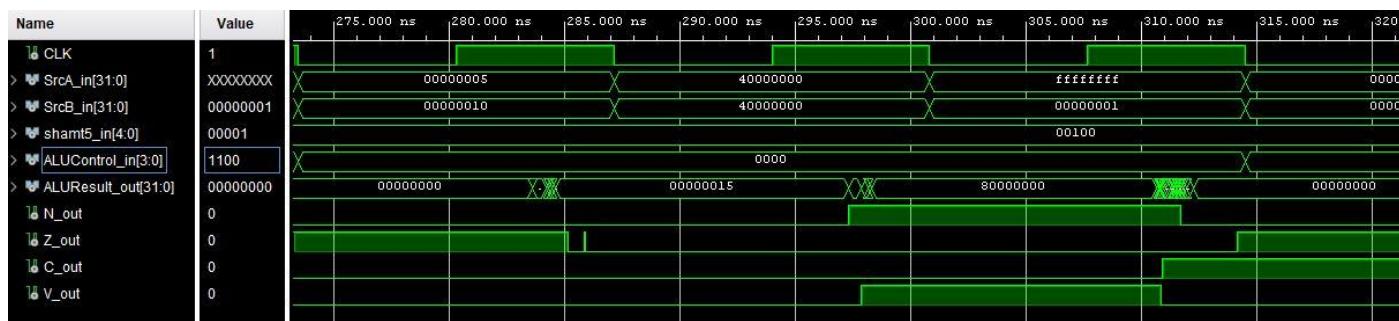
## 2.2.8 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ ADD

Η εντολή ADD αντιστοιχεί σε τιμή σήματος ALUControl = "0000" και εκτελεί πράξη πρόσθεσης μεταξύ των εισόδων SrcA και SrcB. Κατά την εκτέλεση ενημερώνονται οι σημαίες κατάστασης ανάλογα με το αποτέλεσμα. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη περίπτωση δεν ενεργοποιείται καμία σημαία, στη δεύτερη ενεργοποιούνται οι σημαίες N και V, καθώς το αποτέλεσμα είναι αρνητικό και προκύπτει υπερχείλιση (overflow), ενώ στην τρίτη ενεργοποιείται η σημαία C, καθώς υπάρχει μεταφορά (carry) και η σημαία Z, αφού το αποτέλεσμα είναι μηδέν.

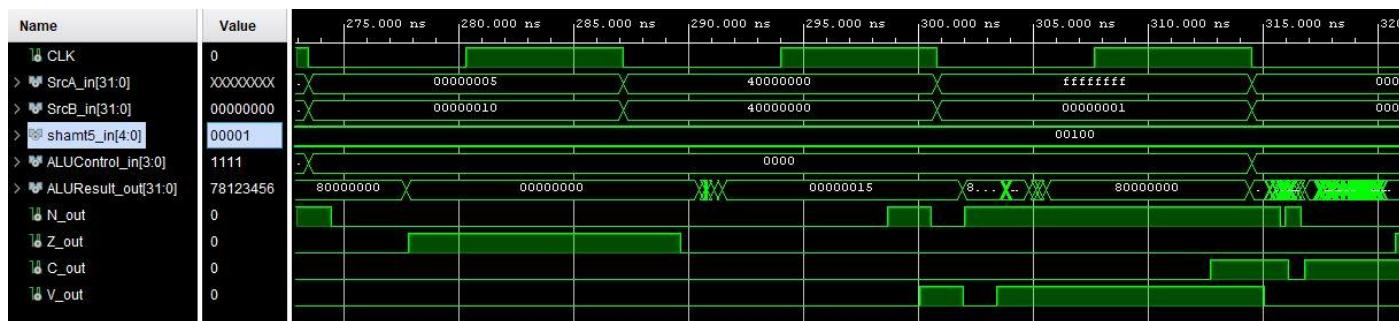
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



Εικόνα 67 – Αποτέλεσμα εντολής ADD σε Behavioral Simulation



Εικόνα 68 – Αποτέλεσμα εντολής ADD σε Synthesis Simulation

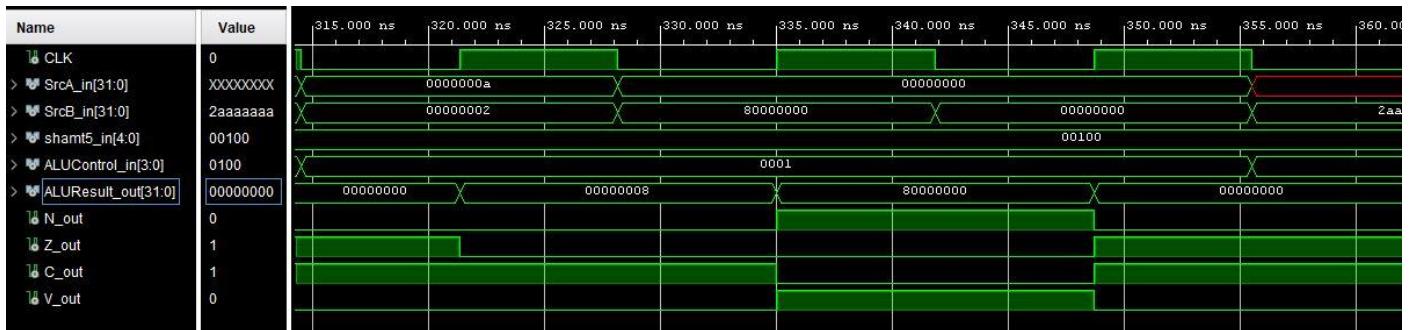


Εικόνα 69 – Αποτέλεσμα εντολής ADD σε Implementation Simulation

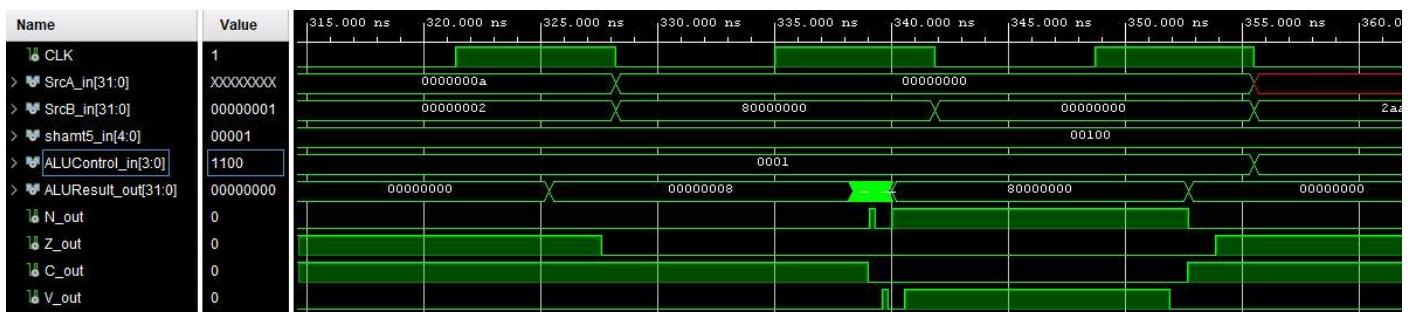
## 2.2.9 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ SUB

Η εντολή SUB αντιστοιχεί σε τιμή σήματος ALUControl = "0001" και εκτελεί πράξη αφαίρεσης της εισόδου SrcB από την είσοδο SrcA. Κατά την εκτέλεση ενημερώνονται οι σημαίες κατάστασης ανάλογα με το αποτέλεσμα. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη περίπτωση δεν ενεργοποιείται η σημαία C, καθώς δεν προκύπτει δανεισμός (borrow), στη δεύτερη ενεργοποιούνται οι σημαίες N και V, αφού το αποτέλεσμα είναι αρνητικό και προκαλεί υπερχείλιση, ενώ στην τρίτη ενεργοποιείται η σημαία Z, επειδή το αποτέλεσμα είναι μηδενικό.

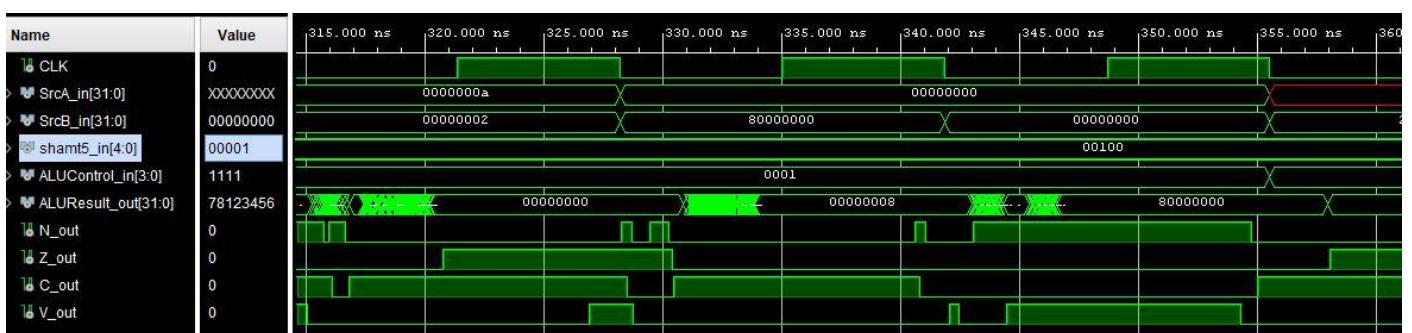
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 70 – Αποτέλεσμα εντολής SUB σε Behavioral Simulation**



**Εικόνα 71 – Αποτέλεσμα εντολής SUB σε Synthesis Simulation**

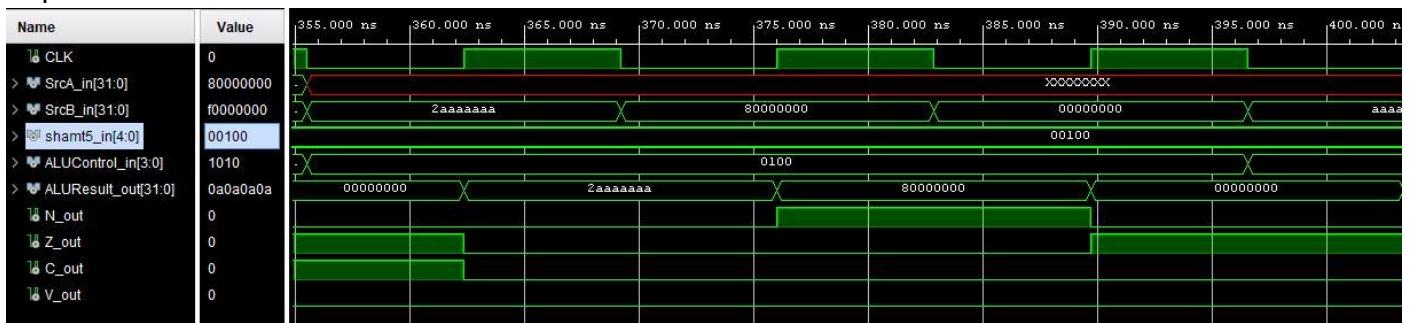


**Εικόνα 72 – Αποτέλεσμα εντολής SUB σε Implementation Simulation**

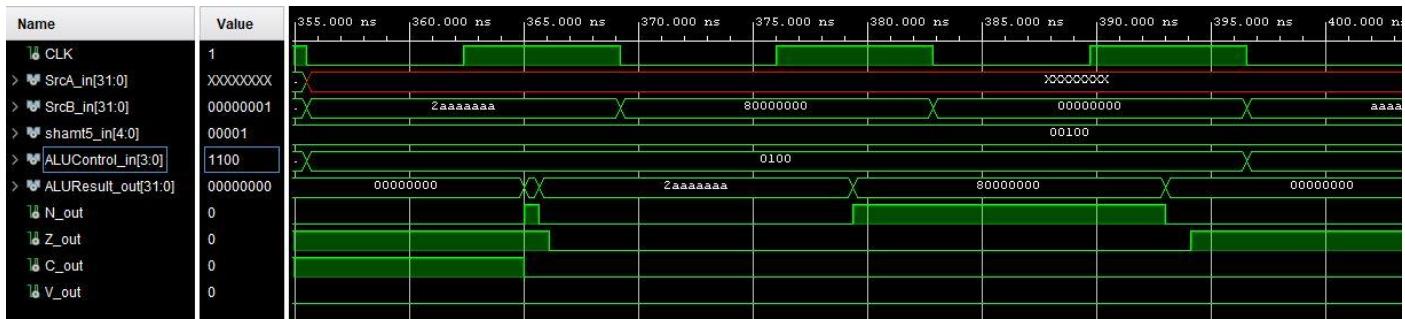
## 2.2.10 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ MOV

Η εντολή MOV αντιστοιχεί σε τιμή σήματος ALUControl = "0100" και αντιγράφει απευθείας την τιμή της εισόδου SrcB στο αποτέλεσμα ALUResult, χωρίς να πραγματοποιεί κάποια αριθμητική ή λογική πράξη. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη δεν ενεργοποιείται καμία σημαία, στη δεύτερη ενεργοποιείται η σημαία N, καθώς το αποτέλεσμα είναι αρνητικό (το πιο σημαντικό bit είναι 1), ενώ στην τρίτη ενεργοποιείται η σημαία Z, επειδή το αποτέλεσμα είναι μηδενικό.

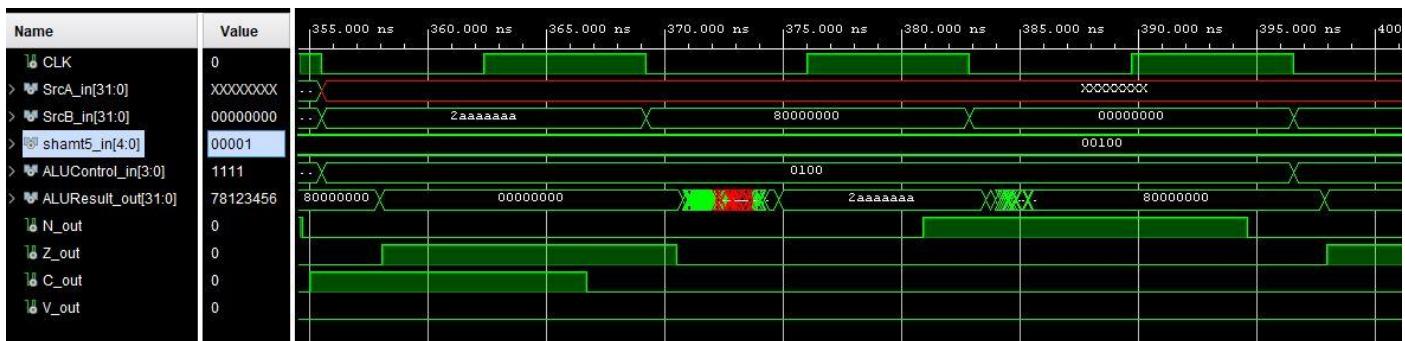
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 73 – Αποτέλεσμα εντολής MOV σε Behavioral Simulation**



Εικόνα 74 – Αποτέλεσμα εντολής MOV σε Synthesis Simulation

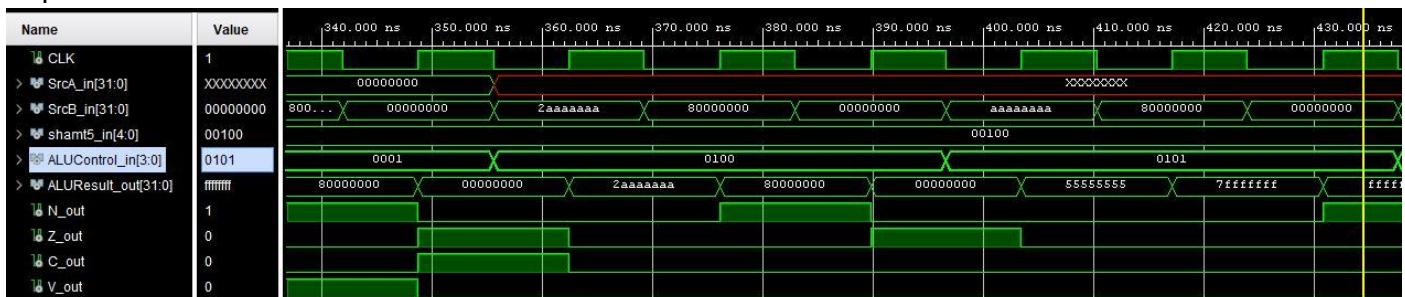


Εικόνα 75 – Αποτέλεσμα εντολής MOV σε Implementation Simulation

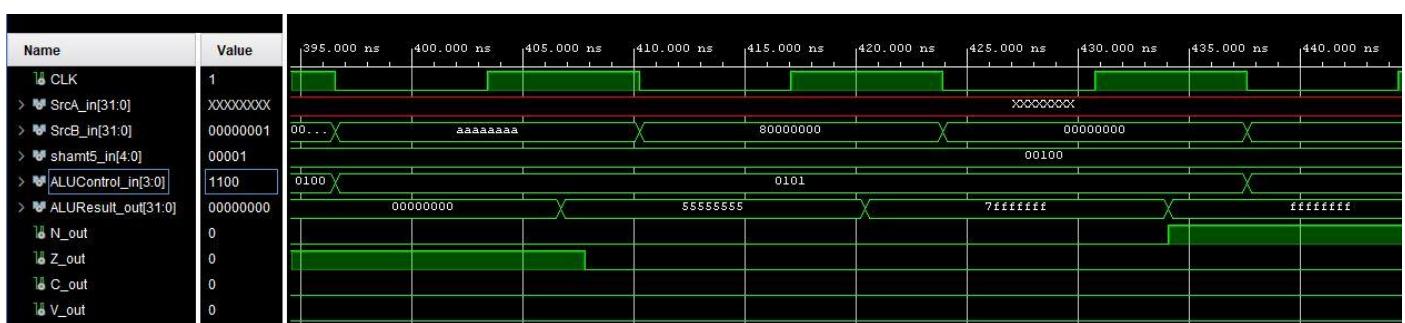
### 2.2.11 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ MVN

Η εντολή MVN αντιστοιχεί σε τιμή σήματος ALUControl = "0101" και εκτελεί bitwise αντιστροφή (NOT) της εισόδου SrcB, αντιστρέφοντας όλα τα bits της. Στο testbench πραγματοποιούνται τρεις δοκιμές. Στην πρώτη δεν ενεργοποιείται καμία σημαία, καθώς το αποτέλεσμα είναι θετικό και διαφορετικό του μηδενός, στη δεύτερη καμία σημαία επίσης δεν ενεργοποιείται, αφού το αποτέλεσμα παραμένει θετικό, ενώ στην τρίτη ενεργοποιείται η σημαία N, καθώς το αποτέλεσμα είναι αρνητικό (όλα τα bits ίσα με 1).

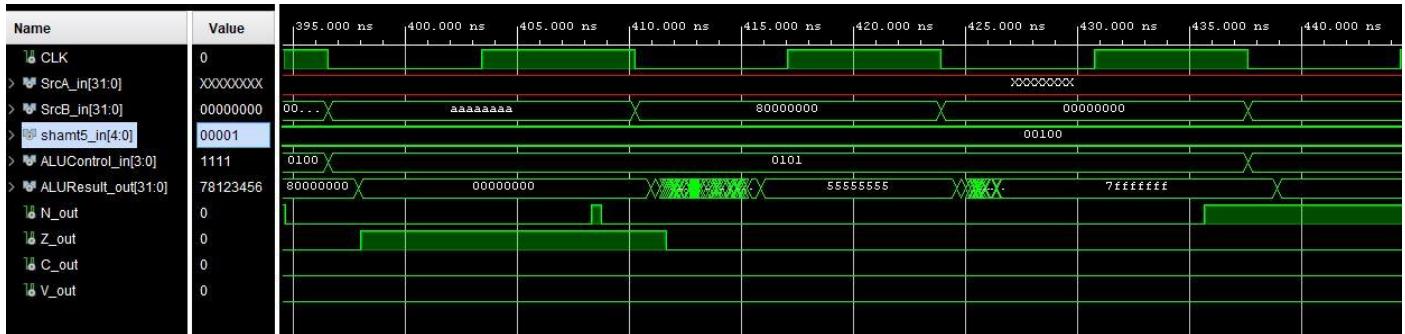
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



Εικόνα 76 – Αποτέλεσμα εντολής MVN σε Behavioral Simulation



Εικόνα 77 – Αποτέλεσμα εντολής MVN σε Synthesis Simulation

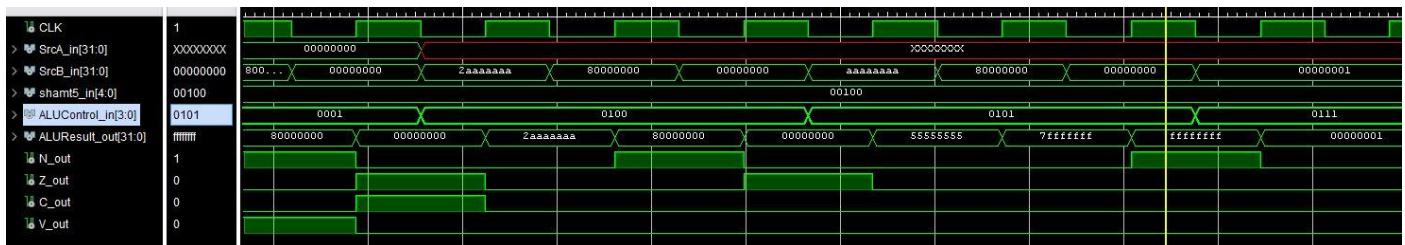


**Εικόνα 78 – Αποτέλεσμα εντολής MVN σε Implementation Simulation**

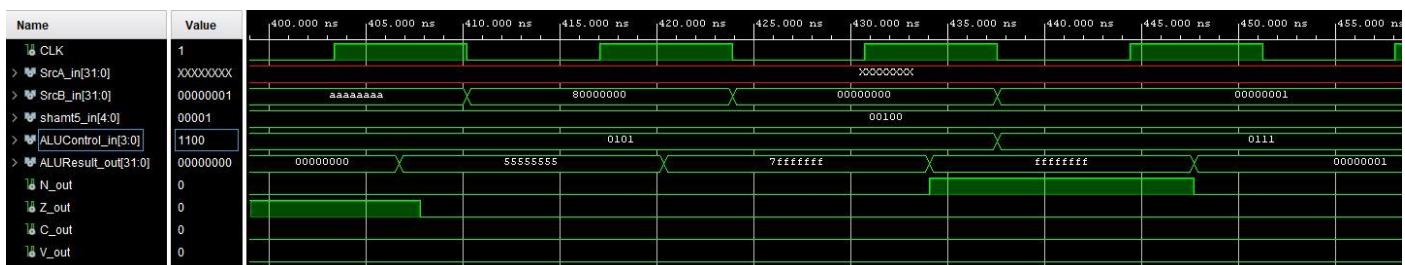
### 2.2.12 ΔΟΚΙΜΗ ΕΝΤΟΛΗΣ NOP

Η εντολή NOP αντιστοιχεί σε τιμή σήματος ALUControl = "0111" και δεν εκτελεί καμία πράξη, διατηρώντας αμετάβλητες τις εισόδους και τις σημαίες. Στο testbench πραγματοποιείται μία δοκιμή, κατά την οποία το αποτέλεσμα παραμένει ίδιο με την είσοδο SrcB, χωρίς να ενεργοποιηθεί καμία σημαία, επιβεβαιώνοντας ότι η εντολή δεν επηρεάζει τη λειτουργία της ALU.

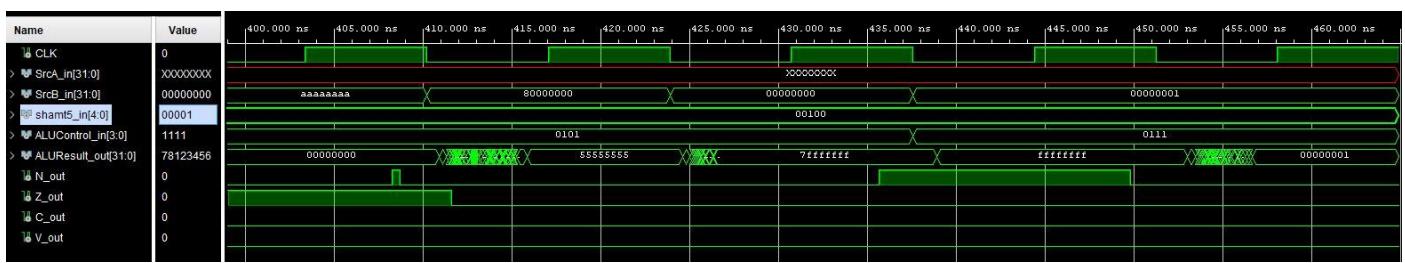
Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation, Synthesis Simulation και Implementation Simulation.



**Εικόνα 79 – Αποτέλεσμα εντολής NOP σε Behavioral Simulation**



**Εικόνα 80 – Αποτέλεσμα εντολής NOP σε Synthesis Simulation**



**Εικόνα 81 – Αποτέλεσμα εντολής NOP σε Implementation Simulation**

### 2.2.13 ΣΧΟΛΙΑΣΜΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ ΧΡΟΝΙΣΜΟΥ ALU

Κατά την ανάλυση των αποτελεσμάτων των προσομοιώσεων της μονάδας ALU, διαπιστώνονται σημαντικές διαφορές ανάμεσα στις φάσεις Behavioral, Synthesis και Implementation, οι οποίες σχετίζονται με το επίπεδο αφαίρεσης και τη φυσική προσέγγιση του κυκλώματος που προσομοιώνεται.

Στην Behavioral προσομοίωση, το κύκλωμα προσομοιώνεται αποκλειστικά βάσει του κώδικα VHDL, χωρίς να λαμβάνονται υπόψη οι φυσικές καθυστερήσεις των πυλών ή των καλωδιώσεων. Το μοντέλο είναι ιδανικό και η εκτέλεση των πράξεων της ALU (όπως πρόσθεση, αφαίρεση, λογικές πράξεις και ολισθήσεις) πραγματοποιείται στιγμιαία. Ως αποτέλεσμα, τα διαγράμματα χρονισμού εμφανίζονται χωρίς μεταβατικές καταστάσεις ή χρονικές αποκλίσεις μεταξύ εισόδων και εξόδων. Η προσομοίωση αυτή χρησιμοποιείται κυρίως για τη λογική επαλήθευση της ορθότητας του κώδικα και όχι για αξιολόγηση καθυστερήσεων.

Στη φάση της Synthesis, ο VHDL κώδικας μετατρέπεται σε ένα λογικό ισοδύναμο κύκλωμα αποτελούμενο από πύλες και flip-flops. Η προσομοίωση πλέον πραγματοποιείται πάνω στο συνθεμένο netlist, το οποίο αρχίζει να ενσωματώνει ρεαλιστικά χρονικά χαρακτηριστικά, όπως καθυστερήσεις διάδοσης των σημάτων μέσα από τη λογική. Έτσι, στα διαγράμματα χρονισμού παρατηρούνται μικρές αποκλίσεις μεταξύ των χρονικών στιγμών αλλαγής των σημάτων, ενώ η μορφή των κυματομορφών γίνεται λιγότερο “ομαλή” σε σχέση με την behavioral προσομοίωση. Οι καθυστερήσεις είναι μεν περιορισμένες, αλλά αντανακλούν το γεγονός ότι η υλοποίηση του κυκλώματος δεν είναι πλέον ιδανική.

Τέλος, στην Implementation έχει γίνει πραγματική τοποθέτηση και δρομολόγηση των πόρων του FPGA. Σε αυτό το στάδιο λαμβάνονται υπόψη όλες οι φυσικές καθυστερήσεις διάδοσης. Κατά συνέπεια, τα διαγράμματα χρονισμού παρουσιάζουν μεγάλες διακυμάνσεις, χρονικές καθυστερήσεις και μεταβατικές τιμές (glitches) στις εξόδους, πριν σταθεροποιηθούν. Αυτές οι διακυμάνσεις είναι απολύτως φυσιολογικές και εκφράζουν την πραγματική ηλεκτρονική συμπεριφορά της ALU μέσα στο FPGA, όπως θα λειτουργούσε σε υλικό επίπεδο.

Για παράδειγμα στην πρώτη εντολή ROR βλέπουμε στην Behavioral προσομοίωση ότι η έξοδος ALUResult έχει σταθεροποιηθεί στα 7ns και στην Synthesis στα 12 ns. Στην Implementation προσομοίωση μέχρι να πάρει η έξοδο της σωστή τιμή βλέπουμε να συμβαίνουν μεγάλες διακύμανσης και εν τέλη η τιμή να σταθεροποιείται στα 26ns για μόλις 3ns. Υπογραμμίζεται ότι ο κύκλος του ρολογιού είναι 12.150ns.

Το ίδιο γίνεται αρκετά έντονα και στην δεύτερη εντολή ASR. Εδώ έχουμε στην Behavioral προσομοίωση την έξοδο ALUResult να σταθεροποιείται στα 61ns και στην Synthesis στα 63 ns. Στην Implementation προσομοίωση μέχρι να πάρει η έξοδο της σωστή τιμή βλέπουμε να συμβαίνουν μεγάλες διακύμανσης και εν τέλη η τιμή να σταθεροποιείται στα 80ns για μόλις 4ns και αφού μάλιστα η τιμή είχε μείνει σχετικά σταθερή σε λάθος αποτέλεσμα μεταξύ των 73-76 ns.

Το ίδιο συμβαίνει και στις επόμενες εντολές, με τις εντολές ολίσθησής πάντως να έχουν τις μεγαλύτερες αποκλίσεις μεταξύ των τριών διαγραμμάτων. Αυτό ίσως συμβαίνει επειδή οι συγκεκριμένες εντολές για να υλοποιηθούν απαιτούν πολλά επίπεδα από πολυπλέκτες και κάθε bit της εξόδου εξαρτάται από πολλαπλά bits της εισόδου μέσω πολυπλεκτών πολλαπλών επιπέδων. Επίσης παρατηρούμε ότι σε κάποιες περιπτώσεις η σταθεροποίηση μιας τιμής παίρνει περισσότερο χρόνο από τον κύκλο του ρολογιού. Όταν θα γίνει η προσομοίωση όλου του επεξεργαστή ή ALU σαν κομμάτι αυτού θα υλοποιηθεί με διαφορετικό τρόπο και έτσι δεν θα παρουσιάζει τις ίδιες καθυστερήσεις, καθώς η τοποθέτηση και η δρομολόγηση των στοιχείων της θα γίνει σε συνάρτηση με το υπόλοιπο κύκλωμα του.

## 2.3 ΔΙΑΓΡΑΜΜΑΤΑ ΧΡΟΝΙΣΜΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

Για τη δοκιμή της ορθής λειτουργίας του επεξεργαστή, αναπτύχθηκε το πρόγραμμα που παρουσιάζεται στην Ενότητα 3.1. Στις επόμενες παραγράφους πραγματοποιείται ανάλυση και σχολιασμός των αποτελεσμάτων που προέκυψαν από τις προσομοιώσεις Behavioral και Implementation.

Σε αντίθεση με τη δοκιμή της μονάδας ALU, όπου για λόγους πληρότητας είχε συμπεριληφθεί και η προσομοίωση Synthesis, στην παρούσα ανάλυση αυτή παραλείπεται. Η επιλογή αυτή για συνοπτικότητα στην παρουσίαση των αποτελεσμάτων. Έτσι και αλλιώς η εκφώνηση της εργασίας ζητά τη μελέτη μόνο των αποτελεσμάτων από τις προσομοιώσεις Behavioral και Implementation.

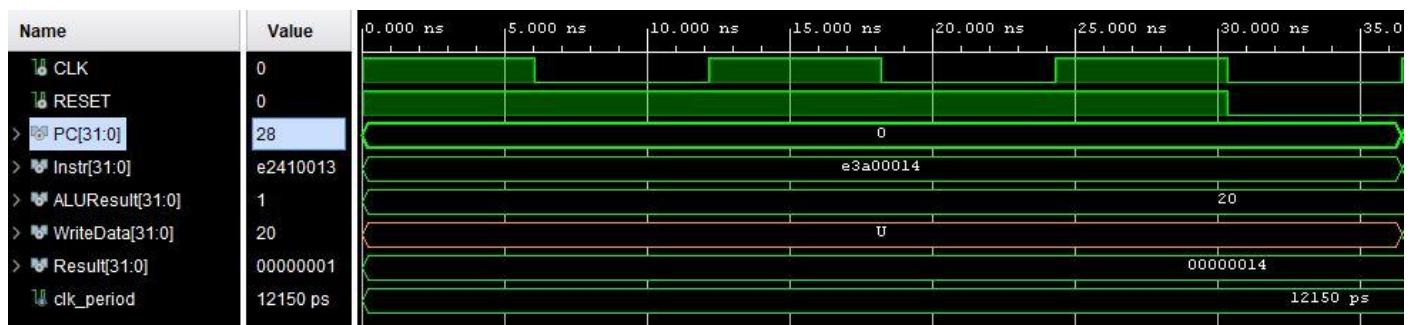
Η διαδικασία της προσομοίωσης ξεκινά με την ενεργοποίηση του RESET. Για διευκόλυνση της παρακολούθησης και επαλήθευσης των αποτελεσμάτων, οι τιμές των σημάτων PC, ALUResult, WriteData και Result παρουσιάζονται σε δεκαδική μορφή.

### 2.3.1 ΕΝΤΟΛΗ MOV R0, 20

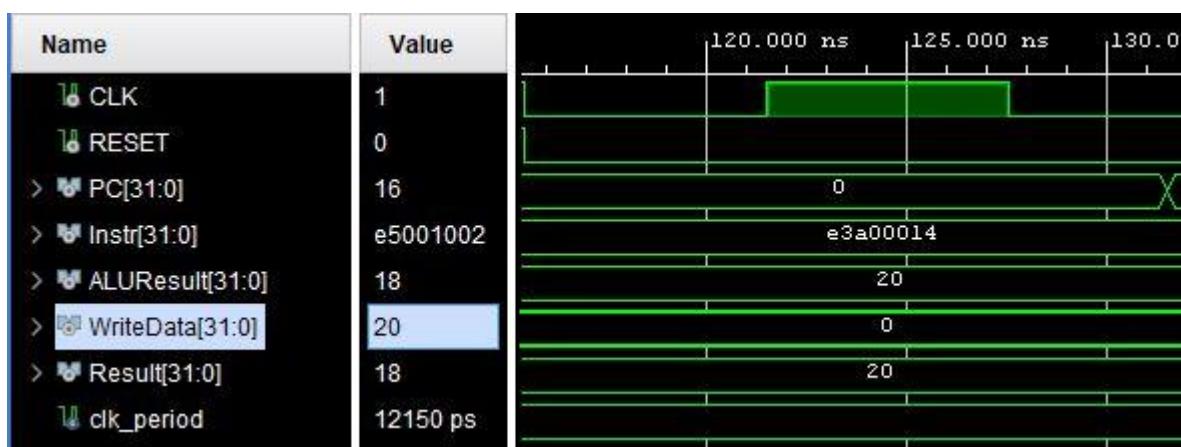
Η εντολή αυτή εκτελεί τη μεταφορά της τιμής 20 στον καταχωρητή R0. Εφόσον πρόκειται για την πρώτη εντολή του προγράμματος, ο μετρητής προγράμματος (PC) έχει αρχική τιμή 0, καθώς δείχνει στην αρχή της μνήμης εντολών. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E3A00014.

Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 20. Η έξοδος WriteData είναι αδιάφορη, καθώς δεν πραγματοποιείται καμία εγγραφή στη μνήμη δεδομένων. Τέλος, η έξοδος Result λαμβάνει επίσης την τιμή 20, καθώς το τελικό αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



Εικόνα 82 – Αποτέλεσμα εντολής MOV R0, 20 σε Behavioral Simulation



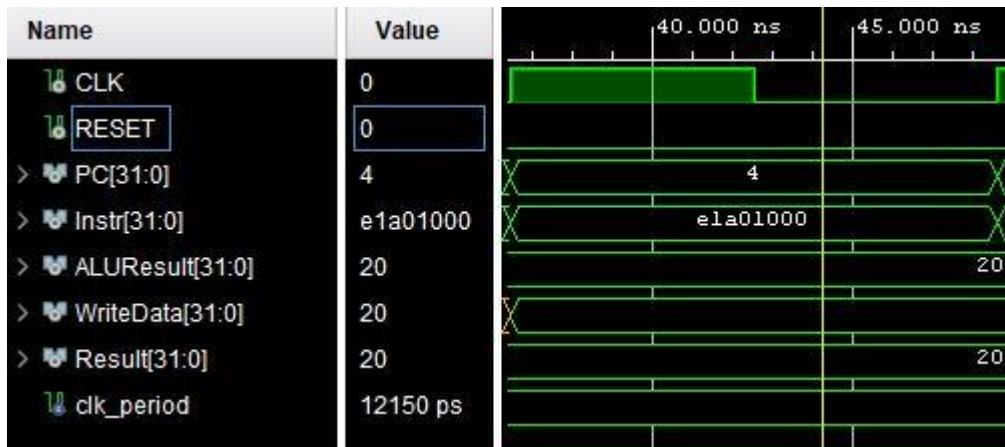
Εικόνα 83 – Αποτέλεσμα εντολής MOV R0, 20 σε Implementation Simulation

### 2.3.2 ΕΝΤΟΛΗ MOV R1, R0

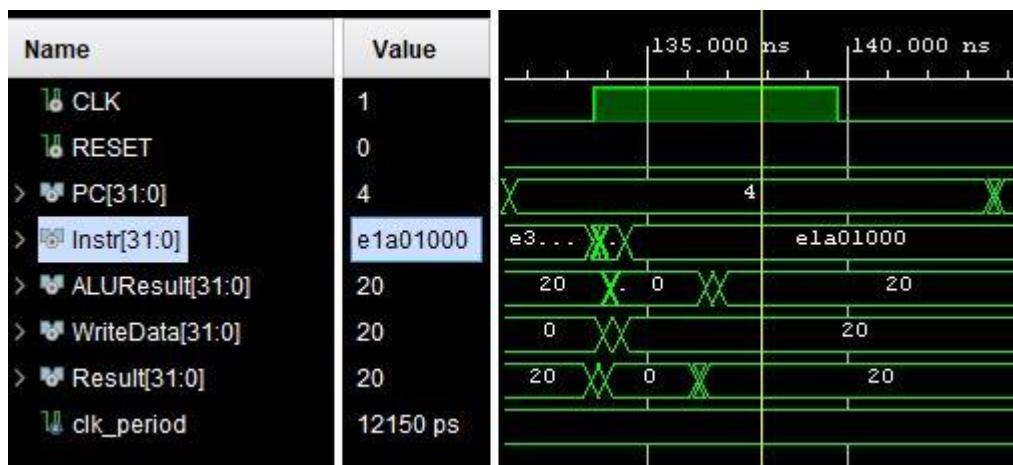
Η εντολή αυτή εκτελεί τη μεταφορά της τιμής του καταχωρητή R0, δηλαδή 20, στον καταχωρητή R1. Ο μετρητής προγράμματος (PC) έχει ήδη προχωρήσει στην επόμενη διεύθυνση (διεύθυνση 4), καθώς η εντολή βρίσκεται στη δεύτερη θέση του προγράμματος. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1A01000.

Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι 20, δηλαδή η τιμή που μεταφέρεται από τον R0 στον R1. Η έξοδος WriteData είναι αδιάφορη, αφού δεν πραγματοποιείται εγγραφή στη μνήμη δεδομένων. Τέλος, η έξοδος Result λαμβάνει επίσης την τιμή 20, καθώς το αποτέλεσμα προέρχεται απευθείας από την ALU και όχι από τη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 84 – Αποτέλεσμα εντολής MOV R1, R0 σε Behavioral Simulation**



**Εικόνα 85 – Αποτέλεσμα εντολής MOV R1, R0 σε Implementation Simulation**

### 2.3.3 ΕΝΤΟΛΗ NOP

Η εντολή NOP είναι μια ουδέτερη εντολή που δεν επιφέρει καμία μεταβολή στην κατάσταση του επεξεργαστή ή στα περιεχόμενα των καταχωρητών. Στο συγκεκριμένο πρόγραμμα βρίσκεται στην τρίτη θέση, με τον μετρητή προγράμματος (PC) να έχει τιμή 8, καθώς προηγούνται δύο εντολές των 4 byte η καθεμία. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1A00000, ο οποίος αντιστοιχεί σε εντολή μεταφοράς δεδομένων χωρίς τροποποίηση των τιμών.

Η NOP δεν επηρεάζει κανένα από τα σήματα ελέγχου, ούτε τις εξόδους της ALU.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 86 – Αποτέλεσμα εντολής NOP σε Behavioral Simulation**

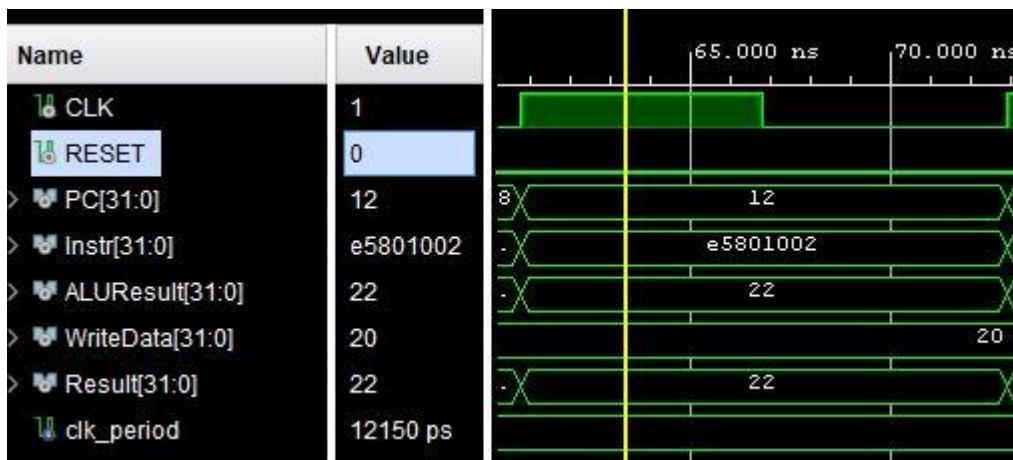


**Εικόνα 87 – Αποτέλεσμα εντολής MOV R1, R0 σε Implementation Simulation**

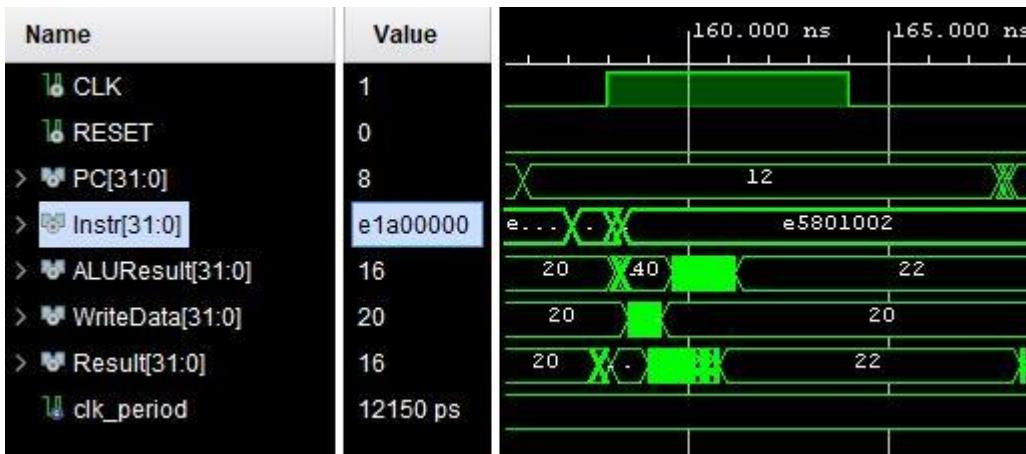
#### 2.3.4 ΕΝΤΟΛΗ STR R1, [R0,2]

Η εντολή STR αποθηκεύει το περιεχόμενο του καταχωρητή R1 στη μνήμη δεδομένων. Η διεύθυνση εγγραφής προκύπτει από το άθροισμα της τιμής του καταχωρητή R0 (20) και της άμεσης τιμής 2, με αποτέλεσμα η τελική διεύθυνση να είναι 22. Στο πρόγραμμα, η εντολή αυτή βρίσκεται στην τέταρτη θέση, επομένων ο μετρητής προγράμματος (PC) έχει τιμή 12. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E5801002. Όσον αφορά τα σήματα ελέγχου, η έξοδος ALUResult αντιστοιχεί στη διεύθυνση 22, η οποία υπολογίζεται από την ALU. Η έξοδος WriteData περιέχει την τιμή 20, δηλαδή το περιεχόμενο του καταχωρητή R1 που πρόκειται να εγγραφεί στη μνήμη. Τέλος, η έξοδος Result λαμβάνει την ίδια τιμή με την ALUResult, δηλαδή 22, αφού το αποτέλεσμα της πράξης προέρχεται απευθείας από την ALU.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 88 – Αποτέλεσμα εντολής STR R1, [R0,2] σε Behavioral Simulation**



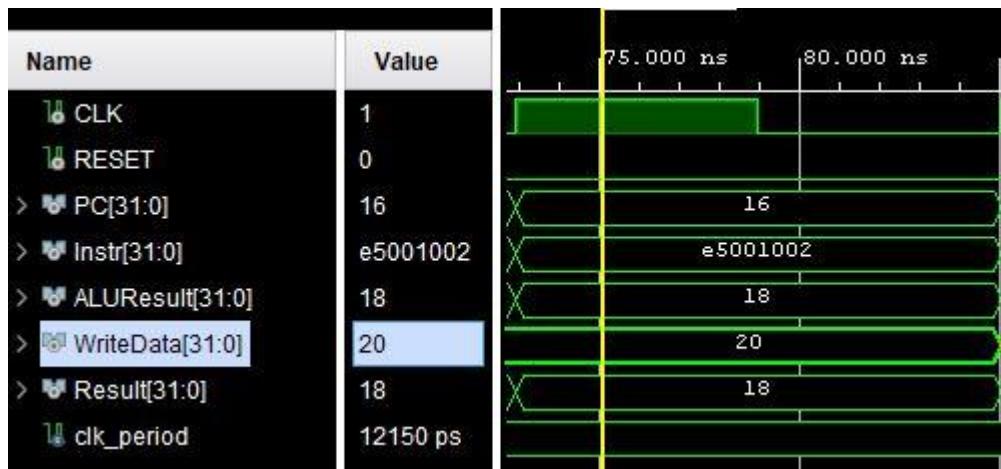
**Εικόνα 89 – Αποτέλεσμα εντολής STR R1, [R0,2] σε Implementation Simulation**

### 2.3.5 ΕΝΤΟΛΗ STR R1, [R0,-2]

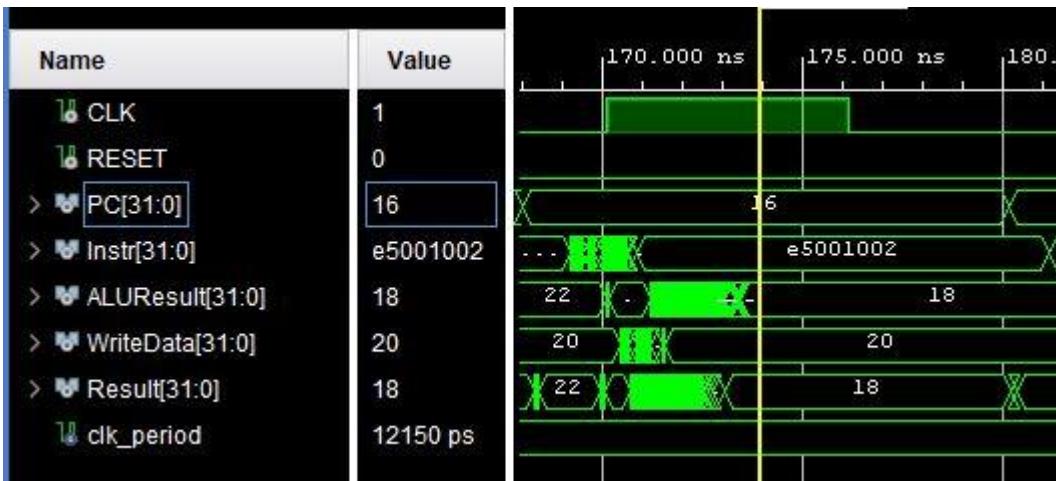
Η εντολή STR αποθηκεύει το περιεχόμενο του καταχωρητή R1 στη μνήμη δεδομένων. Σε αυτήν την περίπτωση, η διεύθυνση εγγραφής προκύπτει από το άθροισμα της τιμής του καταχωρητή R0 (20) και της άμεσης τιμής -2, με αποτέλεσμα η τελική διεύθυνση να είναι 18. Η εντολή βρίσκεται στην πέμπτη θέση του προγράμματος, επομένως ο μετρητής προγράμματος (PC) έχει τιμή 16. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E5001002.

Όσον αφορά τα σήματα ελέγχου, η έξοδος ALUResult υπολογίζει σωστά τη διεύθυνση 18 μέσω της ALU, ενώ η έξοδος WriteData περιέχει την τιμή 20, δηλαδή το περιεχόμενο του καταχωρητή R1 που πρόκειται να γραφτεί στη μνήμη. Τέλος, η έξοδος Result είναι ίση με το ALUResult, δηλαδή 18, καθώς προέρχεται απευθείας από τον υπολογισμό της ALU.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 90 – Αποτέλεσμα εντολής STR R1, [R0,-2] σε Behavioral Simulation**



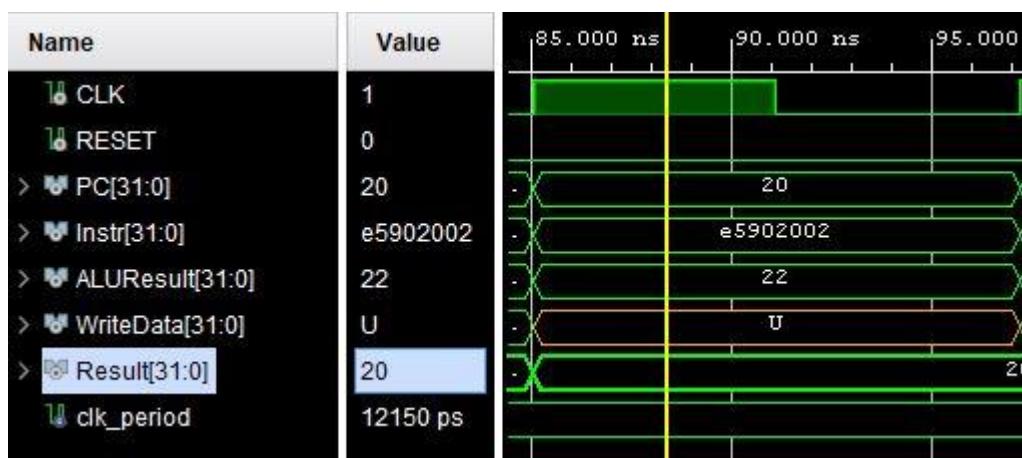
**Εικόνα 91 – Αποτέλεσμα εντολής STR R1, [R0,-2] σε Implementation Simulation**

### 2.3.6 ΕΝΤΟΛΗ LDR R2, [R0,2]

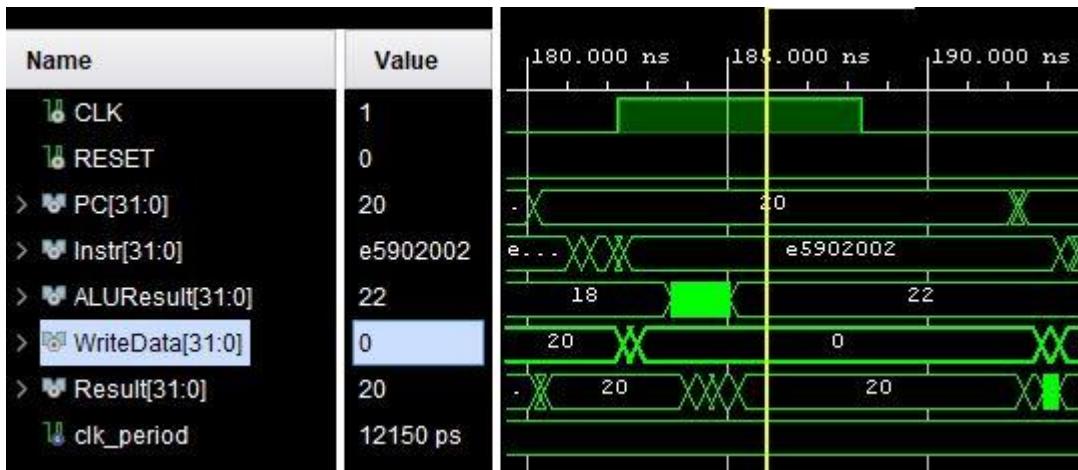
Η εντολή LDR πραγματοποιεί φόρτωση δεδομένων από τη μνήμη δεδομένων στον καταχωρητή R2. Η διεύθυνση ανάγνωσης προκύπτει από το άθροισμα της τιμής του καταχωρητή R0 (20) και της άμεσης τιμής 2, με αποτέλεσμα η τελική διεύθυνση να είναι 22. Από προηγούμενη εκτέλεση της εντολής STR, στη διεύθυνση αυτή έχει ήδη αποθηκευτεί η τιμή 20, επομένως το περιεχόμενο της μνήμης στη διεύθυνση 22 είναι 20. Η εντολή βρίσκεται στην έκτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει τιμή 20. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E5902002.

Όσον αφορά τα σήματα ελέγχου, η έξοδος ALUResult υπολογίζει τη διεύθυνση 22, στην οποία γίνεται η πρόσβαση στη μνήμη δεδομένων. Η έξοδος WriteData είναι αδιάφορη, καθώς δε πραγματοποιείται εγγραφή αλλά ανάγνωση. Τέλος, η έξοδος Result λαμβάνει την τιμή 20, δηλαδή το περιεχόμενο της μνήμης που φορτώθηκε στον καταχωρητή R2.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 92 – Αποτέλεσμα εντολής LDR R2, [R0,2] σε Behavioral Simulation**



**Εικόνα 93 – Αποτέλεσμα εντολής LDR R2, [R0,2] σε Implementation Simulation**

### 2.3.7 ΕΝΤΟΛΗ LDR R3, [R0,-2]

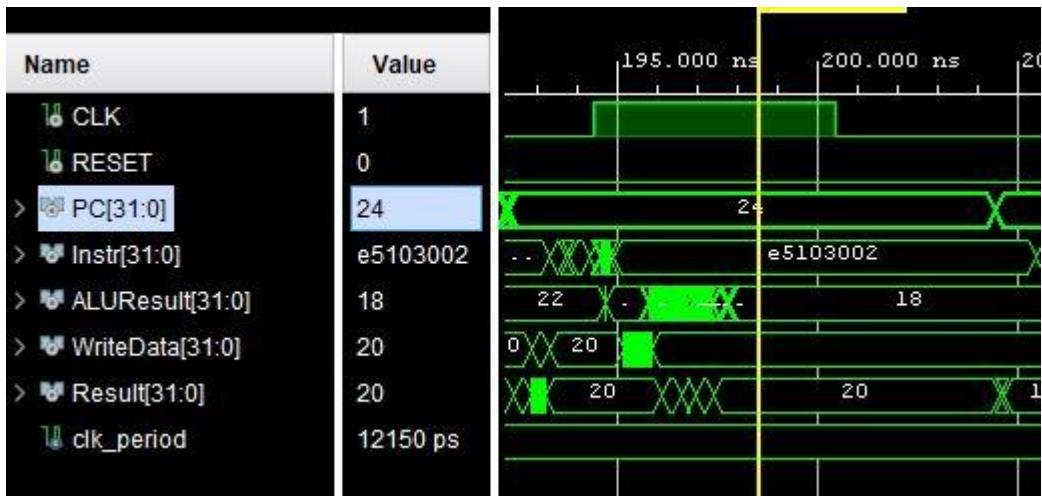
Η εντολή LDR εκτελεί φόρτωση δεδομένων από τη μνήμη δεδομένων στον καταχωρητή R3. Η διεύθυνση προσπέλασης υπολογίζεται από το άθροισμα της τιμής του καταχωρητή R0 (20) και της αρνητικής σταθεράς -2, με αποτέλεσμα η τελική διεύθυνση να είναι 18. Από προηγούμενη εντολή STR R1, [R0, #-2], γνωρίζουμε ότι στη διεύθυνση αυτή είχε αποθηκευτεί η τιμή 20, επομένως κατά την εκτέλεση της τρέχουσας εντολής η τιμή αυτή ανακτάται και φορτώνεται στον καταχωρητή R3.

Η εντολή βρίσκεται στην έβδομη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 24. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E5103002. Όσον αφορά τα σήματα ελέγχου, η έξοδος ALUResult υπολογίζει τη διεύθυνση 18, ενώ η έξοδος WriteData είναι αδιάφορη, αφού δεν πραγματοποιείται εγγραφή στη μνήμη. Τέλος, η έξοδος Result λαμβάνει την τιμή 20, που αντιστοιχεί στο περιεχόμενο της μνήμης στη διεύθυνση 18 και γράφεται στον καταχωρητή R3.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 94 – Αποτέλεσμα εντολής LDR R2, [R0,-2] σε Behavioral Simulation**



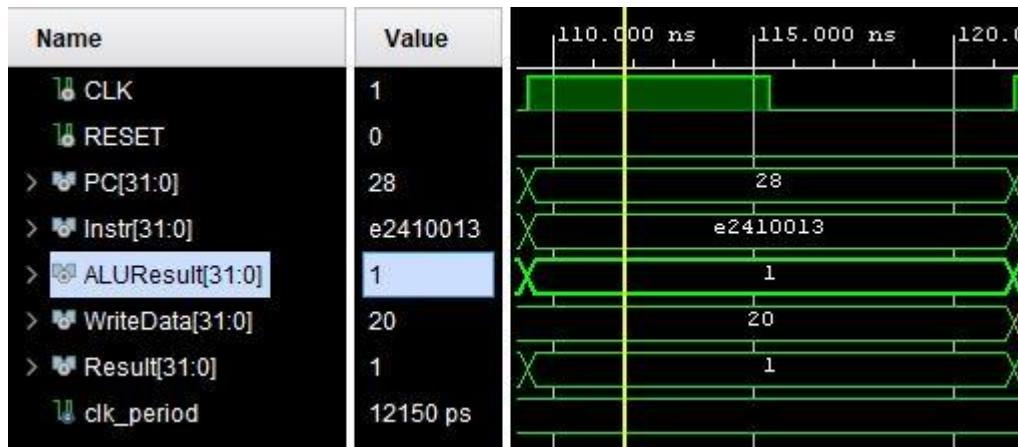
**Εικόνα 95 – Αποτέλεσμα εντολής LDR R2, [R0,-2] σε Implementation Simulation**

### 2.3.8 ΕΝΤΟΛΗ SUB R0, R1, 19

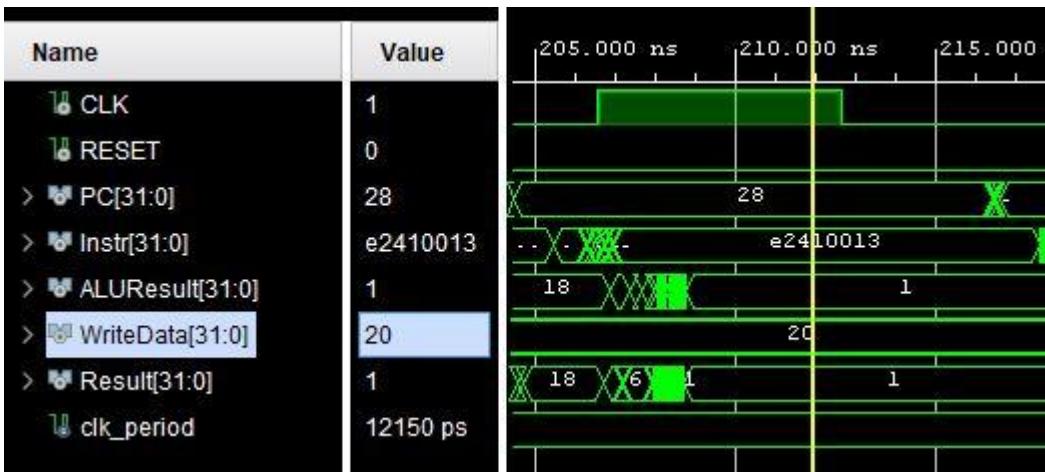
Η εντολή SUB εκτελεί αφαιρεση μεταξύ του περιεχομένου του καταχωρητή R1 και της άμεσης τιμής 19. Εφόσον ο καταχωρητής R1 περιέχει την τιμή 20 (από την προηγούμενη εντολή MOV R1, R0), το αποτέλεσμα της πράξης είναι  $20 - 19 = 1$ . Η εντολή βρίσκεται στην όγδοη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 28.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E2410013. Όσον αφορά τα σήματα ελέγχου, η έξοδος ALUResult παράγει την τιμή 1, που προκύπτει από την πράξη της αφαιρεσης. Η έξοδος WriteData είναι αδιάφορη, καθώς δε γίνεται εγγραφή στη μνήμη δεδομένων. Τέλος, η έξοδος Result είναι ίση με το ALUResult, δηλαδή 1, καθώς η τιμή προέρχεται απευθείας από την ALU και όχι από τη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 96 – Αποτέλεσμα εντολής SUB R0, R1, 19 σε Behavioral Simulation**

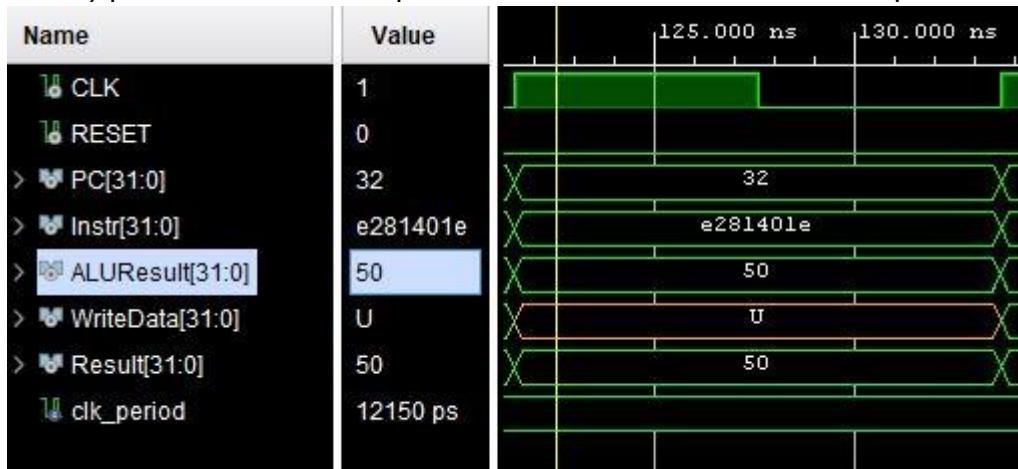


Εικόνα 97 – Αποτέλεσμα εντολής **SUB R0, R1, 19** σε Implementation Simulation

### 2.3.9 ΕΝΤΟΛΗ ADD R4, R1, 30

Η εντολή ADD εκτελεί πρόσθεση μεταξύ του περιεχομένου του καταχωρητή R1 και της άμεσης τιμής 30. Εφόσον ο καταχωρητής R1 περιέχει την τιμή 20, το αποτέλεσμα της πράξης είναι  $20 + 30 = 50$ . Η εντολή βρίσκεται στην ένατη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 32. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E281401E. Όσον αφορά τα σήματα ελέγχου, η έξοδος ALUResult παράγει την τιμή 50, η οποία προκύπτει από την πράξη της πρόσθεσης. Η έξοδος WriteData είναι αδιάφορη, καθώς δεν πραγματοποιείται εγγραφή στη μνήμη. Τέλος, η έξοδος Result είναι ίση με το ALUResult, δηλαδή **50**, αφού το αποτέλεσμα προέρχεται από την ALU και όχι από τη μνήμη δεδομένων.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



Εικόνα 98 – Αποτέλεσμα εντολής **ADD R4, R1, 30** σε Behavioral Simulation



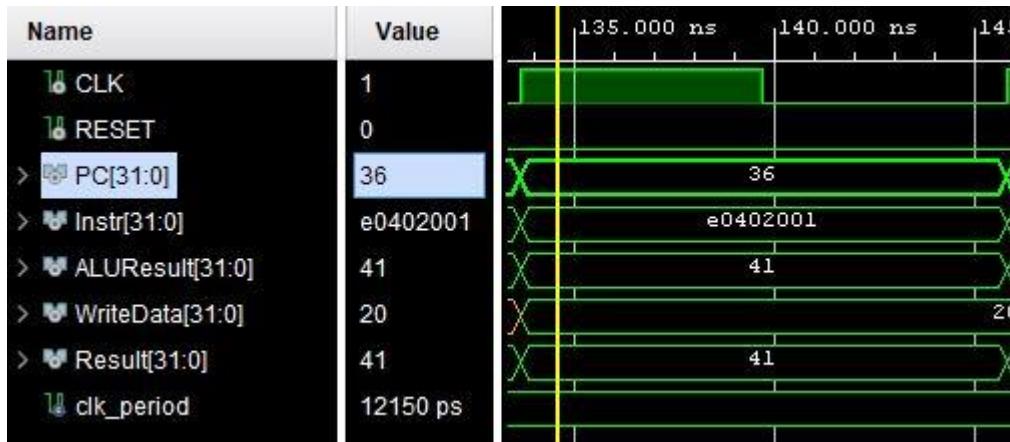
Εικόνα 99 – Αποτέλεσμα εντολής **ADD R4, R1, 30** σε Implementation Simulation

### 2.3.10 ΕΝΤΟΛΗ SUB R2, R0, R1

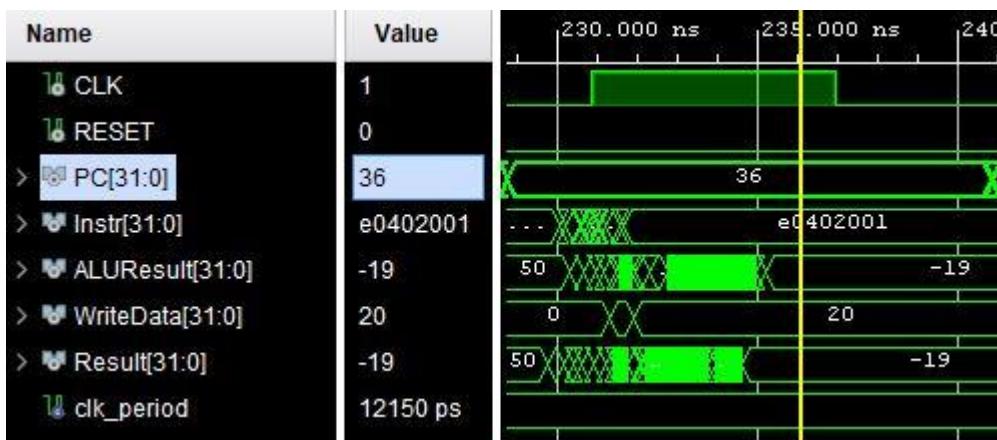
Η εντολή SUB εκτελεί αφαίρεση μεταξύ του περιεχομένου του καταχωρητή R0 και του καταχωρητή R1. Εφόσον ο καταχωρητής R0 περιέχει την τιμή 1 και ο καταχωρητής R1 την τιμή 20, το αποτέλεσμα της πράξης είναι  $1 - 20 = -19$ . Η εντολή βρίσκεται στη δέκατη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 36.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E0402001. Όσον αφορά τα σήματα ελέγχου, η έξοδος ALUResult λαμβάνει την τιμή -19, που προκύπτει από την πράξη της αφαίρεσης. Η έξοδος WriteData είναι αδιάφορη, καθώς δε γίνεται εγγραφή στη μνήμη δεδομένων. Τέλος, η έξοδος Result είναι ίση με το ALUResult, δηλαδή -19, καθώς το αποτέλεσμα προέρχεται απευθείας από την ALU.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



Εικόνα 100 – Αποτέλεσμα εντολής SUB R2, R0, R1 σε Behavioral Simulation



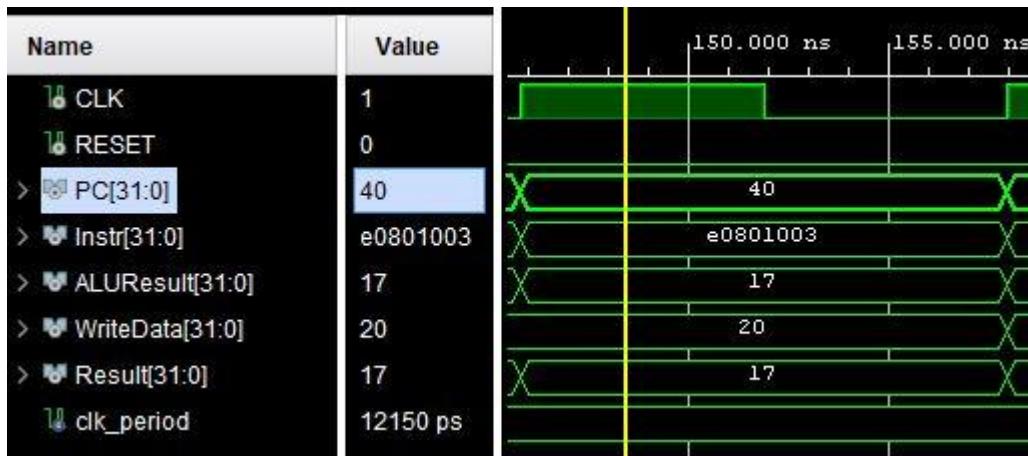
Εικόνα 101 – Αποτέλεσμα εντολής SUB R2, R0, R1 σε Implementation Simulation

### 2.3.11 ΕΝΤΟΛΗ ADD R1, R0, R3

Η εντολή ADD πραγματοποιεί πρόσθεση μεταξύ του περιεχομένου του καταχωρητή R0 και του περιεχομένου του καταχωρητή R3. Δεδομένου ότι ο καταχωρητής R0 περιέχει την τιμή 1 και ο καταχωρητής R3 την τιμή 20 το αποτέλεσμα της πράξης είναι  $20 + 1 = 21$ . Η εντολή βρίσκεται στην εντεκατη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 40.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E0801003. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 21, που προκύπτει από την αφαίρεση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή 21.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 102 – Αποτέλεσμα εντολής ADD R1, R0, R3 σε Behavioral Simulation**



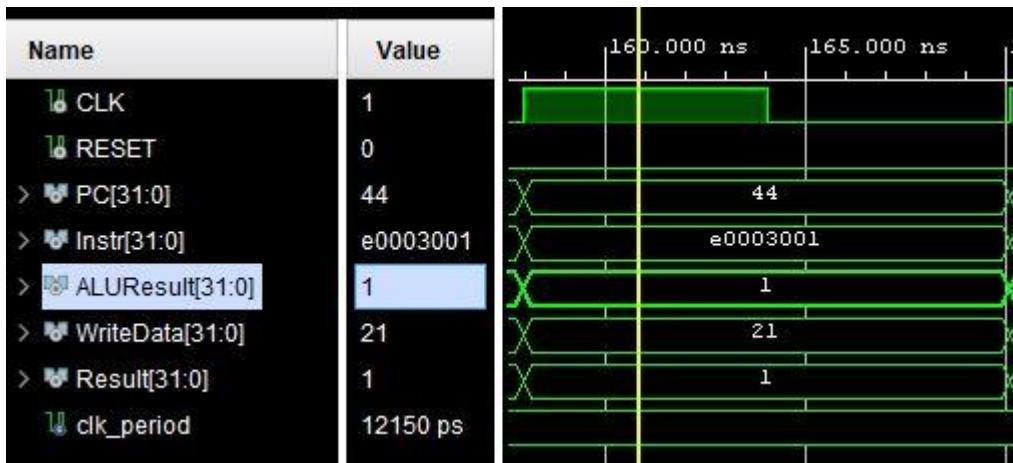
**Εικόνα 103 – Αποτέλεσμα εντολής ADD R1, R0, R3 σε Implementation Simulation**

### 2.3.12 ΕΝΤΟΛΗ AND R3, R0, R1

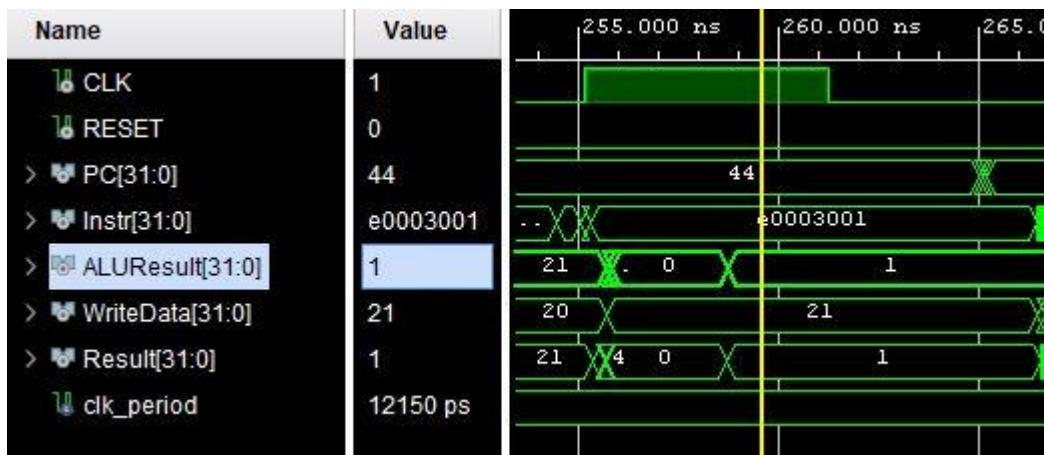
Η εντολή AND πραγματοποιεί το λογικό and μεταξύ του περιεχομένου του καταχωρητή R0 και του περιεχομένου του καταχωρητή R1. Δεδομένου ότι ο καταχωρητής R0 περιέχει την τιμή 1 και ο καταχωρητής R3 την τιμή 21 το αποτέλεσμα της πράξης αυτής είναι 1. Η εντολή βρίσκεται στην δωδέκατη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 44.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E0003001. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 1, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 1.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 104 – Αποτέλεσμα εντολής AND R3, R0, R1 σε Behavioral Simulation**



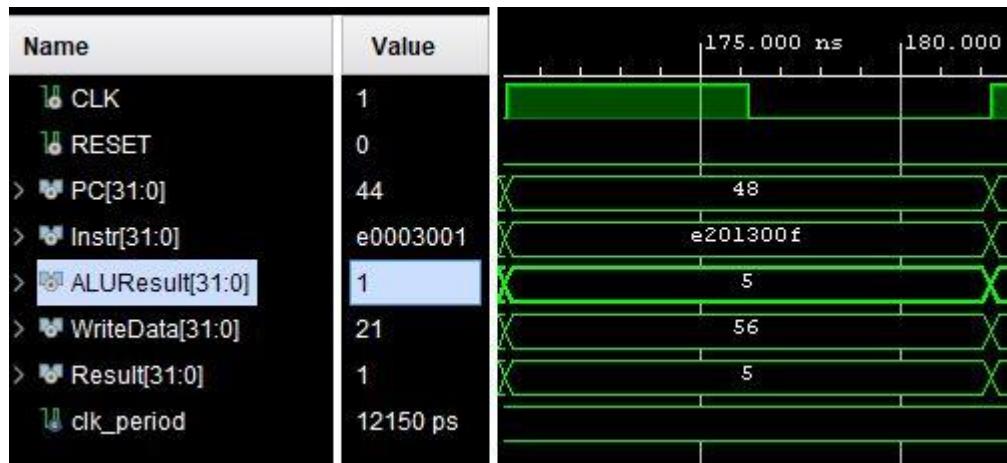
**Εικόνα 105 – Αποτέλεσμα εντολής AND R3, R0, R1 σε Implementation Simulation**

### 2.3.13 ΕΝΤΟΛΗ AND R3, R1, 15

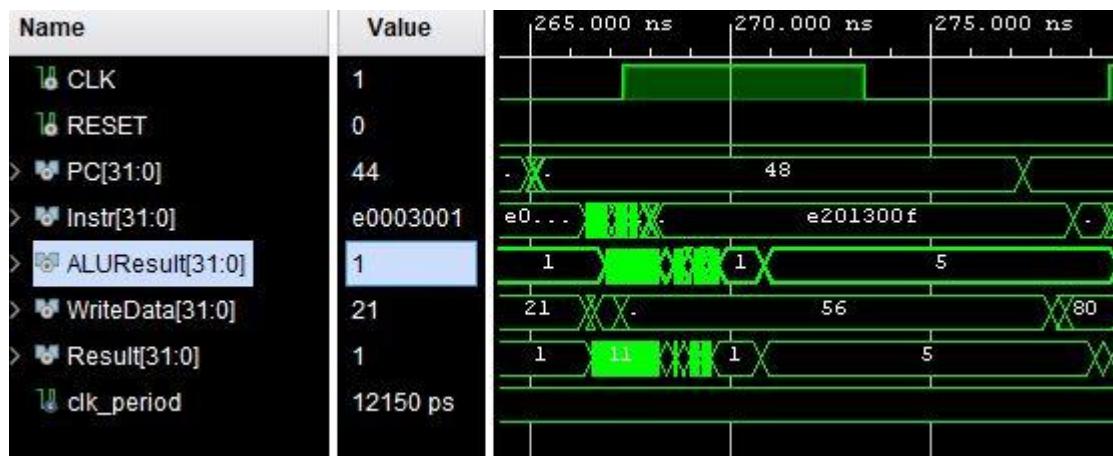
Η εντολή AND πραγματοποιεί το λογικό and μεταξύ του περιεχομένου του καταχωρητή R1 και της τιμής 15. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης αυτής είναι 5. Η εντολή βρίσκεται στην δέκατη τρίτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 48.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E201300F. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 5, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 5.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 106 – Αποτέλεσμα εντολής AND R3, R1, 15 σε Behavioral Simulation**



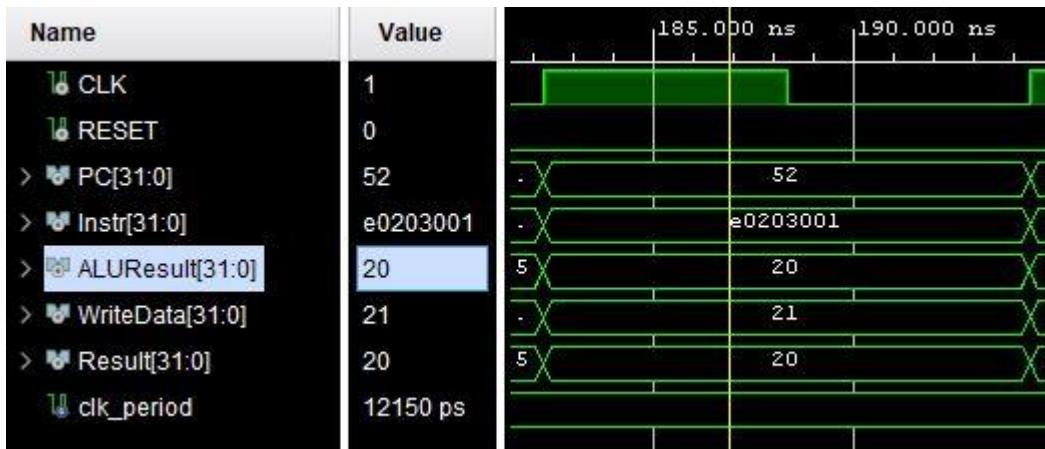
**Εικόνα 107 – Αποτέλεσμα εντολής AND R3, R1, 15 σε Implementation Simulation**

### 2.3.14 ΕΝΤΟΛΗ XOR R3, R0, R1

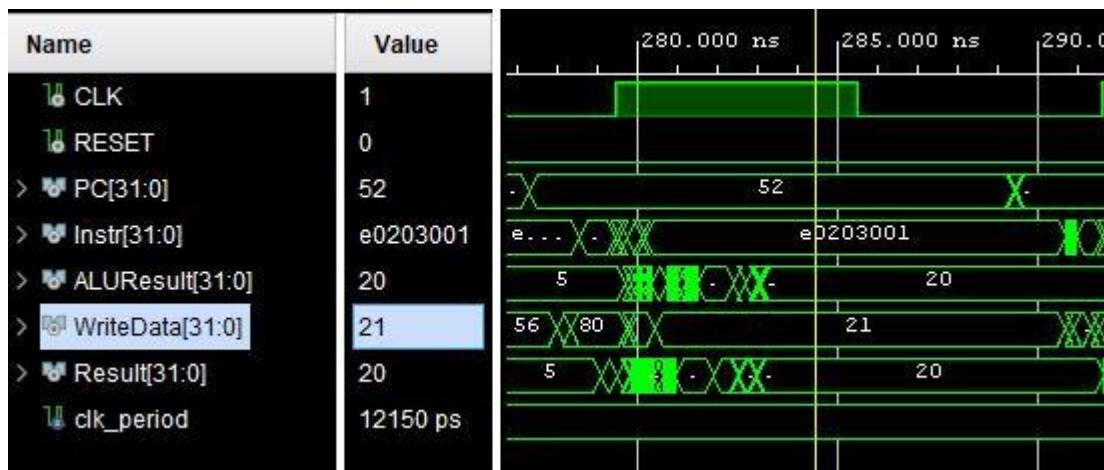
Η εντολή XOR πραγματοποιεί το λογικό xor μεταξύ του περιεχομένου του καταχωρητή R0 και του περιεχομένου του καταχωρητή R1. Δεδομένου ότι ο καταχωρητής R0 περιέχει την τιμή 1 και ο καταχωρητής R1 την τιμή 21 το αποτέλεσμα της πράξης αυτής είναι 20. Η εντολή βρίσκεται στην δέκατη τέταρτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 52.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E0203001. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 20, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 20.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 108 – Αποτέλεσμα εντολής XOR R3, R0, R1 σε Behavioral Simulation**



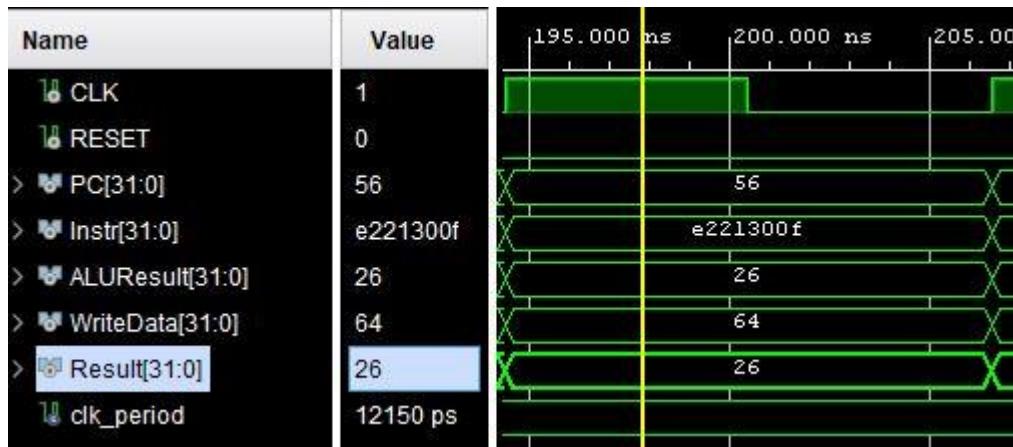
**Εικόνα 109 – Αποτέλεσμα εντολής XOR R3, R0, R1 σε Implementation Simulation**

### 2.3.15 ΕΝΤΟΛΗ XOR R3, R1, 15

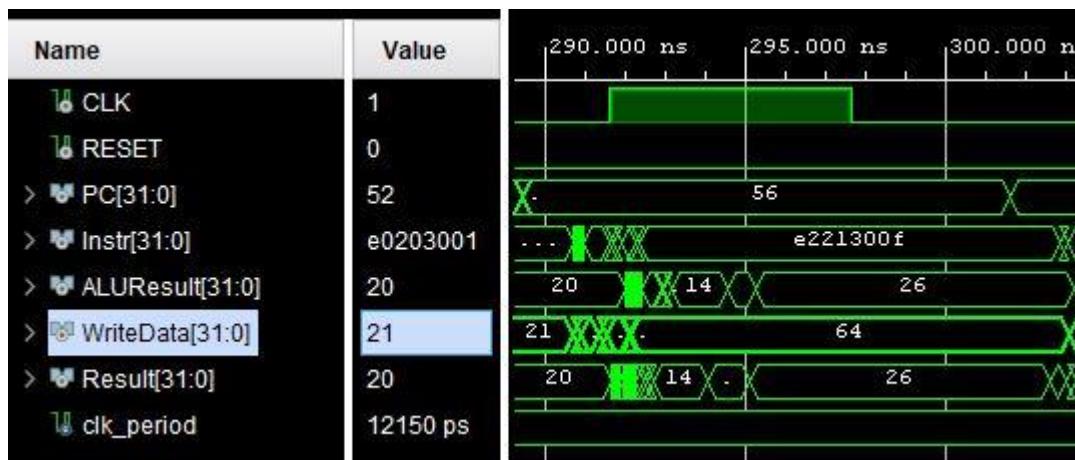
Η εντολή XOR πραγματοποιεί το λογικό xor μεταξύ του περιεχομένου του καταχωρητή R1 και της τιμής 15. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης αυτής είναι 26. Η εντολή βρίσκεται στην δέκατη πεμπτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 56.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E221300F. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 26, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 26.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 110 – Αποτέλεσμα εντολής XOR R3, R1, 15 σε Behavioral Simulation**



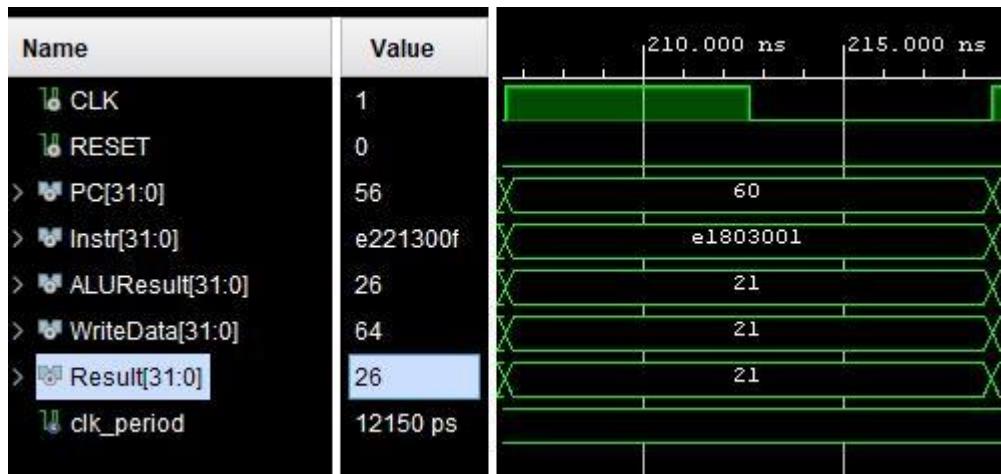
**Εικόνα 111 – Αποτέλεσμα εντολής XOR R3, R1, 15 σε Implementation Simulation**

### 2.3.16 ΕΝΤΟΛΗ OR R3, R0, R1

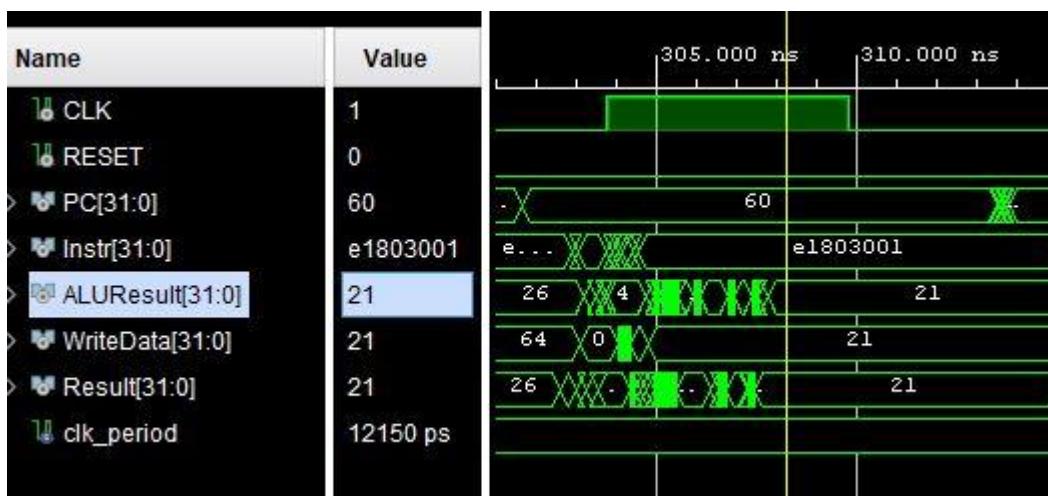
Η εντολή OR πραγματοποιεί το λογικό ορ μεταξύ του περιεχομένου του καταχωρητή R0 και του περιεχομένου του καταχωρητή R1. Δεδομένου ότι ο καταχωρητής R0 περιέχει την τιμή 1 και ο καταχωρητής R1 την τιμή 21 το αποτέλεσμα της πράξης αυτής είναι 21. Η εντολή βρίσκεται στην δέκατη έκτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 60.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1803001. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 21, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 21.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 112 – Αποτέλεσμα εντολής OR R3, R0, R1, σε Behavioral Simulation**



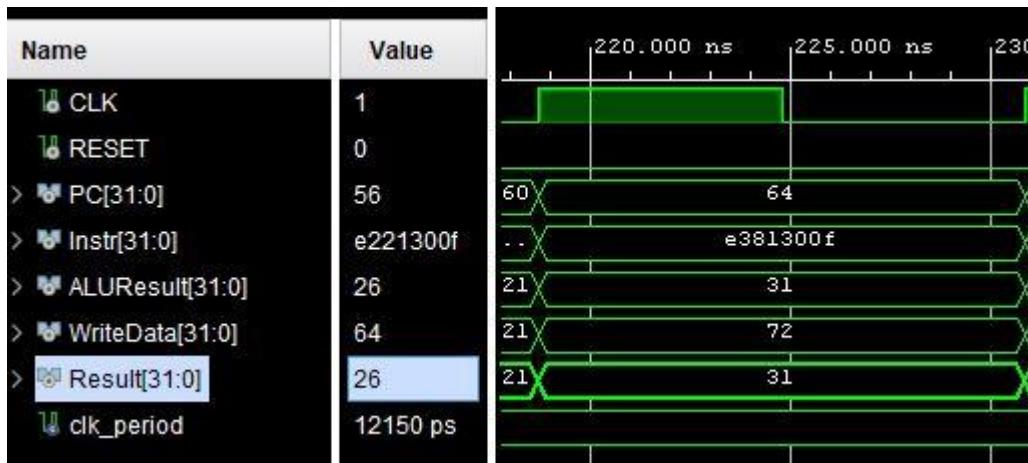
**Εικόνα 113 – Αποτέλεσμα εντολής OR R3, R0, R1, σε Implementation Simulation**

### 2.3.17 ΕΝΤΟΛΗ OR R3, R1, 15

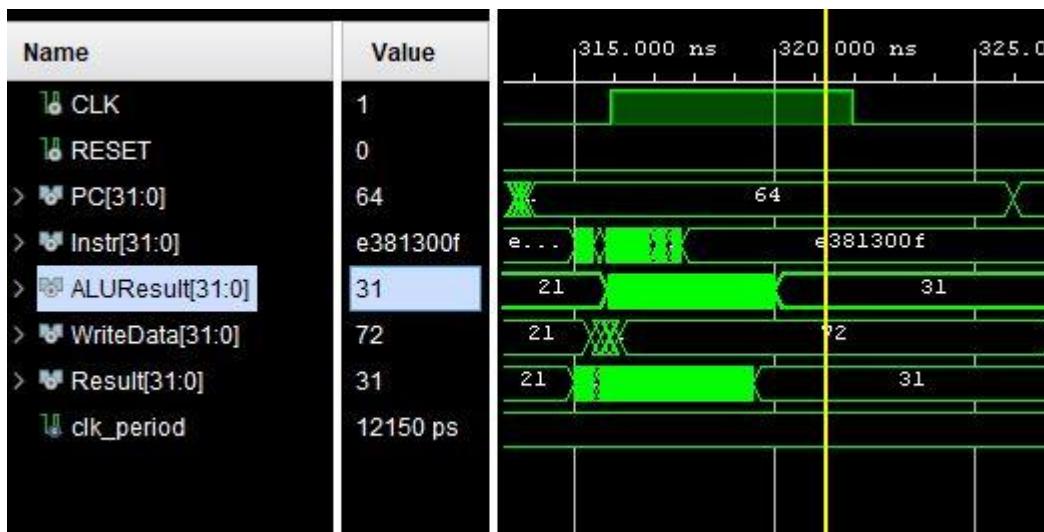
Η εντολή OR πραγματοποιεί το λογικό ορ μεταξύ του περιεχομένου του καταχωρητή R1 και της τιμής 15. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης αυτής είναι 31. Η εντολή βρίσκεται στην δέκατη έβδομη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 64.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E381300F. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 31, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 31.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 114 – Αποτέλεσμα εντολής OR R3, R1, 15 σε Behavioral Simulation**



**Εικόνα 115 – Αποτέλεσμα εντολής OR R3, R1, 15 σε Implementation Simulation**

### 2.3.18 ΕΝΤΟΛΗ LSL R3, R1, 2

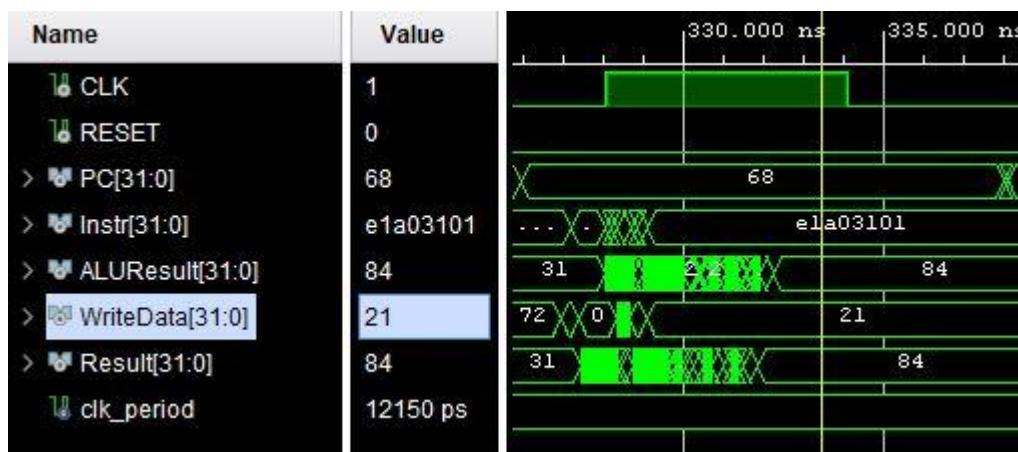
Η εντολή LSL πραγματοποιεί λογική ολίσθηση προς τα αριστερά του περιεχομένου του καταχωρητή R1 κατά 2. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης αυτής είναι 84. Η εντολή βρίσκεται στην δέκατη όγδοη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 68.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1A03101. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 84, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 84.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 116 – Αποτέλεσμα εντολής LSL R3, R1, 2 σε Behavioral Simulation**



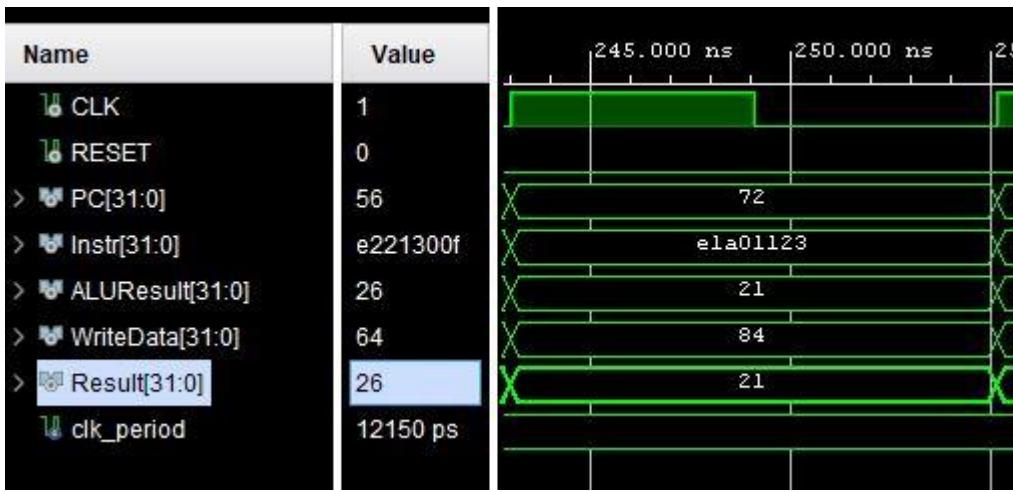
**Εικόνα 117 – Αποτέλεσμα εντολής LSL R3, R1, 2 σε Implementation Simulation**

### 2.3.19 ΕΝΤΟΛΗ LSL R1, R3, 2

Η εντολή LSL πραγματοποιεί λογική ολισθηση προς τα δεξιά του περιεχομένου του καταχωρητή R3 κατά 2. Δεδομένου ότι ο καταχωρητής R3 περιέχει την τιμή 84 το αποτέλεσμα της πράξης αυτής είναι 21. Η εντολή βρίσκεται στην δέκατη ένατη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 72.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1A01123. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 21, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 21.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 118 – Αποτέλεσμα εντολής LSR R1, R3, 2 σε Behavioral Simulation**

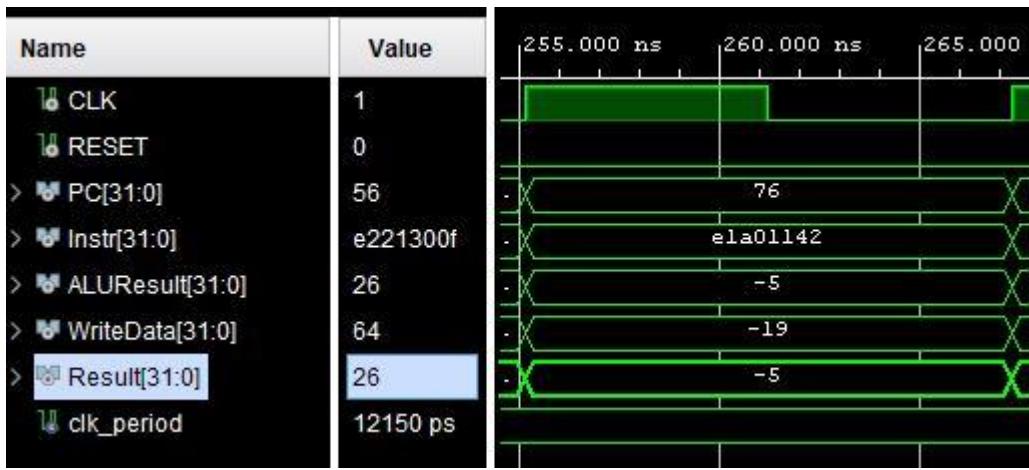


**Εικόνα 119 – Αποτέλεσμα εντολής LSR R1, R3, 2 σε Implementation Simulation**

### 2.3.20 ΕΝΤΟΛΗ ASR R1, R2, 2

Η εντολή ASR πραγματοποιεί αριθμητική ολίσθηση προς τα δεξιά του περιεχομένου του καταχωρητή R2 κατά 2. Δεδομένου ότι ο καταχωρητής R3 περιέχει την τιμή -19 το αποτέλεσμα της πράξης αυτής είναι -5. Η εντολή βρίσκεται στην εικοστή θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 76. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1A01142. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή -5, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με -5.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 120 – Αποτέλεσμα εντολής ASR R1, R2, 2 σε Behavioral Simulation**



**Εικόνα 121 – Αποτέλεσμα εντολής ASR R1, R2, 2 σε Implementation Simulation**

### 2.3.21 ΕΝΤΟΛΗ ROR R1, R3, 2

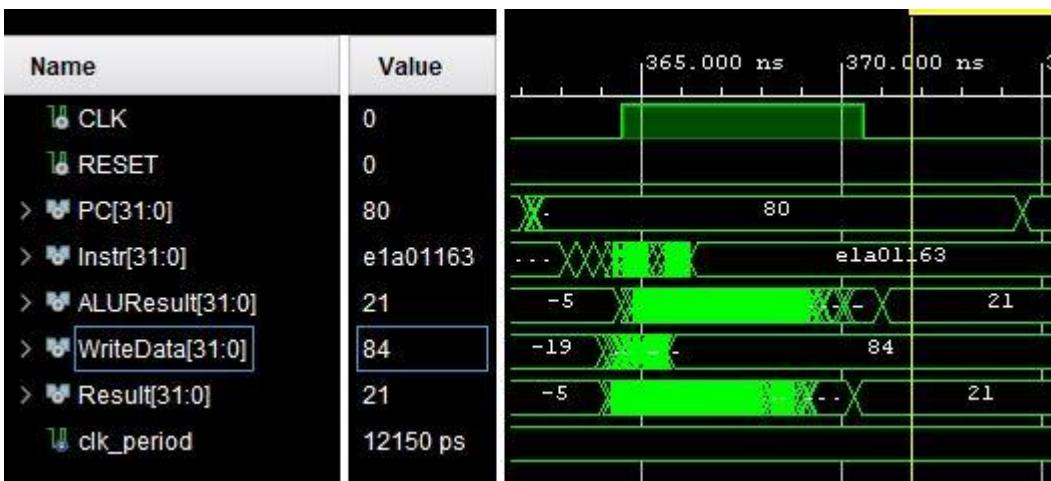
Η εντολή ROR πραγματοποιεί κυκλική ολίσθηση προς τα δεξιά του περιεχομένου του καταχωρητή R3 κατά 2. Δεδομένου ότι ο καταχωρητής R3 περιέχει την τιμή 84 το αποτέλεσμα της πράξης αυτής είναι 21. Η εντολή βρίσκεται στην εικοστή πρώτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 80.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1A01163. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 21, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 21.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 122 – Αποτέλεσμα εντολής ROR R1, R3, 2 σε Behavioral Simulation**



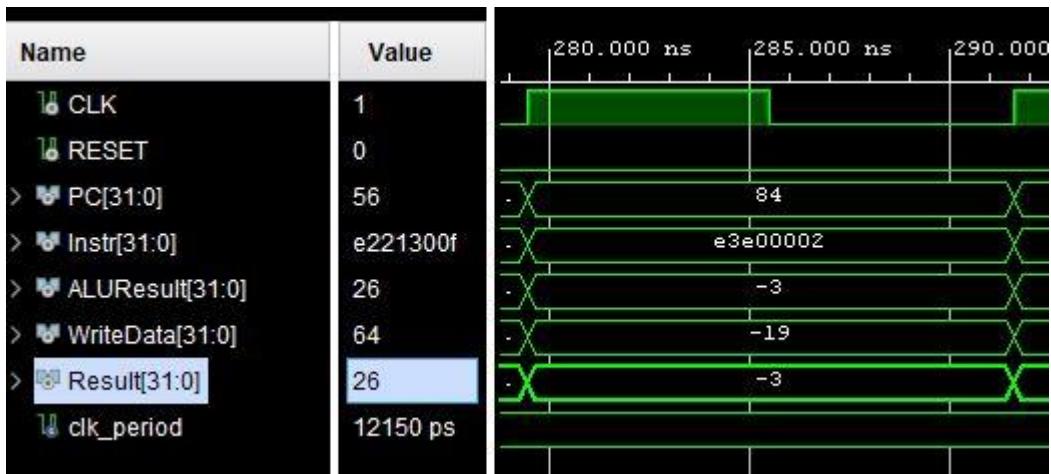
**Εικόνα 123 – Αποτέλεσμα εντολής ROR R1, R3, 2 σε Implementation Simulation**

### 2.3.22 ΕΝΤΟΛΗ MVN R0, 2

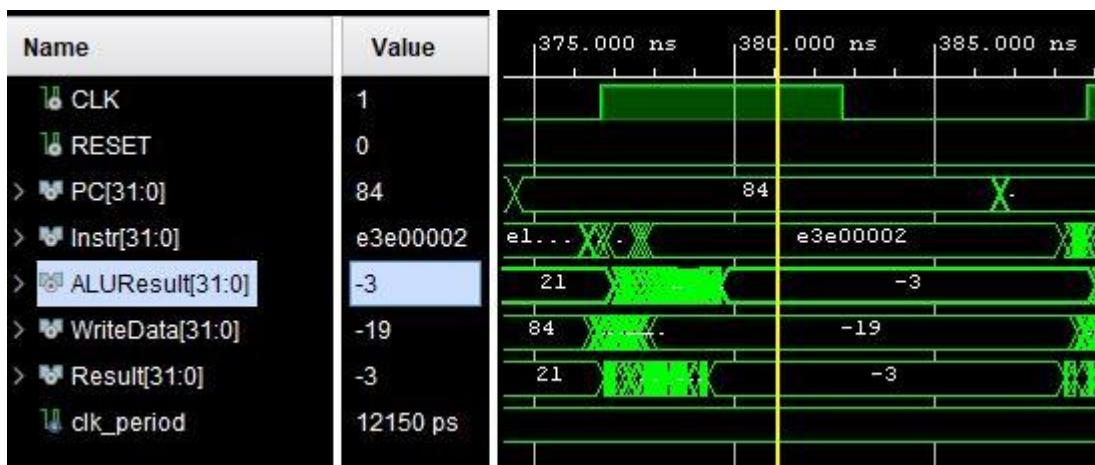
Η εντολή MVN μεταφέρει στον καταχωρητή R0 την τιμή not imm. Το αποτέλεσμα της πράξης αυτής είναι -3. Η εντολή βρίσκεται στην εικοστή δεύτερη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 84.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E3E00002. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή -3, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με -3.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 124 – Αποτέλεσμα εντολής MVN R0, 2 σε Behavioral Simulation**



**Εικόνα 125 – Αποτέλεσμα εντολής MVN R0, 2 σε Implementation Simulation**

### 2.3.23 ΕΝΤΟΛΗ MVN R4, R3

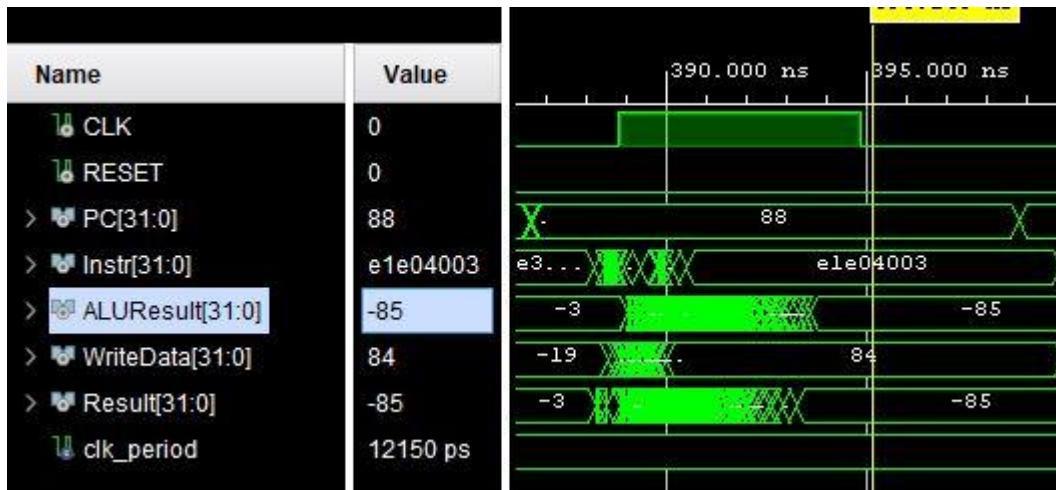
Η εντολή MVN μεταφέρει στον καταχωρητή R4 την τιμή την τιμή ποτ του καταχωρητή R3 η οποία είναι 84. Το αποτέλεσμα της πράξης αυτής είναι -85. Η εντολή βρίσκεται στην εικοστή τρίτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 88.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E1E04003. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή -85, που προκύπτει από την πράξη. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με -85.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 126 – Αποτέλεσμα εντολής MVN R4, R3 σε Behavioral Simulation**



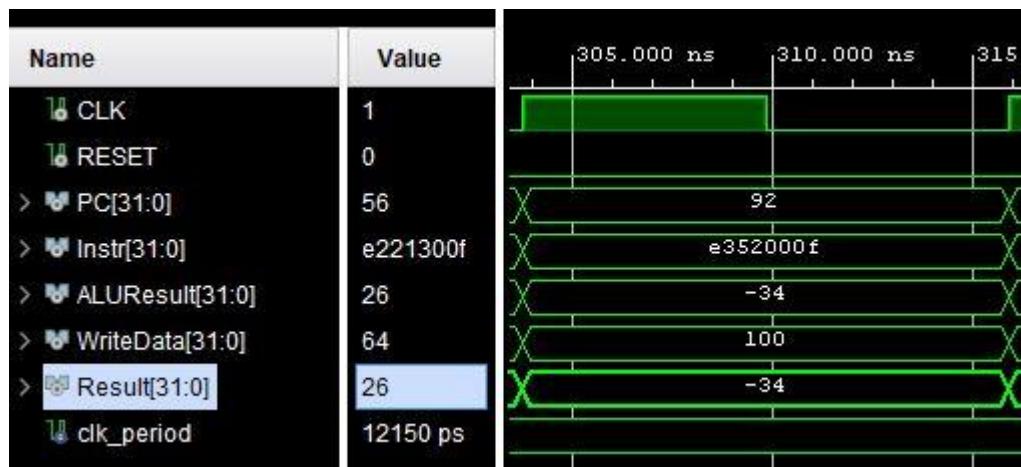
**Εικόνα 127 – Αποτέλεσμα εντολής MVN R4, R3 σε Implementation Simulation**

### 2.3.24 ΕΝΤΟΛΗ CMP R2, 15

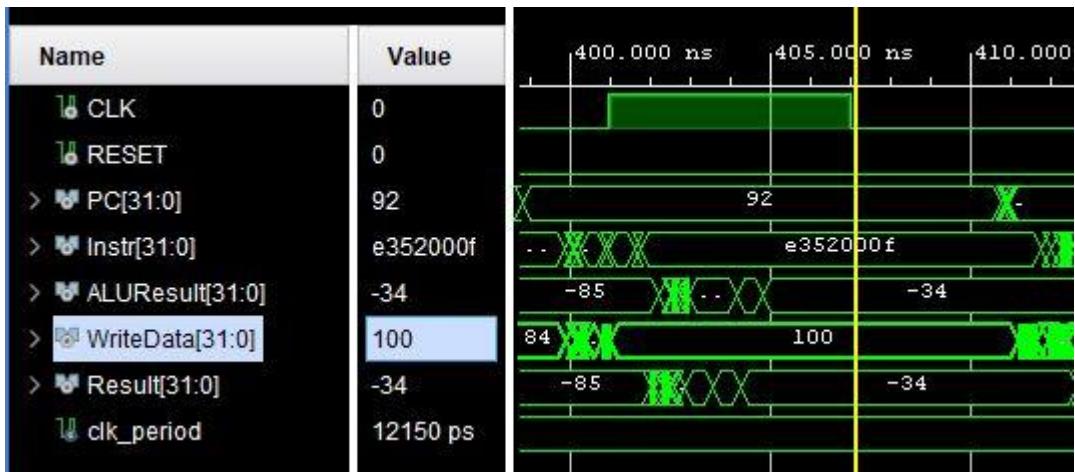
Η εντολή CMP πραγματοποιεί αφαίρεση μεταξύ του περιεχομένου του καταχωρητή R2 και της άμεσης τιμής 15. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή -19, το αποτέλεσμα της πράξης είναι  $-19 - 15 = -34$ . Η συγκεκριμένη εντολή ενημερώνει μόνο τις σημαίες. Η εντολή βρίσκεται στην εικοστή τέταρτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 92.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E352000F. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή -19, που προκύπτει από την αφαίρεση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με -19.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 128 – Αποτέλεσμα εντολής CMP R2, 15 σε Behavioral Simulation**



**Εικόνα 129 – Αποτέλεσμα εντολής CMP R2, 15 σε Implementation Simulation**

### 2.3.25 ΕΝΤΟΛΗ BEQ COROUTINE\_EQ

Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο το μνημονικό EQ ισχύει. Από την εκτέλεση της προηγούμενης εντολής έχουμε  $Z=0$ , άρα δεν εκτελείται. Η εντολή βρίσκεται στην εικοστή πέμπτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 96.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 0A000010. Όσο αφορά τα σήματα ALUResult και Result έχουν την θέση της εντολής η οποία θα εκτελούντα αν ισχυε το μνημονικό. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 130 – Αποτέλεσμα εντολής BEQ COROUTINE\_EQ σε Behavioral Simulation**



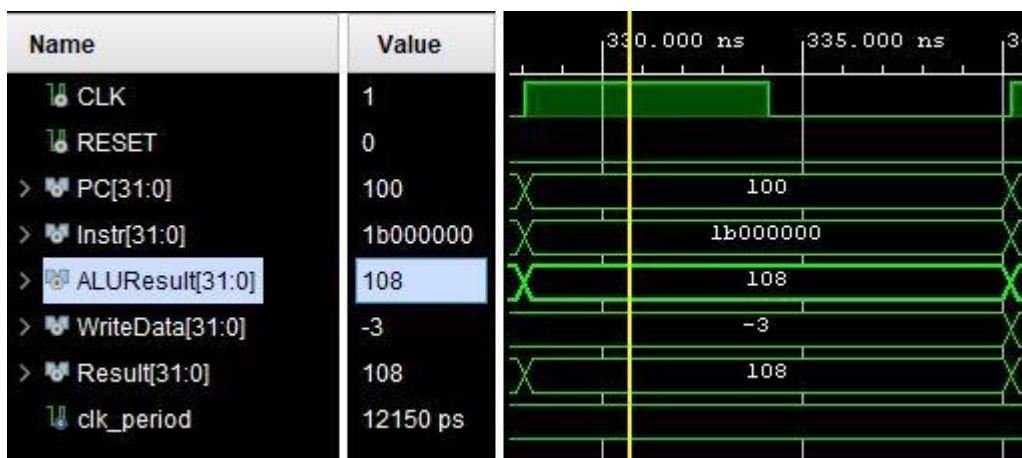
**Εικόνα 131 – Αποτέλεσμα εντολής BEQ COROUTINE\_EQ σε Implementation Simulation**

### 2.3.26 ΕΝΤΟΛΗ BLNE COROUTINE\_NE

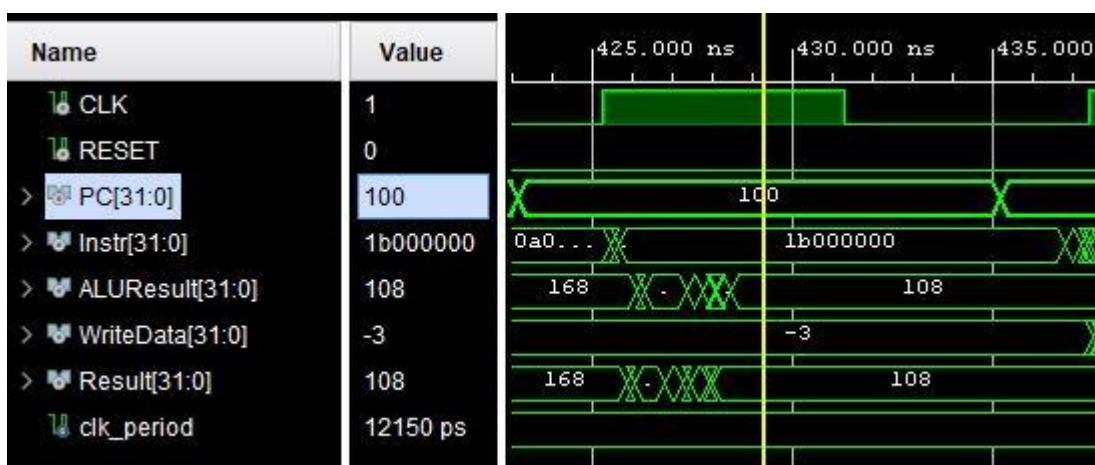
Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο το μνημονικό NE ισχύει. Από την εκτέλεση της προηγούμενης εντολής έχουμε N=1, άρα εκτελείται. Η εντολή βρίσκεται στην εικοστή έκτη θέση του προγράμματος, με τον μετρητή προγράμματος (PC) να έχει την τιμή 100.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 1B000000. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 108, η οποία είναι η θέση της επόμενης εντολής που θα εκτελεστεί. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 108. Επίσης στην συγκεκριμένη εντολή ο link register παίρνει τιμή R14 = PC + 4 = 104.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 132 – Αποτέλεσμα εντολής BLNE COROUTINE\_NE σε Behavioral Simulation**



**Εικόνα 133 – Αποτέλεσμα εντολής BLNE COROUTINE\_NE σε Implementation Simulation**

### 2.3.27 ΕΝΤΟΛΗ ADDS R0, R1, -1

Η εντολή ADD πραγματοποιεί πρόσθεση μεταξύ του περιεχομένου του καταχωρητή R1 και της άμεσης τιμής 30. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης είναι  $21 - 1 = 20$ . Επίσης η συγκεκριμένη εντολή ενεργοποιεί τη σημαία C. Ο μετρητής προγράμματος (PC) έχει την τιμή 108, τιμή η οποία είναι το ALUResult της προηγουμενής εντολής διακλάδωσης.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E2510001. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 20, που προκύπτει από την αφαίρεση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 20.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 134 – Αποτέλεσμα εντολής ADDS R0, R1, -1 σε Behavioral Simulation**



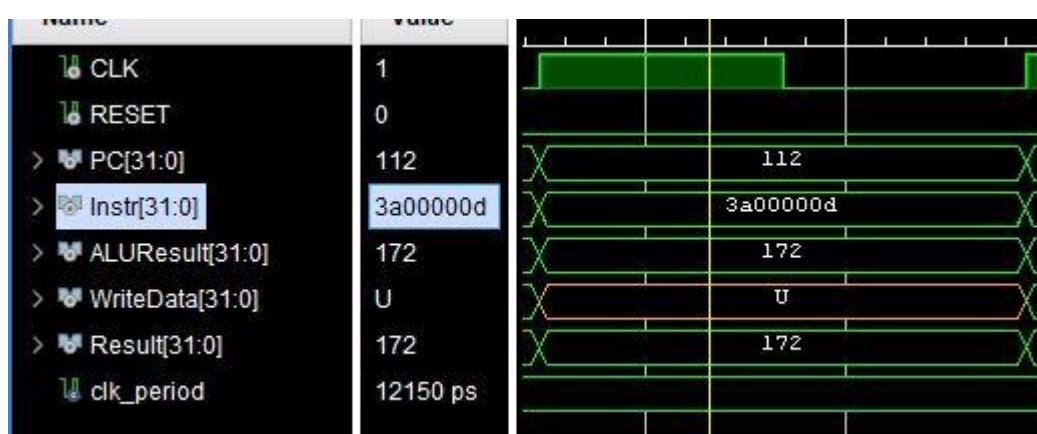
**Εικόνα 135 – Αποτέλεσμα εντολής ADDS R0, R1, -1 σε Implementation Simulation**

### 2.3.28 ΕΝΤΟΛΗ BCC COROUTINE\_CC

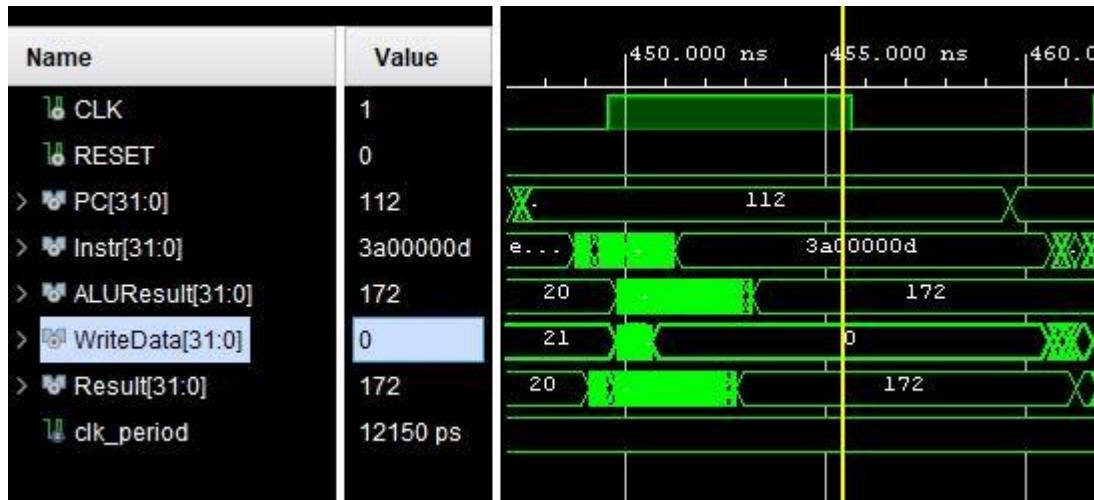
Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό CC ισχύει. Από την εκτέλεση της προηγούμενης εντολής έχουμε C=1, άρα δεν εκτελείται. Η εντολή βρίσκεται στην θέση PC = 108 + 4 = 112. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 3A00000D.

Όσο αφορά τα σήματα ALUResult και Result έχουν την θέση της εντολής η οποία θα εκτελούντα αν ίσχυε το μνημονικό. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 136 – Αποτέλεσμα εντολής BCC COROUTINE\_CC σε Behavioral Simulation**



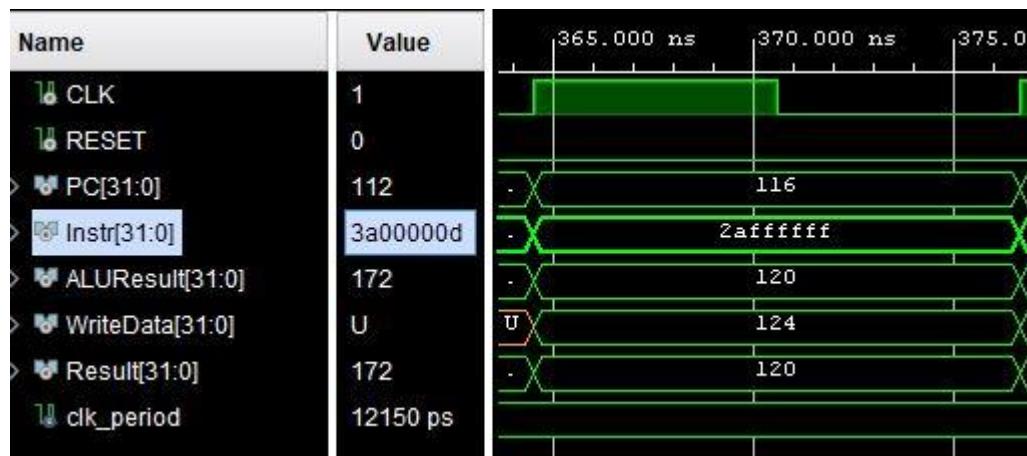
**Εικόνα 137 – Αποτέλεσμα εντολής BCC COROUTINE\_CC σε Implementation Simulation**

### 2.3.29 ΕΝΤΟΛΗ BCS COROUTINE\_CS

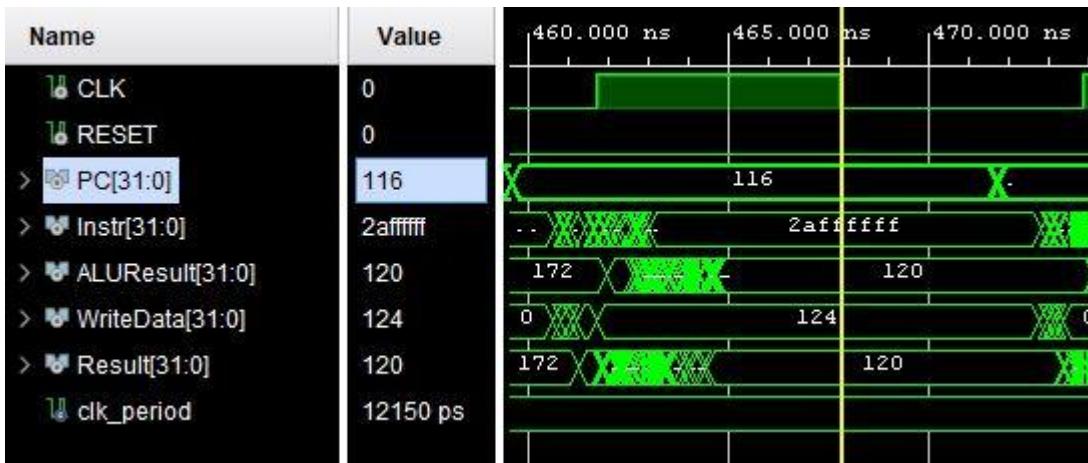
Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό CS ισχύει. Έχουμε C=1, άρα εκτελείται. Η εντολή βρίσκεται στην θέση PC = 112 + 4 = 116.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 2AFFFFFFF. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 120, η οποία είναι η θέση της επόμενης εντολής που θα εκτελεστεί. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 120.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 138 – Αποτέλεσμα εντολής BCS COROUTINE\_CS σε Behavioral Simulation**



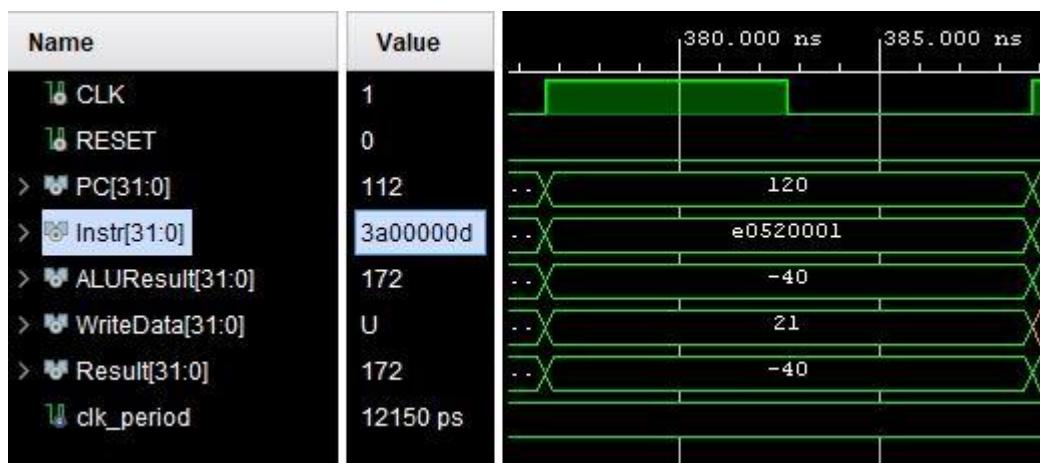
**Εικόνα 139 – Αποτέλεσμα εντολής BCS COROUTINE\_CS σε Implementation Simulation**

### 2.3.30 ΕΝΤΟΛΗ SUBS R0, R2, R1

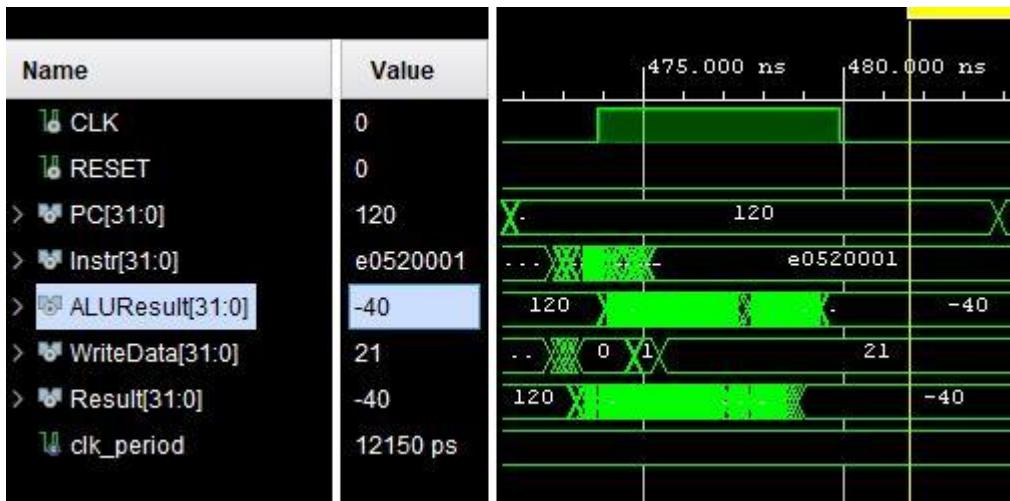
Η εντολή SUB πραγματοποιεί αφαίρεση μεταξύ του περιεχομένου του καταχωρητή R2 και του περιεχομένου του καταχωρητή R1. Δεδομένου ότι ο καταχωρητής R2 περιέχει την τιμή -19 και ο καταχωρητής R1 την τιμή 21 το αποτέλεσμα της πράξης είναι  $-19 - 21 = -40$ . Η συγκεκριμένη εντολή ενεργοποιεί τις σημαία C και N. Ο μετρητής προγράμματος έχει την τιμη  $PC = 116 + 4 = 120$ .

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E0520001. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή -40, που προκύπτει από την αφαίρεση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με -40.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 140 – Αποτέλεσμα εντολής SUBS R0, R2, R1 σε Behavioral Simulation**



**Εικόνα 141 – Αποτέλεσμα εντολής SUBS R0, R2, R1 σε Implementation Simulation**

### 2.3.31 ΕΝΤΟΛΗ BPL COROUTINE\_PL

Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό PL ισχύει. Από την εκτέλεση της προηγούμενης εντολής έχουμε C=1 και N=1, άρα δεν εκτελείται λόγω της σημαίας N. Η εντολή βρίσκεται στην θέση PC = 120 + 4 = 124.

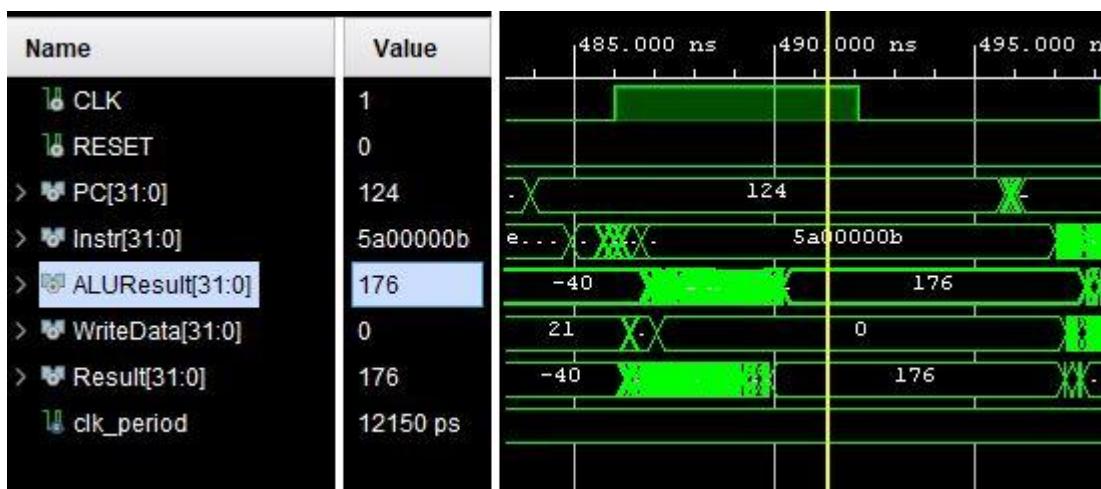
Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E0520001.

Όσο αφορά τα σήματα ALUResult και Result έχουν την θέση της εντολής η οποία θα εκτελούνται αν ίσχυε το μνημονικό. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 142 – Αποτέλεσμα εντολής BPL COROUTINE\_PL σε Behavioral Simulation**



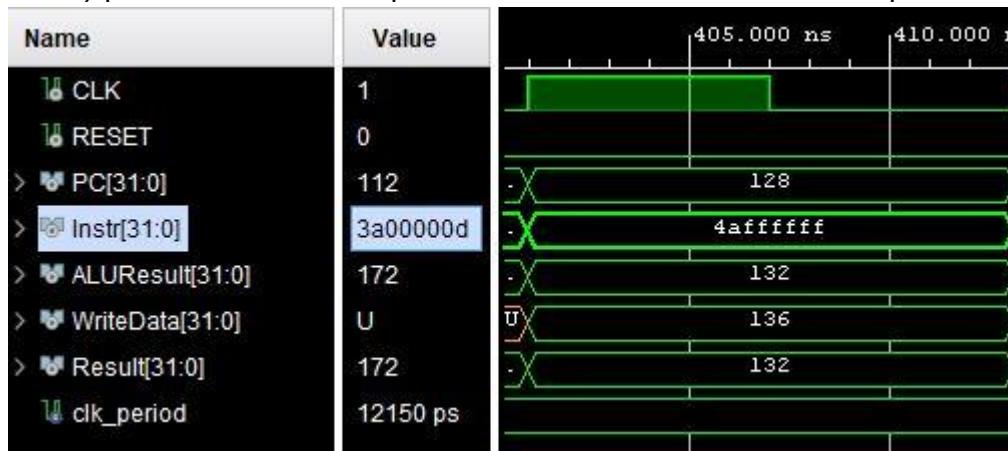
**Εικόνα 143 – Αποτέλεσμα εντολής BPL COROUTINE\_PL σε Implementation Simulation**

### 2.3.32 ΕΝΤΟΛΗ BMI COROUTINE\_MI

Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό MI ισχύει. Έχουμε C=1 και N=1, άρα εκτελείται λόγω της σημαίας N. Η εντολή βρίσκεται στην θέση PC = 124 + 4 = 128.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 4AFFFFFFF. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 132, η οποία είναι η θέση της επόμενης εντολής που θα εκτελεστεί. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 132.

Στις επόμενες εικόνες φαίνονται τα αποτέλεσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 144 – Αποτέλεσμα εντολής BMI COROUTINE\_MI σε Behavioral Simulation**



**Εικόνα 145 – Αποτέλεσμα εντολής BMI COROUTINE\_MI σε Implementation Simulation**

### 2.3.33 ΕΝΤΟΛΗ SUBS R0, R1, 21

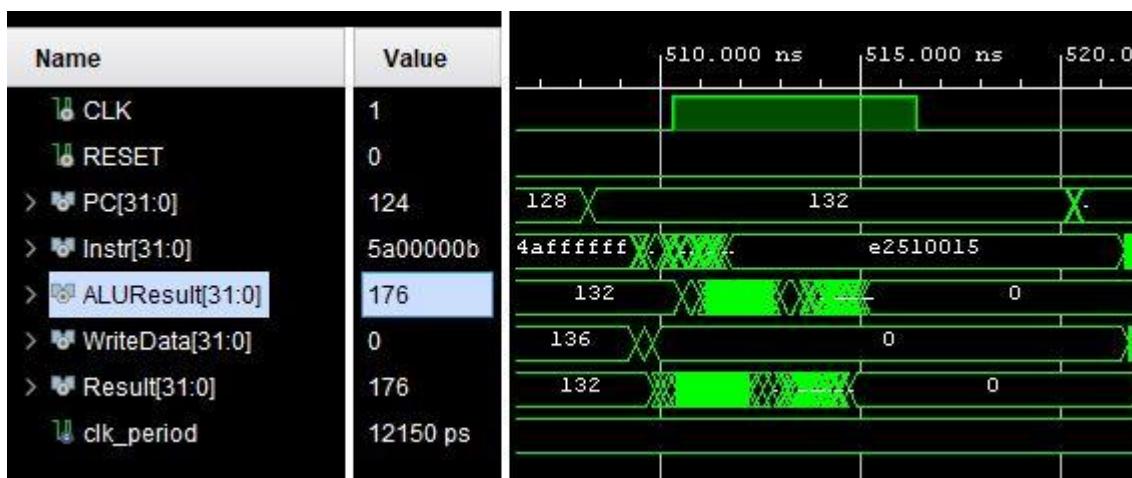
Η εντολή SUB πραγματοποιεί αφαιρέση μεταξύ του περιεχομένου του καταχωρητή R1 και της σταθερής τιμής 21. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης είναι  $21 - 21 = 0$ . Η συγκεκριμένη εντολή ενεργοποιεί τις σημαία C και Z. Ο μετρητής προγράμματος έχει την τιμή PC = 128 + 4 = 132.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι E2510015. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 0, που προκύπτει από την αφαιρέση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 0.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



Εικόνα 146 – Αποτέλεσμα εντολής SUBS R0, R1, 21 σε Behavioral Simulation



Εικόνα 147 – Αποτέλεσμα εντολής SUBS R0, R1, 21 σε Implementation Simulation

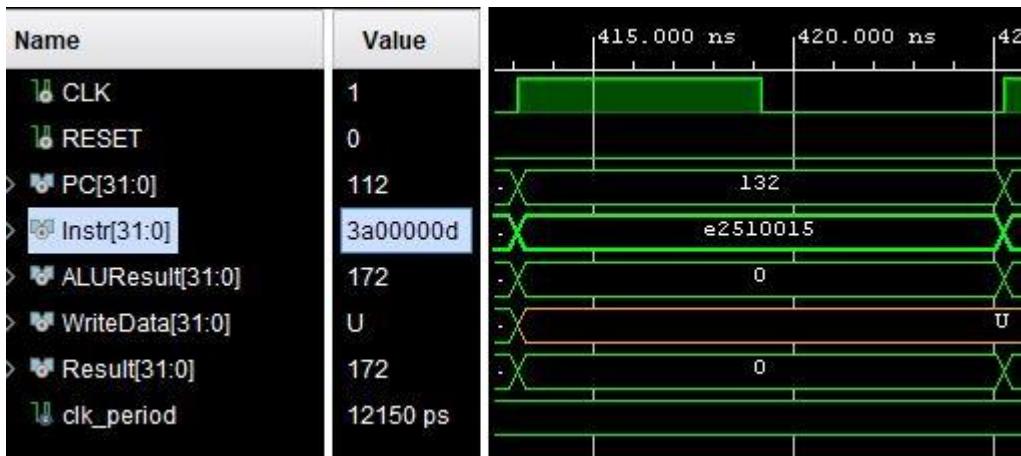
### 2.3.34 ΕΝΤΟΛΗ BVS COROUTINE\_VS

Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό VS ισχύει. Από την εκτέλεση της προηγούμενης εντολής έχουμε C=1 και Z=1, άρα δεν εκτελείται λόγω της σημαίας V η οποία είναι μηδεν. Η εντολή βρίσκεται στην θέση PC = 132 + 4 = 136.

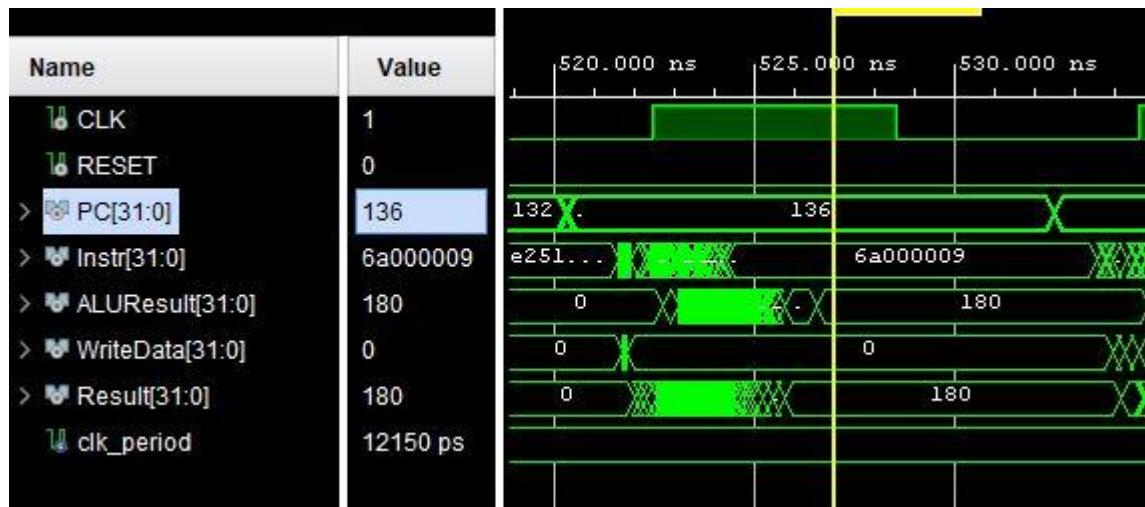
Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 6A000009.

Όσο αφορά τα σήματα ALUResult και Result έχουν την θέση της εντολής η οποία θα εκτελούντα αν ισχυε το μνημονικό. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 148 – Αποτέλεσμα εντολής BVS COROUTINE\_VS σε Behavioral Simulation**



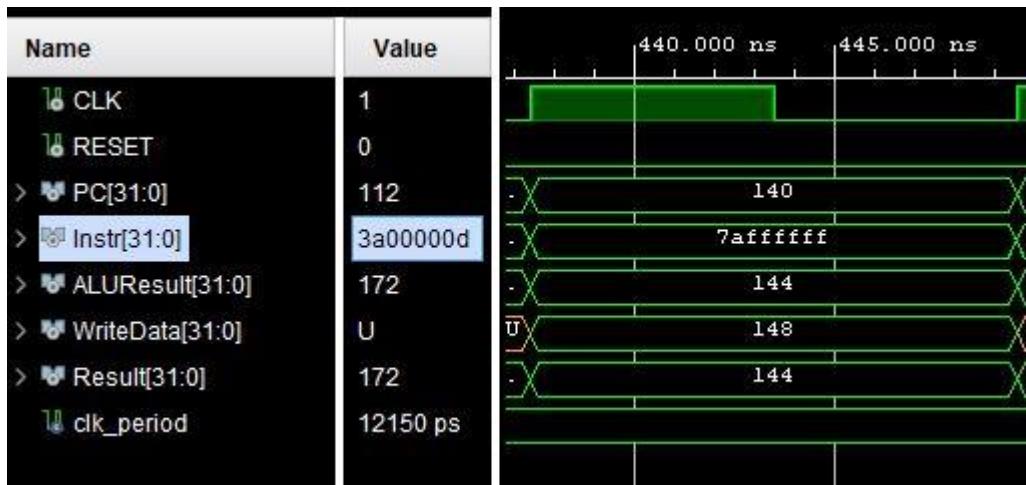
**Εικόνα 149 – Αποτέλεσμα εντολής BVS COROUTINE\_VS σε Implementation Simulation**

### 2.3.35 ΕΝΤΟΛΗ BVC COROUTINE\_VC

Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό VC ισχύει. Έχουμε C=1 και Z=1, άρα εκτελείται λόγω V=0. Η εντολή βρίσκεται στην θέση PC = 136 + 4 = 140.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 7AFFFFFF. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 144, η οποία είναι η θέση της επόμενης εντολής που θα εκτελεστεί. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 144.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 150 – Αποτέλεσμα εντολής BVC COROUTINE VC σε Behavioral Simulation**



**Εικόνα 151 – Αποτέλεσμα εντολής BVC COROUTINE VC σε Implementation Simulation**

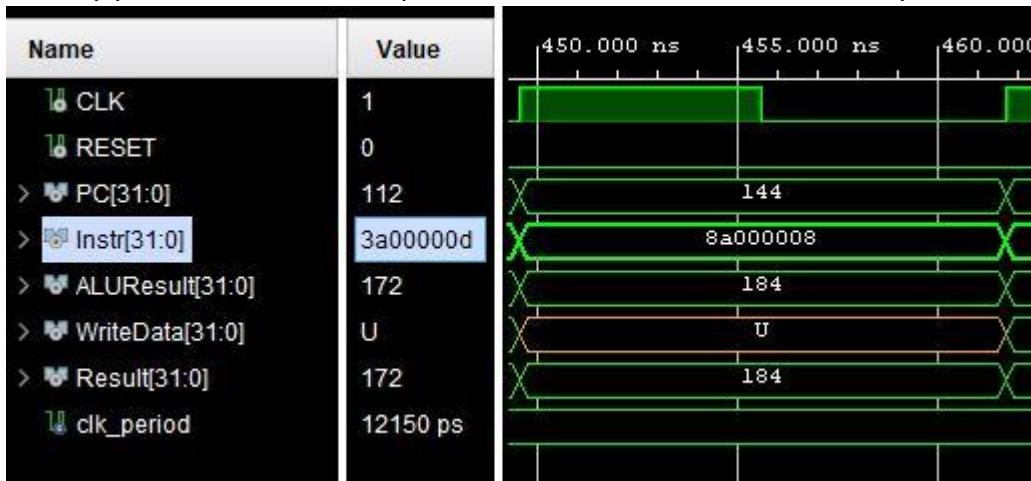
### 2.3.36 ΕΝΤΟΛΗ BLS COROUTINE\_HI

Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό HI ισχύει δηλαδή όταν ισχύει ZC. Οι σημαίες είναι C=1 και Z=1, άρα δεν εκτελείται λόγω της σημαίας Z η οποία είναι ένα. Η εντολή βρίσκεται στην θέση PC = 140 + 4 = 144.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 8A000008.

Όσο αφορά τα σήματα ALUResult και Result έχουν την θέση της εντολής η οποία θα εκτελούντα αν ίσχυε το μνημονικό. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 152 – Αποτέλεσμα εντολής BLS COROUTINE\_HI σε Behavioral Simulation**



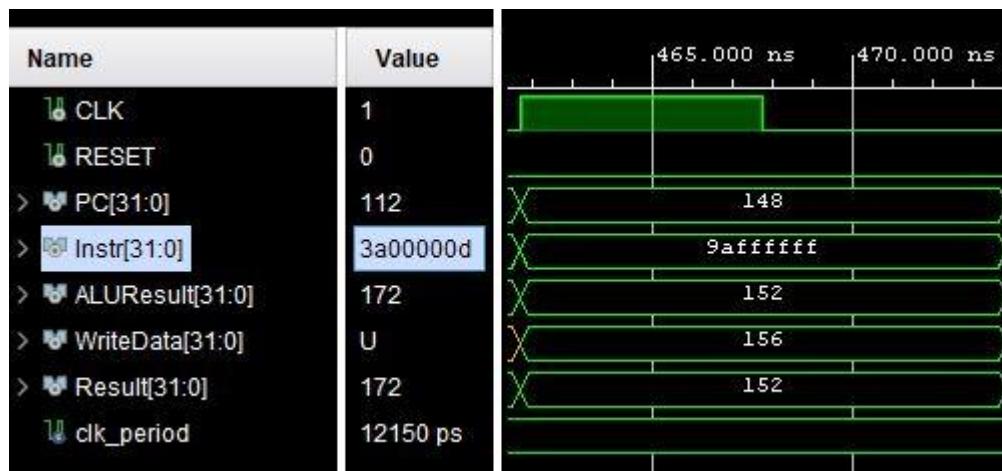
**Εικόνα 153 – Αποτέλεσμα εντολής BLS COROUTINE\_HI σε Implementation Simulation**

### 2.3.37 ΕΝΤΟΛΗ BHI COROUTINE\_LS

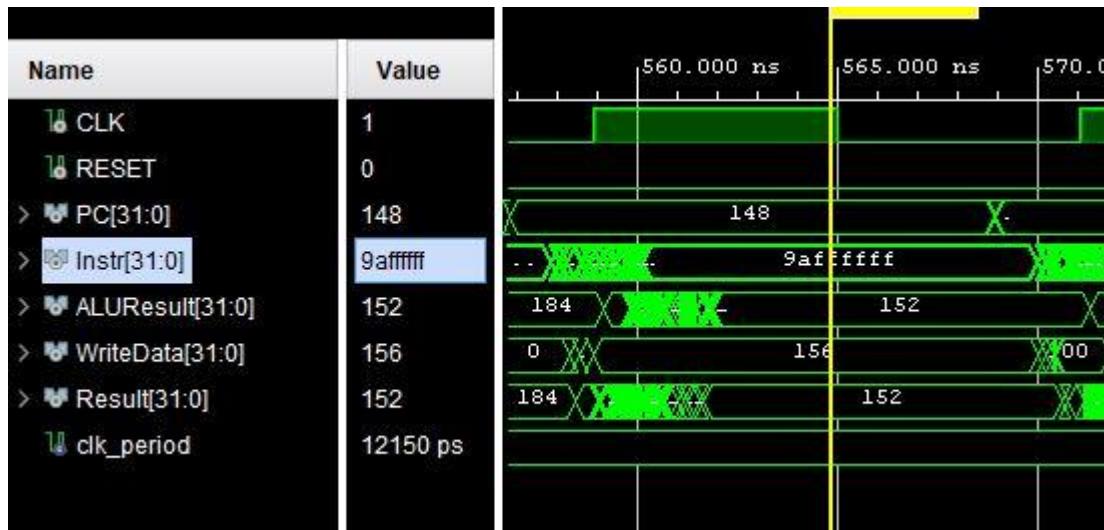
Η συγκεκριμένη εντολή διακλάδωσης εκτελείται μόνο όταν το μνημονικό LS ισχύει δηλαδή όταν ισχύει  $Z+C$ . Έχουμε  $C=1$  και  $Z=1$ , άρα εκτελείται λόγω  $Z$ . Η εντολή βρίσκεται στην θέση  $PC = 144 + 4 = 148$ .

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι 9AFFFFFF. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 152, η οποία είναι η θέση της επόμενης εντολής που θα εκτελεστεί. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 152.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 154 – Αποτέλεσμα εντολής BHI COROUTINE\_LS σε Behavioral Simulation**



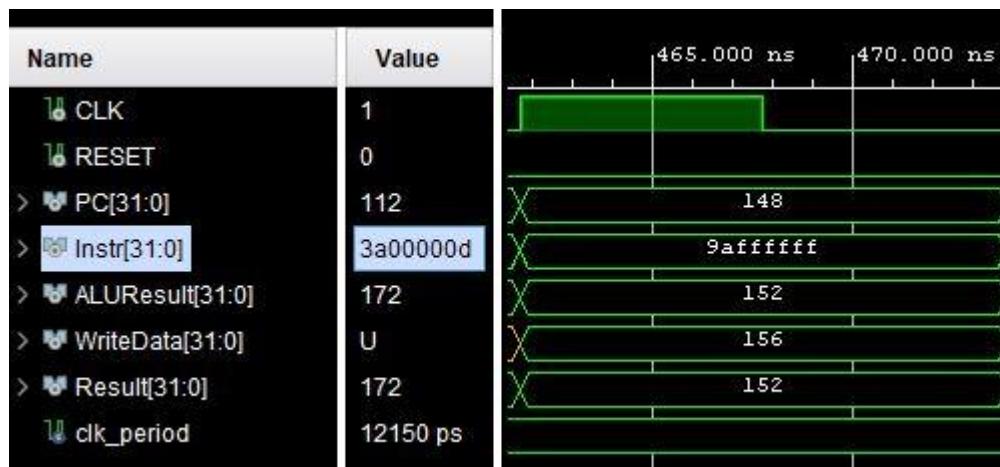
**Εικόνα 155 – Αποτέλεσμα εντολής BHI COROUTINE\_LS σε Implementation Simulation**

### 2.3.38 ΕΝΤΟΛΗ ADDGE R4, R1, 30

Η εντολή ADD πραγματοποιεί πρόσθεση μεταξύ του περιεχομένου του καταχωρητή R1 και της άμεσης τιμής 30. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης είναι  $21 + 30 = 51$ . Η εντολή θα εκτελεστεί μόνο όταν ισχύει το μνημονικό GE, δηλαδή όταν ισχύει ΝΘV. Είναι C=1 και Z=1 άρα N xor V δίνει 1 και η εντολή εκτελέται. Η εντολή βρίσκεται στην θέση PC =  $148 + 4 = 152$ .

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι A281401E. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 51, που προκύπτει από την αφαίρεση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 51.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 156 – Αποτέλεσμα εντολής ADDGE R4, R1, 30 σε Behavioral Simulation**



**Εικόνα 157 – Αποτέλεσμα εντολής ADDGE R4, R1, 30 σε Implementation Simulation**

### 2.3.39 ΕΝΤΟΛΗ ADDGT R4, R1, 30

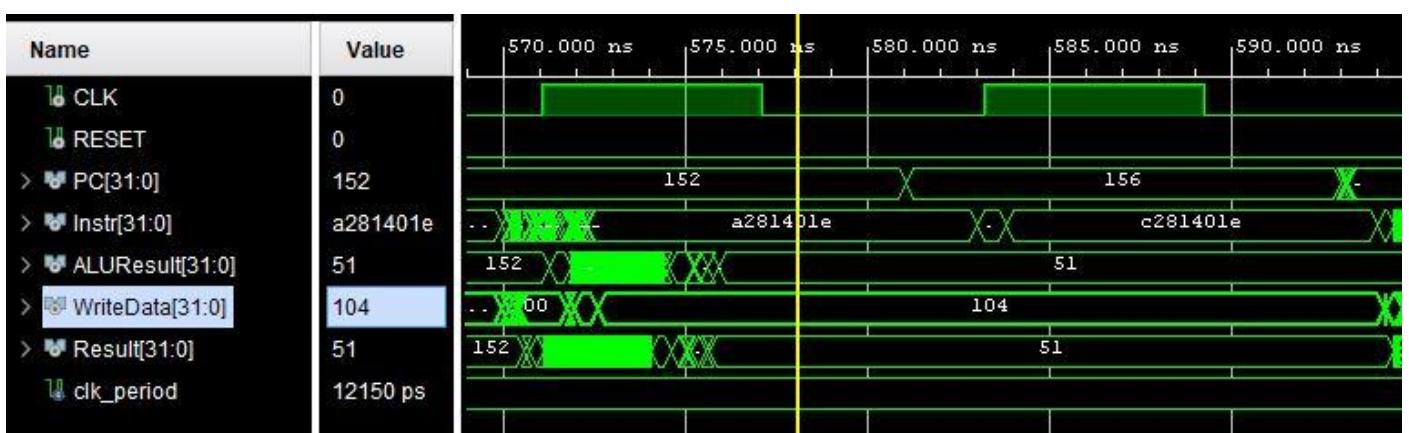
Η εντολή ADD πραγματοποιεί πρόσθιεση μεταξύ του περιεχομένου του καταχωρητή R1 και της άμεσης τιμής 30. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης είναι  $21 + 30 = 51$ . Η εντολή θα εκτελεστεί μόνο όταν ισχύει το μνημονικό GT, δηλαδή όταν ισχύει Ζ ΝΘΒ. Είναι C=1 και Z=1 λόγω της σημαίας Z η εντολή δεν εκτελείται. Η εντολή βρίσκεται στην θέση PC =  $152 + 4 = 156$ .

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι C281401E. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 51, που προκύπτει από την αφαίρεση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 51.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 158 – Αποτέλεσμα εντολής ADDGT R4, R1, 30 σε Behavioral Simulation**



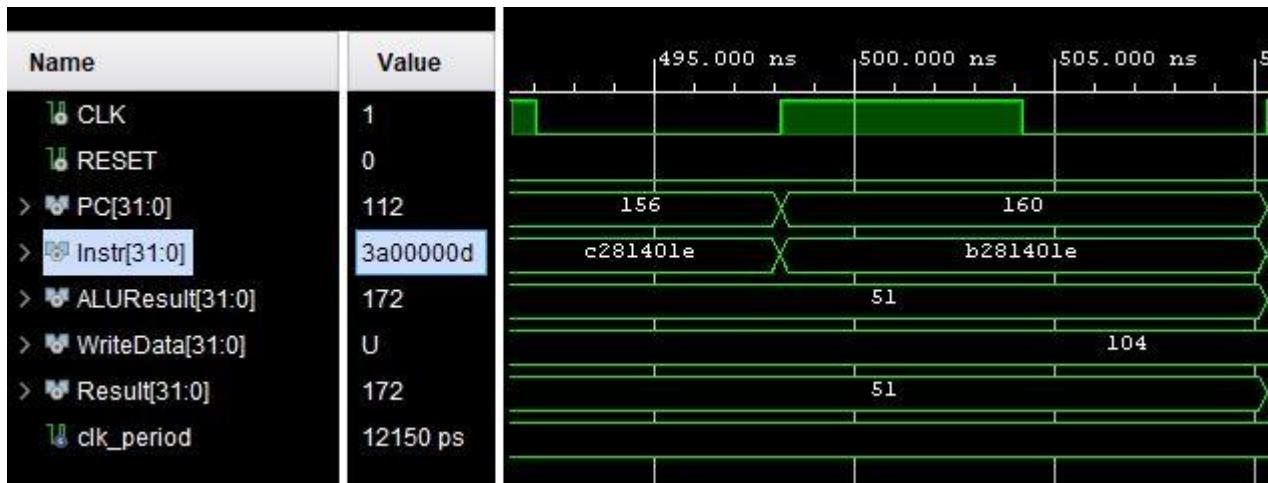
**Εικόνα 159 – Αποτέλεσμα εντολής ADDGT R4, R1, 30 σε Implementation Simulation**

### 2.3.40 ΕΝΤΟΛΗ ADDLT R4, R1, 30

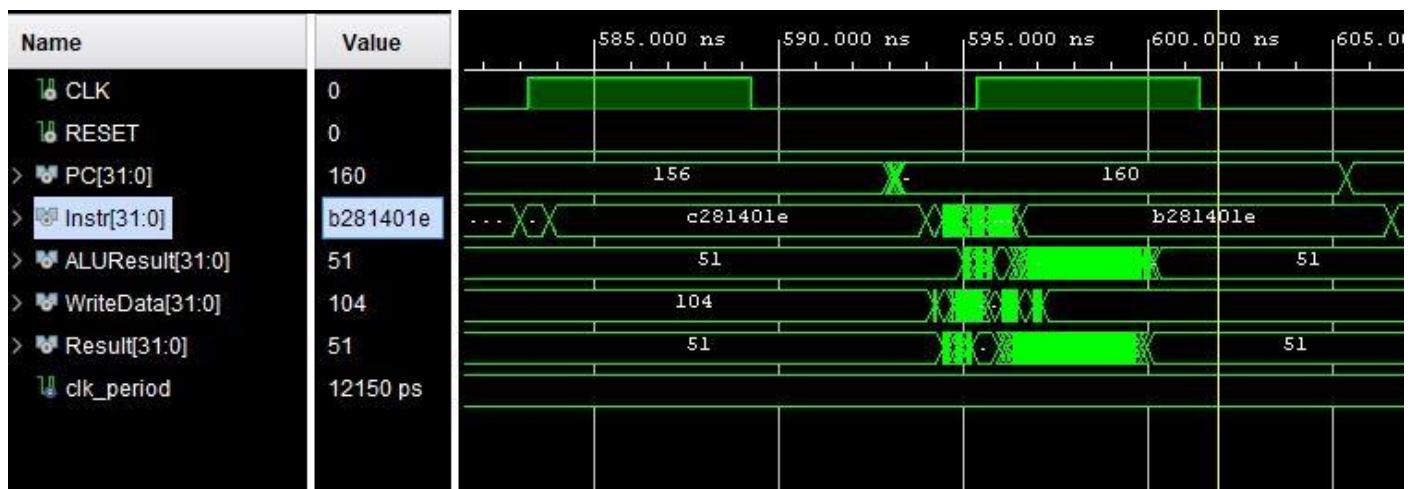
Η εντολή ADD πραγματοποιεί πρόσθεση μεταξύ του περιεχομένου του καταχωρητή R1 και της άμεσης τιμής 30. Δεδομένου ότι ο καταχωρητής R1 περιέχει την τιμή 21 το αποτέλεσμα της πράξης είναι  $21 + 30 = 51$ . Η εντολή θα εκτελεστεί μόνο όταν ισχύει το μνημονικό LT, δηλαδή όταν ισχύει ΝΘV. Είναι C=1 και Z=1 και 0 xor 0 κάνει 0. Η εντολή βρίσκεται στην θέση PC = 156 + 4 = 160.

Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι B281401E. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 51, που προκύπτει από την αφαίρεση. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή της εξόδου Result είναι ίση με το ALUResult, δηλαδή ίση με 51.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



Εικόνα 160 – Αποτέλεσμα εντολής ADDLT R4, R1, 30 σε Behavioral Simulation

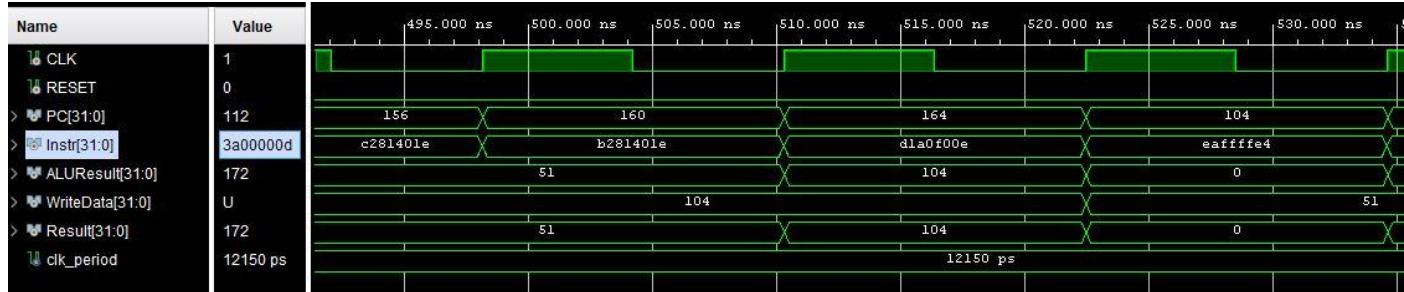


Εικόνα 161 – Αποτέλεσμα εντολής ADDLT R4, R1, 30 σε Implementation Simulation

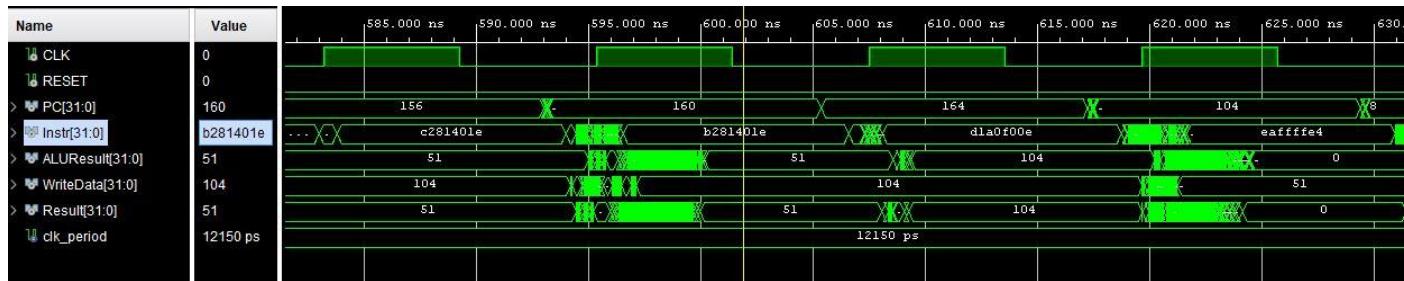
### 2.3.41 ΕΝΤΟΛΗ MOVLE PC, R14

Η εντολή αυτή πραγματοποιεί τη μεταφορά της τιμής του R14 = 104 στον καταχωρητή PC. Η εντολή βρίσκεται στην θέση PC = 160 + 4 = 164. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι D1A0F00E. Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 104 δηλαδή το περιεχόμενο του ALUResult. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή εξόδου Result είναι ίση με 104, καθώς η διαδρομή που ακολουθείται περνά από την ALU και όχι από τη μνήμη δεδομένων. Με αυτή την εντολή μεταφερόμαστε στην επόμενη από αυτή της εντολής BL που εκτελέστηκε πιο πριν και η οποία βρίσκεται στη θέση PC=104.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation.



**Εικόνα 162 – Αποτέλεσμα εντολής MOVLE PC, R14 σε Behavioral Simulation**



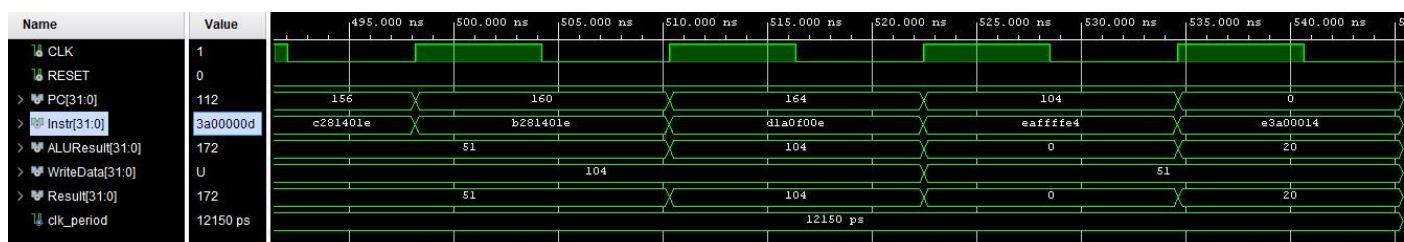
**Εικόνα 163 – Αποτέλεσμα εντολής MOVLE PC, R14 σε Implementation Simulation**

### 2.3.42 ΕΝΤΟΛΗ B MAIN

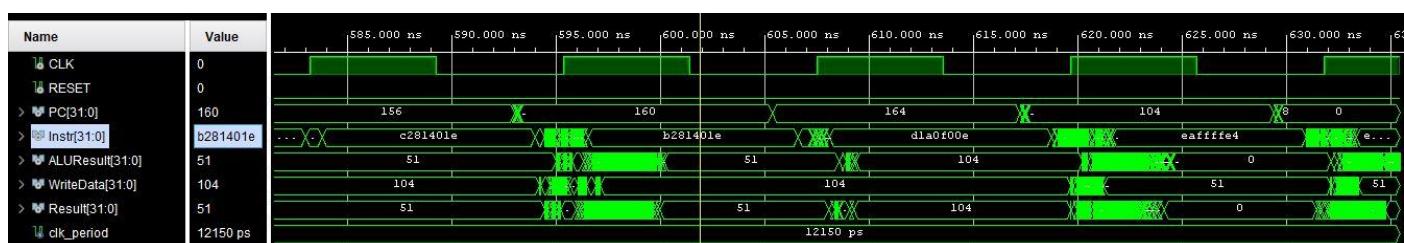
Η συγκεκριμένη εντολή είναι εντολή διακλάδωσης που μας οδηγεί πάλι στην αρχή του προγράμματος. Η τιμή του PC προέρχεται από την προηγούμενη εντολή και είναι 104. Ο κωδικός της εντολής σε δεκαεξαδική μορφή είναι EAFFFFE4.

Όσον αφορά τα σήματα ελέγχου, το αναμενόμενο αποτέλεσμα στην έξοδο ALUResult είναι η τιμή 0 δηλαδη το περιεχόμενο του ALUResult. Η έξοδος WriteData είναι αδιάφορη δεδομένου ότι δε γίνεται εγγραφή στη μνήμη. Η τιμή εξοδου Result είναι ίση με 0.

Στις επόμενες εικόνες φαίνονται τα αποτελέσματα των Behavioral Simulation και Implementation Simulation. Όπως φαίνεται στην εικόνα εκτελείται προς επαλήθευση και η πρώτη εντολή του προγράμματος.



**Εικόνα 164 – Αποτέλεσμα εντολής B MAIN σε Behavioral Simulation**



**Εικόνα 165 – Αποτέλεσμα εντολής B MAIN σε Implementation Simulation**

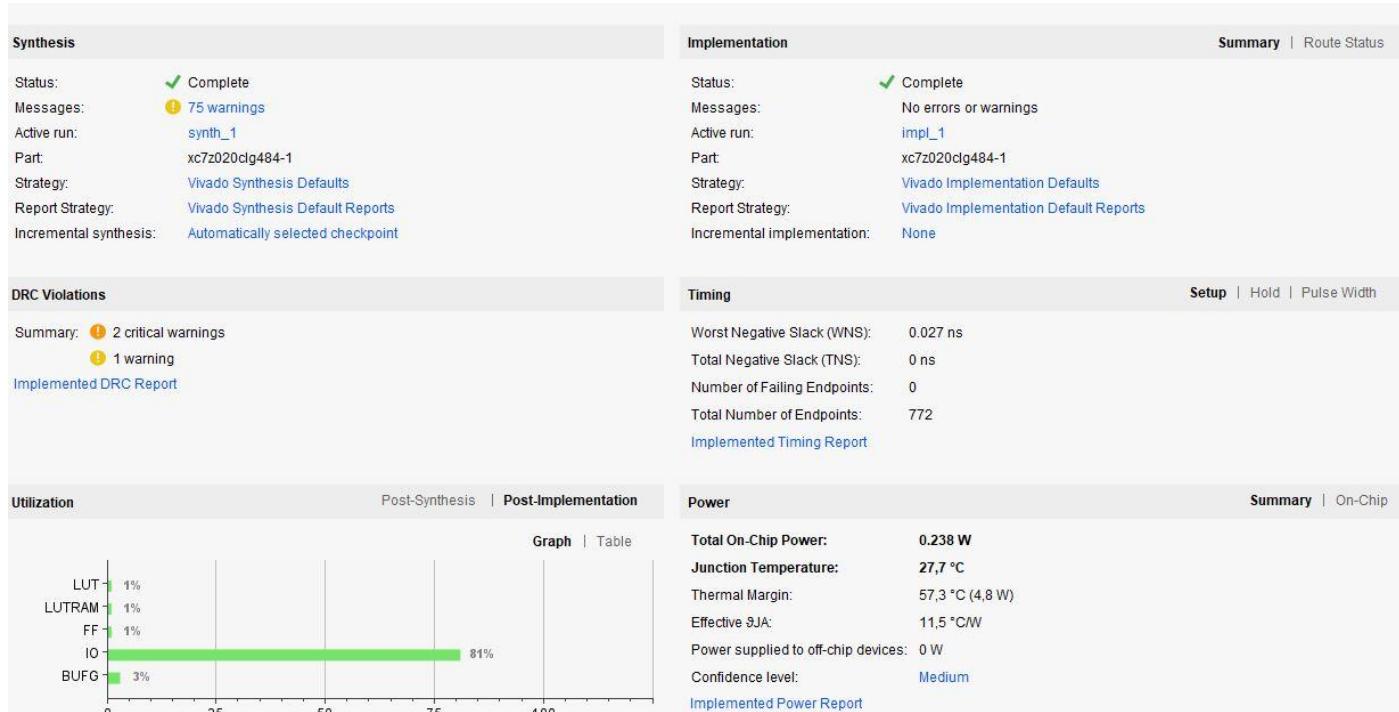
### **2.3.43 ΣΧΟΛΙΑΣΜΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ ΧΡΟΝΙΣΜΟΥ ΕΠΕΞΕΡΓΑΣΤΗ**

Κατά την ανάλυση των διαγραμμάτων χρονισμού του επεξεργαστή παρατηρείται ότι οι διαφορές μεταξύ Behavioral και Implementation προσομοίωσης είναι μικρότερες σε σχέση με αυτές που εμφανίζονται στη μονάδα ALU. Τα σήματα φαίνεται να σταθεροποιούνται σε μικρότερο χρονικό διάστημα και τα φαινόμενα μεταβατικών καταστάσεων (glitches) είναι πολύ λιγότερα. Αυτό ενδεχομένως οφείλεται στο ότι οι περισσότερες λειτουργίες είναι συγχρονισμένες με το ρολόι, καθώς όλες οι υπομονάδες (ALU, αρχείο καταχωρητών, μνήμη εντολών και δεδομένων) λειτουργούν σε καθορισμένα χρονικά στάδια και στο οποίο οι καταχωρητές που παρεμβάλλονται ανάμεσα στα στάδια του datapath απομονώνουν τις καθυστερήσεις, με αποτέλεσμα η συνολική συμπεριφορά να είναι πιο “καθαρή” χρονικά.

Συνολικά, ο επεξεργαστής παρουσιάζει πιο σταθερή χρονική συμπεριφορά, με τις καθυστερήσεις να παραμένουν εντός του κύκλου του ρολογιού (12.150 ns), επιβεβαιώνοντας ότι η υλοποίηση λειτουργεί σωστά και χωρίς χρονικά προβλήματα.

### 3. ΑΝΑΛΥΣΗ ΤΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΥΝΘΕΣΗΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

Το στάδιο του Implementation είναι το τελευταίο στάδιο πριν τη φόρτωση του bitstream στην πλακέτα που έχουμε επιλέξει. Σε αυτό το στάδιο το Vivado παρέχει πλήθος πληροφοριών σχετικά με τους πόρους που χρησιμοποιούνται και καθώς και την κατανάλωση. Στην παρακάτω εικόνα φαίνεται η καρτέλα Project Summary.



**Εικόνα 166 – Καρτέλα Project Summary**

Στο πάνω μέρος βλέπουμε ότι τόσο η Synthesis όσο και το Implementation έχουν ολοκληρωθεί επιτυχώς. Στη φάση Implementation δεν έχει βγει κάποιο σφάλμα ή προειδοποίηση. Στη φάση της Synthesis υπάρχουν προειδοποίήσεις οι οποίες σχετίζονται με το ότι κάποια bits του instruction που είναι είσοδος στο Control δεν χρησιμοποιούνται πράγμα φυσιολογικό. Οι προειδοποίήσεις που βγαίνουν στην καρτέλα DRC έχουν να κάνουν με το ελλιπές αρχείο constraints.

Στην καρτέλα Power παρουσιάζεται η εκτίμηση κατανάλωσης ισχύος μετά την ολοκλήρωση της υλοποίησης του επεξεργαστή στο FPGA. Στην καρτέλα Utilization φαίνονται οι πόροι που έχουν χρησιμοποιηθεί. Κοιτάμε την καρτέλα Post Implementation καθώς σε αυτό το στάδιο το Vivado έχει ολοκληρώσει την τοποθέτηση και τη δρομολόγηση του κυκλώματος πάνω στην επιλεγμένη πλακέτα. Ως αποτέλεσμα, οι τιμές που εμφανίζονται σε αυτή την καρτέλα είναι πιο ακριβείς και ρεαλιστικές, καθώς αντικατοπτρίζουν την πραγματική κατανομή των πόρων στο υλικό.

Η καρτέλα Project Summary είναι συνοπτική. Στο Report Utilization μπορούμε να δούμε πιο αναλυτικά δεδομένα. Στην παρακάτω εικόνα φαίνονται οι πόροι που χρησιμοποιούνται.

Name	^ 1	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Bonded IOB (200)	BUFGCTRL (32)
Processor		763	36	2	1	239	687	76	162	1
> ControlComp (Control)		21	0	2	1	11	21	0	0	0
> DatapathComp (Datapath)		743	36	0	0	235	667	76	0	0

**Εικόνα 167 – Καρτέλα Project Utilization – χρησιμοποιούμενοι πόροι**

Χρησιμοποιούνται 162 Bonded IOBs, που αντιστοιχούν στις εξωτερικές εισόδους/εξόδους του κυκλώματος, αριθμός αναμενόμενος δεδομένου των εισόδων CLK (1 bit), RESET (1 bit) και των εξόδων PC, Instr, ALUResult, WriteData και Result που είναι 32bit. Επίσης χρησιμοποιείται 1 Slice BUFGCTRL για τη διανομή του ρολογιού.

Ο Processor χρησιμοποιεί 763 Slice LUTs από τις διαθέσιμες 53.200, δηλαδή ένα πολύ μικρό ποσοστό των συνολικών πόρων του FPGA. Από αυτούς, οι 687 LUTs χρησιμοποιούνται για λογικές λειτουργίες και οι 76 LUTs αξιοποιούνται ως μνήμη. Οι Slice Registers ανέρχονται σε 36, εκ των οποίων 32 για τον PC (λογικό είναι 32bit) και 4 για τις σημαίες (λογικό είναι 4bit) όπως φαίνεται και στην παρακάτω εικόνα. Αυτοί οι Registers δεν αφορούν όλους τους καταχωρητές αλλά αυτούς που υλοποιούνται με flip – flop που έχουν συντεθεί ως ανεξάρτητα στοιχεία χρονισμού.

Name	Slice LUTs (53200)	Slice Registers (106400)
MoveComp (MoveOper)	25	0
NorrComp (Norr)	7	0
ShiftComp (ShiftOperaf)	159	0
Data_Memory (DM)	32	0
Extender (Extend)	14	0
Instr_Mem (IM)	27	0
Mux2To1_A1_Reg (Mux2T	3	0
Mux2To1_A2_Reg (Mux2T	4	0
Mux2To1_A3_Reg (Mux2T	3	0
Mux2To1_PCN (Mux2To1_	16	0
Mux2To1_Result (Mux2To	32	0
Mux2To1_SrcB (Mux2To1_	32	0
Mux2To1_WD3_Reg (Mux	32	0
PC (PCreg)	0	32
PCPlus4_Comp (INC4_	1	0
PCPlus8_Comp (INC4)	1	0
Register_File (RegFilesW	173	0
Status_Register (StatusR	0	4

**Εικόνα 168 – Κατανομή Slice Registers**

Οι υπόλοιποι καταχωρητές υλοποιούνται με LUT as Memory όπως φαίνεται και στην παρακάτω εικόνα.

Name	Slice LUTs	Slice Registers (106400)	F7 Muxes	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)
Processor	763	36	2	1	239	687	76
DatapathComp (Datapath)	743	36	0	0	235	667	76
ALU_Comp (ALU)	373	0	0	0	139	373	0
Register_File (RegFilesW	173	0	0	0	86	129	44
Mux2To1_WD3_Reg (Mux	32	0	0	0	25	32	0
Mux2To1_SrcB (Mux2To1_	32	0	0	0	29	32	0
Mux2To1_Result (Mux2To	32	0	0	0	25	32	0
Data_Memory (DM)	32	0	0	0	8	0	32

**Εικόνα 169 – Κατανομή LUT as Memory**

Το αρχείο καταχωρητών αποτελείται από 16 καταχωρητές των 32 bit, δηλαδή συνολικά 64 byte αποθηκευτικού χώρου. Αυτό αντιστοιχεί σε 512 bit δεδομένων. Στην υλοποίηση αυτή χρησιμοποιούνται 44 LUTs σε λειτουργία μνήμης (LUT as Memory), καθώς κάθε τέτοιο LUT μπορεί να αποθηκεύσει περισσότερα από ένα bit. Πιο συγκεκριμένα, τα συγκεκριμένα FPGA διαθέτουν LUTs με έξι εισόδους και μία έξοδο, οι οποίες μπορούν να διαμορφωθούν ώστε να λειτουργούν και με πέντε εισόδους και δύο έξοδους, επιτρέποντας την αποθήκευση τουλάχιστον 2 bit ανά στοιχείο. Στην πράξη, το Vivado εφαρμόζει βελτιστοποιήσεις και απλοποίησεις της λογικής Boole, με αποτέλεσμα ορισμένες LUT as Memory να αξιοποιούνται ακόμη πιο αποδοτικά, αποθηκεύοντας περισσότερα bit εντός του ίδιου block. Με αυτόν τον τρόπο, η υλοποίηση του αρχείου καταχωρητών επιτυγχάνεται με χαμηλή κατανάλωση πόρων, χωρίς να απαιτείται επιπλέον block RAM. Ομοίως και με την μνήμη δεδομένων.

Η μονάδα ελέγχου καταλαμβάνει μόλις 21 LUTs, χωρίς να χρησιμοποιεί καθόλου καταχωρητές, γεγονός που δείχνει ότι πρόκειται για μια καθαρά συνδυαστική μονάδα με χαμηλές απαιτήσεις. Αντίθετα, η DatapathComp είναι πιο “βαριά” μονάδα, καθώς χρησιμοποιεί 743 LUTs και όλους τους απαιτούμενους καταχωρητές, αφού περιλαμβάνει στοιχεία όπως το αρχείο καταχωρητών, την ALU και τις μονάδες μνήμης.

Name	Slice LUTs	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)
Processor	763	36	2	1	239	687
DatapathComp (Datapath)	743	36	0	0	235	667
ControlComp (Control)	21	0	2	1	11	21

**Εικόνα 170 – Κατανομή πόρων μεταξύ Datapath και Control**

Συνολικά, η σχεδίαση είναι αρκετά ελαφριά και αποδοτική, χρησιμοποιεί πολύ μικρό ποσοστό των διαθέσιμων πόρων του FPGA και παρουσιάζει σαφή διαχωρισμό ανάμεσα στη λογική ελέγχου και στο datapath. Γενικά, ως ποσοστό χρήσης, μόνο τα I/O έφτασαν κοντά στο μέγιστο όριο, καθώς ο αριθμός των σημάτων εισόδου και εξόδου του συστήματος είναι σχετικά μεγάλος σε σχέση με το μέγεθος της σχεδίασης. Παρ' όλα αυτά, η κατανάλωση των υπόλοιπων πόρων (LUTs, Flip-Flops, LUTRAM) παραμένει σε πολύ χαμηλά επίπεδα.

Name	Slice LUTs	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Bonded IOB (200)	BUFGCTRL (32)
Processor	1.43%	0.03%	<0.01%	<0.01%	1.80%	1.29%	0.44%	81.00%	3.13%
DatapathComp (Datapath)	1.40%	0.03%	0.00%	0.00%	1.77%	1.25%	0.44%	0.00%	0.00%
ControlComp (Control)	0.04%	0.00%	<0.01%	<0.01%	0.08%	0.04%	0.00%	0.00%	0.00%

**Εικόνα 171 – Ποσοστό χρησιμοποιούμενων πόρων**

Η συνολική ισχύς που απορροφάται από το κύκλωμα (Total On-Chip Power) είναι 0,238 W, τιμή αρκετά χαμηλή που υποδηλώνει ότι η σχεδίαση αξιοποιεί αποτελεσματικά τους διαθέσιμους πόρους. Η θερμοκρασία σύνδεσης (Junction Temperature) φτάνει μόλις τους 27,7°C, ένδειξη ότι το σύστημα λειτουργεί σε ασφαλή θερμικά επίπεδα χωρίς υπερθέρμανση. Το Thermal Margin, το οποίο ανέρχεται σε 57,3°C (4,8 W), δείχνει πως υπάρχει σημαντικό περιθώριο πριν επιτευχθεί κρίσιμο θερμικό φορτίο, ενώ η τιμή Effective ΘJA = 11,5 °C/W αποτυπώνει ικανοποιητική θερμική συμπεριφορά και καλή απαγωγή θερμότητας. Επιπλέον, δεν παρατηρείται κατανάλωση ισχύος από εξωτερικά κυκλώματα (0 W off-chip power), καθώς το σύστημα λειτουργεί εξολοκλήρου στο εσωτερικό του FPGA.

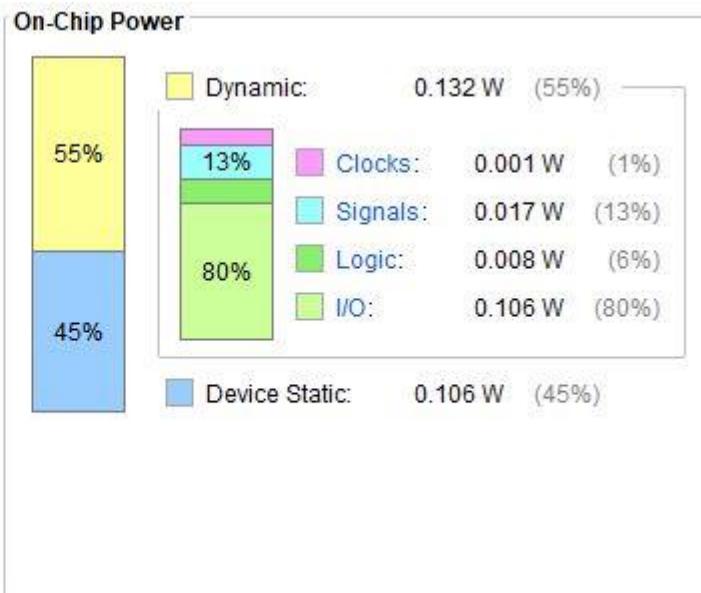
Όσο αφορά την κατανομή της καταναλισκόμενης ενέργειας το 45% (0,106 W) αφορά τη στατική ισχύ (Device Static), δηλαδή την ισχύ που καταναλώνεται ανεξάρτητα από τη λειτουργία του κυκλώματος, λόγω διαρροών και της βασικής λειτουργίας του FPGA. Το υπόλοιπο 55% (0,132 W) είναι δυναμική ισχύς, η οποία σχετίζεται με τη μετάβαση σημάτων και την ενεργό λειτουργία των επιμέρους μονάδων.

Αναλυτικά, η λογική (Logic) συνεισφέρει το 80% της δυναμικής ισχύος, γεγονός αναμενόμενο αφού αποτελεί το κύριο ενεργό τμήμα του datapath. Οι γραμμές σήματος (Signals) συνεισφέρουν 13%, αντιπροσωπεύοντας τη μεταγωγή δεδομένων μεταξύ των επιμέρους μονάδων, ενώ τα ρολόγια (Clocks) έχουν πολύ μικρή συμμετοχή (1%), κάτιο που δείχνει ότι το δίκτυο ρολογιού είναι αποδοτικά σχεδιασμένο. Τέλος, τα I/O καταναλώνουν 6%, καθώς δεν υπάρχει εξωτερική επικοινωνία πέρα από τα βασικά σήματα εισόδου και εξόδου.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

<b>Total On-Chip Power:</b>	0.238 W
<b>Design Power Budget:</b>	Not Specified
<b>Power Budget Margin:</b>	N/A
<b>Junction Temperature:</b>	27,7°C
Thermal Margin:	57,3°C (4,8 W)
Effective θJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity



**Εικόνα 172 – Καταναλισκόμενη ενέργεια**

## 4. ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΑΣ ROR

Η εντολή ROR πραγματοποιεί περιστροφή των bit του τελεστή προς τα δεξιά, κατά αριθμό θέσεων που καθορίζεται από μια ακέραια σταθερά. Με τον τρόπο αυτό, τα bit που "φεύγουν" από τα δεξιά επανεμφανίζονται στα αριστερά, διατηρώντας τον συνολικό αριθμό των bit του τελεστή αμετάβλητο.

### 4.1 ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΗΣ ROR

Η εντολή ROR ως τελεστή προέλευσης χρησιμοποιεί το περιεχόμενο ενός καταχωρητή (Rm), ενώ η ποσότητα περιστροφής καθορίζεται από τα αντίστοιχα bit της εντολής(shamt5), δηλαδή είναι πεδίο του σήματος εντολής. Το αποτέλεσμα της πράξης αποθηκεύεται σε έναν καταχωρητή προορισμού(Rd).

Ανήκει στις εντολές επεξεργασίας δεδομένων και έχει την αντίστοιχη μορφή κωδικοποίησης. Το πεδίο op έχει την τιμή '00', όπως συμβαίνει σε όλες τις εντολές επεξεργασίας δεδομένων, ενώ το πεδίο cmd, που καθορίζει τον ακριβή τύπο της εντολής, έχει την τιμή '1101', όπως και στις υπόλοιπες εντολές ολίσθησης και περιστροφής. Η διαφοροποίηση της εντολής ROR γίνεται μέσα από το πεδίο sh, το οποίο για την περίπτωση αυτή είναι μοναδικό και έχει την τιμή '11'. Η ποσότητα της περιστροφής καθορίζεται από το πεδίο shamt5, το οποίο έχει πλάτος 5 bit και μπορεί να πάρει τιμές από 0 έως 31 — με το 0 να αντιστοιχεί σε καμία περιστροφή και το 31 σε περιστροφή κατά 31 θέσεις για έναν 32-bit τελεστή. Ο καταχωρητής Rn δεν χρησιμοποιείται και έχει σταθερά την τιμή '0000', αφού η εντολή χρησιμοποιεί μόνο έναν τελεστή προέλευσης (Rm) και έναν τελεστή προορισμού (Rd). Το πεδίο cond μπορεί να λάβει οποιαδήποτε συνθήκη, ώστε η εντολή να εκτελείται υπό όρους βάσει των σημαιών του καταχωρητή κατάστασης CPSR. Τέλος, η εντολή μπορεί να ενημερώνει τις σημαίες Z και N: η Z αν το αποτέλεσμα είναι μηδέν, και η N αν το πιο σημαντικό bit (bit 31) του αποτελέσματος είναι ίσο με 1, εφόσον το πεδίο S έχει την τιμή '1'. Στα πλαίσια της εργασίας έχει δηλωθεί ότι για τις εντολές ολίσθησης S=0.

Εντολή ROR																										
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm																
X	X	X	0	0	0	1	1	0	1	X	0	0	0	0	X	X	X	X	X	1	1	0	X	X	X	X

Πίνακας 22 – Εντολη ROR

Από πλευράς υλοποίησης, η εντολή ROR, όπως και οι υπόλοιπες εντολές επεξεργασίας δεδομένων, ολοκληρώνεται σε τέσσερα στάδια (κύκλους ρολογιού).

Στο πρώτο στάδιο γίνεται η ανάγνωση της εντολής από τη μνήμη εντολών και η αποθήκευσή της στον καταχωρητή εντολής.

Στο δεύτερο στάδιο παράγονται τα σήματα ελέγχου για την ανάγνωση των δεδομένων από το αρχείο καταχωρητών. Πιο συγκεκριμένα, διαβάζεται ο καταχωρητής που καθορίζεται από το πεδίο Rm της εντολής, μέσω της θύρας A2, και η τιμή του εμφανίζεται στην έξοδο RD2.

Στο τρίτο στάδιο, η τιμή του τελεστή προέλευσης εισάγεται στη μονάδα ALU, στη θύρα SrcB, και πραγματοποιείται η πράξη της περιστροφής προς τα δεξιά από το αντίστοιχο κύκλωμα ολίσθησης. Η ποσότητα περιστροφής προέρχεται από το πεδίο shamt5 της εντολής.

Τέλος, στο τέταρτο στάδιο, το αποτέλεσμα εγγράφεται στον καταχωρητή προορισμού. Αν ο καταχωρητής προορισμού είναι κάποιος γενικής χρήσης, η εγγραφή γίνεται μέσω των θυρών WD και WA του αρχείου καταχωρητών, ενώ αν είναι ο PC, το αποτέλεσμα της ALU διοχετεύεται στον πολυπλέκτη εισόδου του PC. Αν η εντολή φέρει και το πρόθεμα S, τότε στο ίδιο στάδιο γίνεται ενημέρωση του CPSR με τις νέες τιμές των σημαιών που παράγει η ALU.

### 4.2 ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΜΕ ΤΟ CONTROL UNIT

Η μονάδα ελέγχου δέχεται ως είσοδο την τρέχουσα εντολή και τις τιμές των σημαιών (flags). Ως έξοδος, παράγει τα ακόλουθα σήματα ελέγχου:

- RegSrc: Καθορίζει τις πηγές για τις εισόδους του αρχείου καταχωρητών.
- ALUSrc: Επιλέγει αν η δεύτερη είσοδος της ALU (SrcB) προέρχεται από την έξοδο RD2 του αρχείου καταχωρητών ή από την επεκταμένη άμεση τιμή (extender).
- ImmSrc: Ορίζει τον τύπο επέκτασης για τις άμεσες τιμές (μηδενός για 12-bit σε 32-bit ή πρόσημου για 24-bit σε 32-bit).
- ALUControl: Καθοδηγεί την ALU ως προς την αριθμητική ή λογική πράξη που θα εκτελέσει.
- MemToReg: Επιλέγει αν η τιμή που γράφεται στο αρχείο καταχωρητών προέρχεται από τη μνήμη δεδομένων ή από το αποτέλεσμα της ALU.
- RegWrite: Ενεργοποιεί την εγγραφή στο αρχείο καταχωρητών.
- MemWrite: Ενεργοποιεί την εγγραφή στη μνήμη δεδομένων.
- FlagsWrite: Ενεργοποιεί την ενημέρωση των σημαίων κατάστασης.
- PCSrc: Καθορίζει την πηγή της επόμενης διεύθυνσης εντολής.

### Δομή και Λειτουργία της Μονάδας Ελέγχου

Η μονάδα ελέγχου αποτελείται από τις ακόλουθες υπομονάδες:

- Αποκωδικοποιητής Εντολών: Αναλύει τα διάφορα πεδία της εντολής και παράγει τα σήματα ελέγχου.
- Αποκωδικοποιητής Σημάτων Έγκρισης Εγγραφής: Ελέγχει τα σήματα που επιτρέπουν την εγγραφή στους καταχωρητές, στη μνήμη και στις σημαίες.
- Λογική Επιλογής Διεύθυνσης Επόμενης Εντολής: Υπολογίζει την επόμενη τιμή του PC.
- Λογική Ελέγχου Συνθήκης: Εξετάζει εάν μια εντολή εκτελείται υπό συνθήκη.
- 

### Λειτουργία της Λογικής Ελέγχου Συνθήκης

Η λογική ελέγχου συνθήκης ελέγχει αν η εντολή (όπως η ROR) πρέπει να εκτελεσθεί βάσει της συνθήκης που ορίζεται στα bits 31-28 της εντολής και των τρεχουσών σημαίων. Εάν η συνθήκη δεν ικανοποιείται, παράγεται το σήμα CondEx = 1. Αυτό το σήμα συνδέεται με πύλες AND, με αποτέλεσμα τα σήματα MemToReg, RegWrite, FlagsWrite και PCSrc να μηδενίζονται, ακυρώνοντας έτσι όλες τις εγγραφές και την αλλαγή ροής του προγράμματος.

### Λογική Επιλογής Διεύθυνσης (PCLogic)

Οι είσοδοι της PCLogic περιλαμβάνουν τα πεδία Rd και op[1] της εντολής, καθώς και το εσωτερικό σήμα RegWrite\_in. Η έξοδος είναι το εσωτερικό σήμα PCSrc\_in, το οποίο ελέγχεται από το CondEx\_in. Όταν CondEx\_in = 0, το PCSrc\_in γίνεται 0. Για εντολές επεξεργασίας δεδομένων, το op[1] είναι 0. Από τον πίνακα αληθείας προκύπτει ότι για τέτοιες εντολές (συμπεριλαμβανομένης της ROR), το PCSrc\_in γίνεται 1 μόνο όταν Rd = 15 (δηλαδή, όταν ο προορισμός είναι ο καταχωρητής PC).

### Αποκωδικοποιητής Σημάτων Έγκρισης Εγγραφής

Αυτός ο αποκωδικοποιητής χειρίζεται τα σήματα που επιτρέπουν την εγγραφή στο αρχείο καταχωρητών (RegWrite), στη μνήμη δεδομένων (MemWrite) και στον καταχωρητή σημαίων (FlagsWrite). Οι είσοδοι του είναι τα πεδία op, S/L και 1L της εντολής, καθώς και το εσωτερικό σήμα NoWrite\_in. Για εντολές επεξεργασίας δεδομένων (όπως η ROR):

- Το RegWrite\_in είναι πάντα 1 (επιτρέπει εγγραφή στον καταχωρητή)
- Το FlagsWrite\_in είναι 1 μόνο όταν S = 1 (ενημέρωση σημαίων)
- Το MemWrite\_in είναι πάντα 0 (δεν γίνεται εγγραφή στη μνήμη)
- Το NoWrite\_in είναι πάντα 0. Γίνεται 1 μόνο στην εντολή CMP η οποία δεν εγγράφει ούτε στη μνήμη ούτε στους καταχωρητες.

## Αποκωδικοποιητής Εντολών

Ο αποκωδικοποιητής εντολών έχει ως εισόδους τα πεδία op, sh, shamt5 και funct της εντολής. Οι έξοδοι του περιλαμβάνουν τα σήματα RegSrc[1:0], ALUSrc, ImmSrc, ALUControl[1:0], MemToReg και το εσωτερικό σήμα NoWrite\_in. Για εντολές ολίσθησης (shift):

- RegSrc:
  - RegSrc[1] = 0 (χρήση του καταχωρητή Rm)
  - RegSrc[0] = X (το Rn δεν χρησιμοποιείται)
  - RegSrc[2] = 0 (το αποτέλεσμα γράφεται στον Rd)
- ALUSrc: 0 (πηγή είναι καταχωρητής, όχι άμεση τιμή)
- ImmSrc: Αδιάφορο (δεν απαιτείται επέκταση)
- ALUControl: 1111 για εκτέλεση πράξης ROR
- MemToReg: 0 (το αποτέλεσμα προέρχεται από την ALU)
- NoWrite\_in: 0 (επιτρέπεται η εγγραφή στον καταχωρητή προορισμού)