ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Σχεδίαση Ψηφιακών Συστημάτων

Project 1: Μεθοδολογία σχεδίασης επεξεργαστή ενός κύκλου

Τμήμα:	Φυσικής (Μ.Δ.Ε. στον Ηλεκτρονικό Αυτοματισμό)
Ημερομηνία project:	31/03/2022
Διδάσκων:	Αντώνης Πασχάλης, Διονύσης Βασιλόπουλος

Ονοματεπώνυμο:	Ευάγγελος Πετρόπουλος
Α/Μ:	7110132100213
Email:	vaggelispetr@phys.uoa.gr

Περιεχόμενα

Πίνακας Εικόνων

Γενική περιγραφή των στοιχείων και της δομής του επεξεργαστή ενός κύκλου που υλοποιήθηκε.

Περιγραφή του σύνολο των εντολών που έχει υλοποιηθεί.

Στην παρούσα εργασία υλοποιήθηκε με την γλώσσα περιγραφής υλικού VHDL η μικροαρχιτεκτονική ενός απλοποιημένου επεξεργαστή ενός κύκλου, αρχιτεκτονικής ARM η οποία αποτελείται από σταθερού μήκους λέξεις εντολών των 32 bit. Πιο συγκεκριμένα το σύνολο των εντολών που υλοποιήθηκε οι οποίες πραγματοποιούνται και προαιρετικά υπό συνθήκη είναι το εξής : ADD(S), SUB(S), CMP, AND(S), OR(S), XOR(S), MOV(S), MVN (S), LSL(S), LSR(S), ASR(S), ROR(S), LDR, STR, B, BL.

Γενικές πληροφορίες για την αρχιτεκτονική επεξεργαστών ARM

Η αρχιτεκτονική ARM ορίζει εντολές σε γλώσσα μηχανής σταθερής κωδικοποίησης των 32 bit με τρεις κύριες μορφές εντολών:

1. εντολές επεξεργασίας δεδομένων
2. εντολές μνήμης
3. εντολές διακλάδωσης

Η γενική μορφή της εντολής παρουσιάζεται στην παρακάτω εικόνα:

Επεξήγηση των διάφορων πεδίων της εντολής:

1. Πεδίο cond (condition, 4 bit, 31:28) για τη δήλωση των συνθηκών βάσει των οποίων θα πραγματοποιηθεί η εντολή.
    - cond = 1110 για τις εντολές χωρίς συνθήκη
    - cond = 0000 – 1101 για τις εντολές υπό συνθήκη
2. Πεδίο op (opcode, 2 bit, 27:26) για τη δήλωση της μορφής των εντολών
    - op = 00 για τις εντολές επεξεργασίας δεδομένων
    - op = 01 για τις εντολές μνήμης
    - op = 10 για τις εντολές διακλάδωσης
3. Πεδίο funct (function, 6 bit, 25:20) για τη δήλωση της λειτουργίας/πράξης
    - Υποπεδίο I (immediate, 1 bit, 25) για τη δήλωση ύπαρξης άμεσου ή έμμεσου τελεστέου στο πεδίο src2 (I=1 άμεσος τελεστέος)
    - Υποπεδίο cmd (command, 4 bit, 24:21) για τη δήλωσης της πράξης/λειτουργίας
      ADD = 0100, SUB = 0010, CMP = 1010, AND = 0000, ORR = 1100, EOR = 0001
      LSL = LSR = ASR = ROR = 1101, MOV = 1101, MVN = 1111, BIC = 1110
    - Υποπεδίο S (1 bit, 20) για τη ενημέρωση ή μη των σημαιών συνθήκης (με S = 1, η εντολή ενημερώνει τις σημαίες)
4. Πεδίο Rn (4 bit, 19:16) για τη δήλωση του καταχωρητή προέλευσης του πρώτου τελεστέου
5. Πεδίο Rd (4 bit, 15:12) για τη δήλωση του καταχωρητή προορισμού του αποτελέσματος της πράξης
6. Πεδίο Src2 (12 bit, 11:0) για τον προσδιορισμό του δευτέρου τελεστέου προέλευσης το οποίο χωρίζεται σε υποπεδία ανάλογα με την τιμή του πεδίου I:
    - Αν I=1 δηλαδή υπάρχει άμεσος τελεστέος τότε το Πεδίο Src2 (12 bit, 11:0) απαρτίζεται από:

1) Υποπεδίο imm8 (8 bit, 7:0) για την αποθήκευση ενός μη προσημασμένου άμεσου τελεστέου των 8 bit
2) Υποπεδίο rot (4 bit, 11:8) για τη δήλωση της άμεσης ποσότητας περιστροφής προς τα δεξιά του άμεσου τελεστέου των 8 bit

- Αν I=0 δηλαδή υπάρχει άμεσος τελεστέος τότε το Πεδίο Src2 (12 bit, 11:0) απαρτίζεται από:
  1) Υποπεδίο Rm (4 bit, 3:0) για τη δήλωση του καταχωρητή προέλευσης του δευτέρου τελεστέου
  2) Υποπεδίο shamt5 (shift amount, 5 bit, 11:7) για τη δήλωση της σταθερής ποσότητας ολίσθησης (0-31)
  3) Υποπεδίο sh (shift, 2 bit, 6:5) για τη δήλωση του τύπου της ολίσθησης (LSL = 00, LSR = 01, ASR = 10, ROR = 11)

Ο επεξεργαστής που κατασκευάστηκε στα πλαίσια της παρούσας εργασίας αποτελείται από δύο κυρίως μονάδες που επικοινωνούν μεταξύ τους, τη μονάδα ελέγχου (Control Unit ή Controller) και τη διαδρομή δεδομένων (Datapath). Η μονάδα ελέγχου είναι υπεύθυνη για να δίνει οδηγίες στη διαδρομή δεδομένων σχετικά με το πώς να εκτελέσει κάθε εντολή, παράγοντας και συγχρονίζοντας τα κατάλληλα σήματα ελέγχου των διαφόρων υπομονάδων από τις οποίες απαρτίζεται η διαδρομή δεδομένων. Από την άλλη μεριά η διαδρομή δεδομένων χρησιμοποιώντας τα κατάλληλα σήματα ελέγχου από την μονάδα ελέγχου, ώστε να εκτελεί διάφορες λειτουργίες (εκτέλεση πράξεων, αποθήκευση και ανάκτηση δεδομένων) με λέξεις δεδομένων στις διάφορες υπομονάδες διευθετώντας κατάλληλα τη διαδρομή των δεδομένων. Επειδή για την κατασκευή της μικροαρχιτεκτονική του επεξεργαστή ενός κύκλου ακολουθήθηκε structural design πρέπει να επισημανθεί πως οι δύο κυρίως μονάδες του ελέγχου και της διαδρομής δεδομένων αποτελούνται από κάποια πιο ψηφιακά δομικά στοιχεία (components) εσωτερικά τα οποία πιο πάνω αναφέρθηκαν ως υπομονάδες. Στην συνέχεια θα αναλυθούν διεξοδικά τα ψηφιακά δομικά στοιχεία που χρησιμοποιήθηκαν τόσο από τη μονάδα ελέγχου όσο και την διαδρομή δεδομένων κατά ιεραρχική σχεδιαστική προσέγγιση bottom-up. Τέλος πρέπει να αναφερθεί πως το ανώτερο ιεραρχικά επίπεδο του επεξεργαστή (top module) θα περιέχει τόσο τη μονάδα ελέγχου όσο και την διαδρομή δεδομένων.

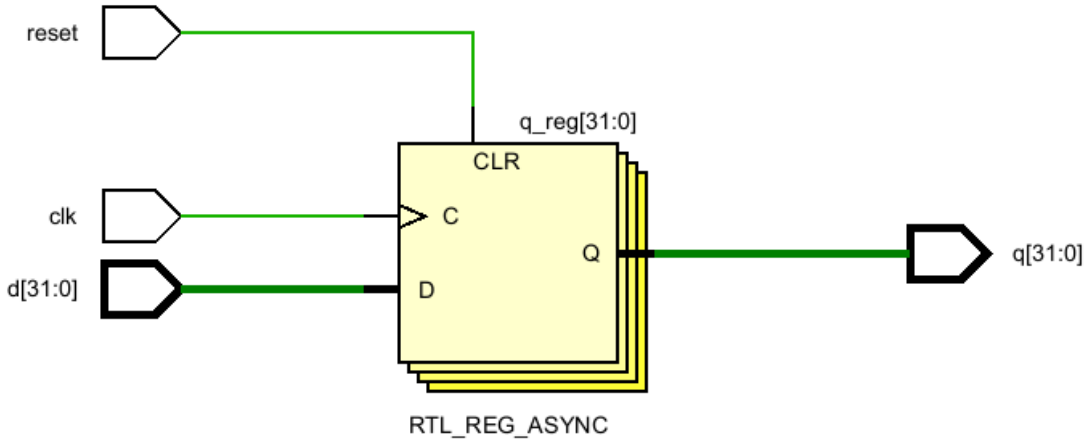
Περιγραφή των ψηφιακών δομικών στοιχείων της διαδρομής δεδομένων (datapath)

Όπως προαναφέρθηκε η διαδρομή δεδομένων αποτελείται από κάποια ψηφιακά δομικά στοιχεία (components) τα οποία ευθέως αμέσως θα σκιαγραφηθούν και στην συνέχεια θα δοθεί ο κώδικας στην γλώσσα περιγραφής υλικού VHDL για την υλοποίηση τους:

1) Καταχωρητής μετρητή προγράμματος (program counter - PC) των 32 bit : ο οποίος δεν είναι κάτι άλλο από έναν καταχωρητή 32 bit, που στην είσοδο του (d) εισέρχεται η διεύθυνση (32 bit) της επόμενης εντολής στην Instruction Memory και στην έξοδο του (q) εξέρχεται η διεύθυνση (32 bit) της τρέχουσας εντολής στην Instruction Memory. Ως καταχωρητής η τιμή του από την είσοδο στην έξοδο περνά κατά την ανερχόμενη ακμή του (1) bit σήματος ελέγχου εισόδου clock (clk) ενώ η έξοδος του αρχικοποιείται στην τιμή 0 με το (1) bit σήματος αρχικοποίησης εξόδου reset.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity flopr is -- flip-flop with synchronous reset
generic(width: positive := 32);
port(clk, reset: in STD_LOGIC;
d: in STD_LOGIC_VECTOR(width-1 downto 0);
q: out STD_LOGIC_VECTOR(width-1 downto 0));
end;


architecture Behavioral of flopr is

begin
process(clk, reset) begin
if reset='1' then q <= (others => '0');
elsif(clk = '1' and clk'event) then
q <= d;
end if;
end process;


end Behavioral;
```

Εικόνα 1: Κώδικας καταχωρητή μετρητή προγράμματος

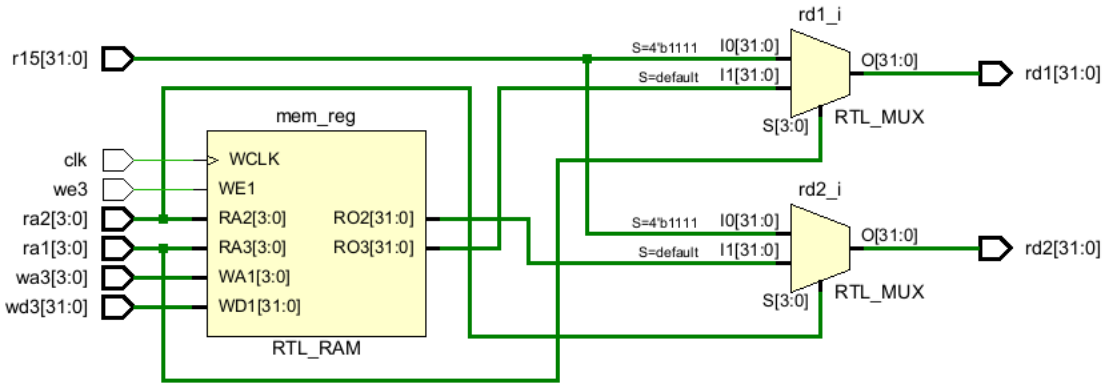


Εικόνα 2: RTL Schematic καταχωρητή μετρητή προγράμματος

2) Αρχείο καταχωρητών (register file - RF): το οποίο περιέχει 15 καταχωρητές των 32 bit (R0–R14). Επίσης διαθέτει δύο θύρες A1(4 bit) και A2(4 bit) μέσω των οποίων προσδιορίζεται από την έξοδο ποιανού καταχωρητή θα αναγνωστούν δεδομένα από τις θύρες RD1 (32 bit) και RD2 (32 bit) αντίστοιχα. Ακόμη υπάρχει μια θύρα A3(4 bit) μέσω της οποίας προσδιορίζεται στην είσοδο ποιανού καταχωρητή θα σταλούν δεδομένα από την θύρα WD3 (32 bit) για εγγραφή στον καταχωρητή αυτόν. Επειδή όπως προαναφέρθηκε το αρχείο καταχωρητών περιέχει ουσιαστικά 15 καταχωρητές των 32 bit είναι απαραίτητη η ύπαρξη ενός σήματος ελέγχου εισόδου clock (clk) (1 bit) και ενός σήματος ελέγχου εγγραφής wright enable (WE3) (1 bit) έτσι ώστε να γίνεται σύγχρονη εγγραφή των δεδομένων από την θύρα WD3 (32 bit) στον κατάλληλο καταχωρητή που υποδεικνύει η θύρα A3(4 bit) κατά την ανερχόμενη ακμή του clock και μόνο όταν το wright enable έχει την τιμή 1. Αντίθετα η ανάγνωση δεδομένων μέσω των θυρών RD1 (32 bit) και RD2 (32 bit) από τους κατάλληλους καταχωρητές που υποδεικνύουν οι θύρες A1(4 bit) και A2(4 bit) γίνεται ασύγχρονα χωρίς την έγκριση κάποιου σήματος. Επιπλέον επειδή και καταχωρητής μετρητή προγράμματος θεωρείται τμήμα του αρχείου καταχωρητών μέσω της θύρας R15 μεταφέρεται την τιμή μετρητή προγράμματος προστιθέμενη με τον αριθμό ‘8’ (PC+8) στις θύρες RD1 ή RD2, όταν στις θύρες A1 ή A2 καταφθάνει η τιμή ‘15’.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity regfile is -- three-port register file
port(clk: in STD_LOGIC;
we3: in STD_LOGIC;
ra1, ra2, wa3: in STD_LOGIC_VECTOR(3 downto 0);
wd3, r15: in STD_LOGIC_VECTOR(31 downto 0);
rd1, rd2: out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture Behavioral of regfile is
type ramtype is array (15 downto 0) of STD_LOGIC_VECTOR(31 downto 0);
signal mem: ramtype;
begin
process(clk) begin
if rising_edge(clk) then
if we3 = '1' then mem(to_integer(unsigned(wa3))) <= wd3;
end if;
end if;
end process;
process(clk,we3,ra1, ra2, wa3,wd3, r15) begin
if (to_integer(unsigned(ra1)) = 15) then rd1 <= r15;
else rd1 <= mem(to_integer(unsigned(ra1)));
end if;
if (to_integer(unsigned(ra2)) = 15) then rd2 <= r15;
else rd2 <= mem(to_integer(unsigned(ra2)));
end if;
end process;

end Behavioral;
```

Εικόνα 3: Κώδικας Register file

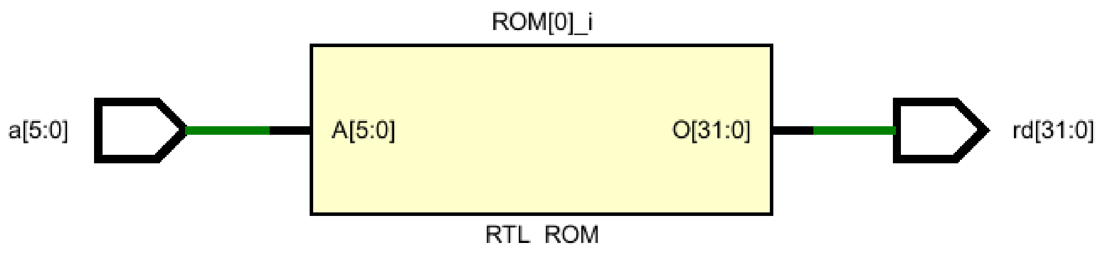


Εικόνα 4: RTL Schematic Register file

3) Μνήμη εντολών (instruction memory - IM) που περιέχει 2^6 λέξεις με 32 bit η κάθε λέξη (η κάθε λέξη είναι ουσιαστικά μία εντολή) : η οποία δέχεται ως είσοδο (A) την διεύθυνση 6 bit όπου βρίσκεται αποθηκευμένη εσωτερικά στην μνήμη εντολών η τρέχουσα εντολή και ότι περιέχει αυτή η διεύθυνση, δηλαδή ουσιαστικά την τρέχουσα εντολή των 32 bit το μεταφέρει στην μοναδική έξοδο read data (RD).

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity imem is -- instruction memory
generic(
    N : positive := 6; --address length
    M : positive := 32); --data word length
port(a: in STD_LOGIC_VECTOR(N-1 downto 0);
rd: out STD_LOGIC_VECTOR(M-1 downto 0));
end imem;

architecture Behavioral of imem is
    type ROM_array is array (0 to (2**N)-1) of STD_LOGIC_VECTOR(M-1 downto 0);
    constant ROM : ROM_array :=(
    X"E3A00000",  -- START: MOV R0, #0
    X"E3E01000",  --        MVN R1, #0
    X"E580100F",  --        STR R1, [R0,#15]
    X"E590200F",  --        LDR R2, [R0,#15]
    X"E1A02202",  --        LSL R2, R2, #4
    X"E1A02242",  --        ASR R2, R2, #4
    X"E1A02222",  --        LSR R2, R2, #4
    X"E1A02262",  --        ROR R2, R2, #4
    X"E580100E",  --        STR R1, [R0,#14]
    X"E590300E",  --        LDR R3, [R0,#14]
    X"E1822003",  --        ORR R2, R2, R3
    X"E2804002",  --        ADD R4, R0, #2
    X"E2845001",  --        SUB R5, R4, #-1
    X"E1510003",  --        CMP R1, R3
    X"00456004",  --        SUBEQ R6, R5, R4
    X"10846005",  --        ADDNE R6, R4, R5
    X"E1A00006",  --        MOV R0, R6
    X"E2557005",  --        SUBS R7, R5, #5
    X"41A08001",  --        MOVMI R8, R1
    X"51A08007",  --        MOVPL R8, R7
    X"E1A00000",  --        NOP

     X"EB000000",  --        BL ROUTINE
     X"E3A00001",  --        MOV R0, #1
     X"E1A0D00E",  --        ROUTINE: MOV R13, R14
     X"EAFFFFE2",  --        B START
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000");

 begin
 rd <= ROM(to_integer(unsigned(a)));
 end Behavioral;
```

Εικόνα 5: Κώδικας Instruction memory



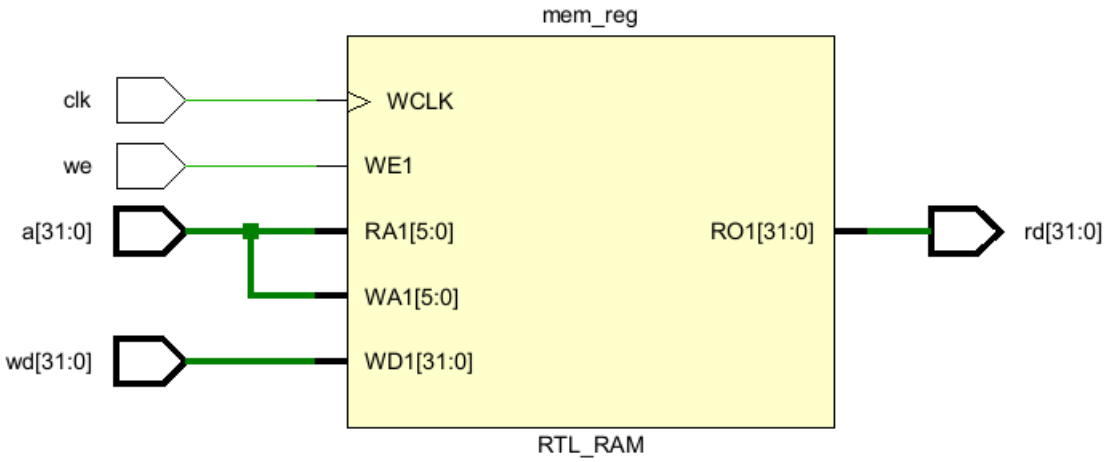
Εικόνα 6: RTL Schematic Instruction memory

4) Μνήμη δεδομένων (data memory - DM) των 2^6 λέξεων με 32 bit η κάθε λέξη(η κάθε λέξη είναι ουσιαστικά ένα δεδομένο) : η οποία δέχεται ως είσοδο (A) την διεύθυνση 6 bit όπου υποδεικνύει εσωτερικά στην μνήμη δεδομένων την θέση που πρέπει να εγγραφεί ή διαβαστεί ένα δεδομένο των 32 bit.

Η εγγραφή πραγματοποιείται με την ύπαρξη ενός σήματος ελέγχου εισόδου clock (clk) (1 bit) και ενός σήματος ελέγχου εγγραφής wright enable (WE) (1 bit) έτσι ώστε να γίνεται σύγχρονη εγγραφή των δεδομένων από την δεύτερη θύρα εισόδου write data (WD) (32 bit) στον κατάλληλο καταχωρητή που υποδεικνύει η θύρα A(6 bit) κατά την ανερχόμενη ακμή του clock και μόνο όταν το wright enable έχει την τιμή 1. Αντίθετα η ανάγνωση δεδομένων πραγματοποιείται μέσω της θύρας read data (RD) (32 bit) από την κατάλληλη διεύθυνση που υποδεικνύει η θύρα A (6 bit) γίνεται ασύγχρονα χωρίς την έγκριση κάποιου σήματος.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity dmem is -- data memory
port(clk, we: in STD_LOGIC;
a, wd: in STD_LOGIC_VECTOR(31 downto 0);
rd: out STD_LOGIC_VECTOR(31 downto 0));
end dmem;

architecture Behavioral of dmem is

begin
process (clk,a)
type ramtype is array (63 downto 0) of STD_LOGIC_VECTOR(31 downto 0);
variable mem: ramtype;
begin -- read or write memory
--loop
if (clk = '1' and clk'event) then
if (we = '1') then
mem(to_integer(unsigned(a(7 downto 2)))) := wd;
end if;
end if;
rd <= mem(to_integer(unsigned(a(7 downto 2))));
--wait on clk, a;
--end loop;
end process;

end Behavioral;
```

Εικόνα 7: Κώδικας Data memory

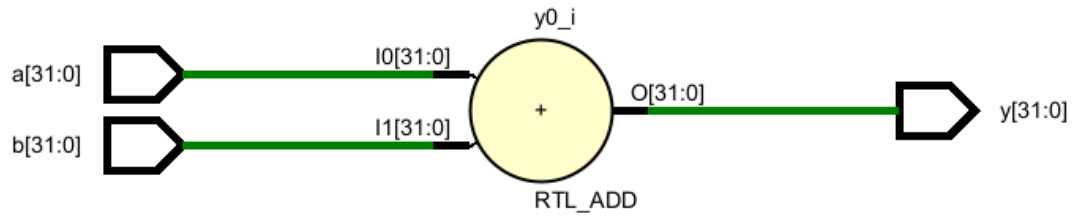


Εικόνα 8: RTL Schematic Data memory

5) Αθροιστής μη προσημασμένων αριθμών (unsigned adder) με 32 bit: ο οποίος ουσιαστικά λαμβάνει ένα μη προσημασμένο αριθμό 32 bit και του προσθέτει τον αριθμό '4'. Δηλαδή ο αθροιστής αυτός λαμβάνει στην μια από τις δύο εισόδους του (α) σε μορφή μη προσημασμένου αριθμού 32 bit, τον αριθμό '4' και στην άλλη είσοδο (b) ένα τυχαίο μη προσημασμένο αριθμό 32 bit και αφού τους προσθέσει στην έξοδο του (y) δίδεται το αποτέλεσμα της πρόσθεσης σε μορφή μη προσημασμένων αριθμού 32 bit.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity adder is -- adder
generic (WIDTH : positive := 32);
port(a, b: in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
y: out STD_LOGIC_VECTOR(WIDTH-1 downto 0));
end;

architecture Behavioral of adder is

begin
ADDER: process (a, b)
variable a_s, b_s, y_s: unsigned (WIDTH-1 downto 0);
begin
a_s := unsigned(a); -- numeric_std
b_s := unsigned(b); -- numeric_std
y_s := a_s + b_s; -- numeric_std
y <= std_logic_vector(y_s(WIDTH-1 downto 0)); -- numeric_std
end process;


end Behavioral;
```

Εικόνα 9: Κώδικας unsigned adder

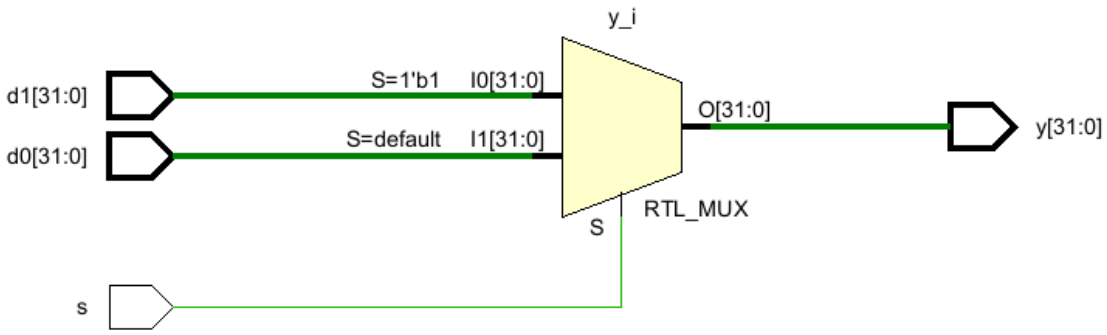


Εικόνα 10: RTL Schematic unsigned adder

6) Πολυπλέκτης 2 σε 1 (mux 2 to 1) με λέξεις των 32 bit ή των 4 bit: ο οποίος δέχεται στις δύο εισόδους του ‘d0’, ‘d1’, δύο λέξεις των 32 bit ή των 4 bit αντίστοιχα. Μέσω ενός σήματος ελέγχου επιλογής εισόδου s (source) (1 bit) επιλέγεται αν s=’1’ να περάσει στην έξοδο ‘y’ η λέξη της εισόδου ‘d1’ αλλιώς αν s=’0’ να περάσει στην έξοδο ‘y’ η λέξη της εισόδου ‘d0’.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux2 is -- two-input multiplexer
generic(width : positive := 32);
port(d0, d1: in STD_LOGIC_VECTOR(width-1 downto 0);
s: in STD_LOGIC;
y: out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture Behavioral of mux2 is

begin

y <= d1 when s='1' else d0;

end Behavioral;
```

Εικόνα 11: Κώδικας multiplexer

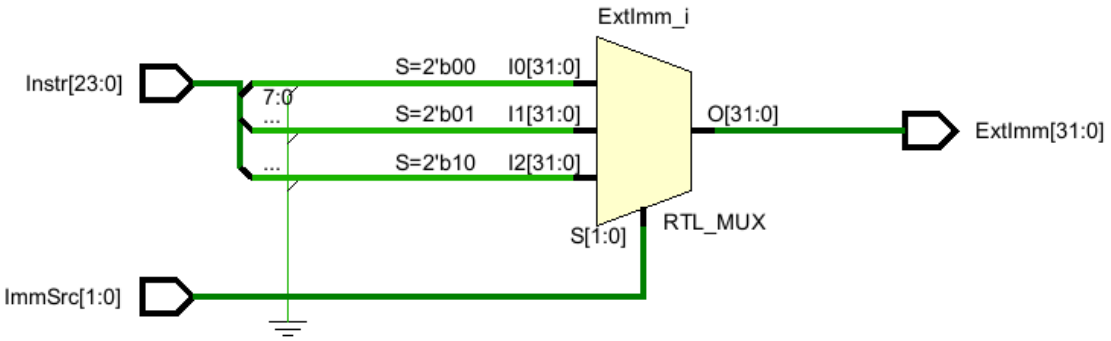


Εικόνα 12: RTL Schematic multiplexer

7) Μονάδα επέκτασης μηδενός ή πρόσημου (extend): η οποία δέχεται ως είσοδο (Instr) λέξεις με 24 bits και ένα σήμα ελέγχου είδους επέκτασης (Immsrc) των 2 bits. Αν Immsrc=’00’ πραγματοποιείται επέκτασης μηδενός από τα 8 πρώτα bit της εισόδου Instr στα 32 bit ενώ αν Immsrc=’01’ πραγματοποιείται επέκτασης μηδενός από τα 12 πρώτα bit της εισόδου Instr στα 32 bit. Από την άλλη μεριά αν Immsrc=’10’ πραγματοποιείται επέκταση πρόσημου από τα 26 bit, που λαμβάνονται αν η είσοδος (Instr) των 24 bit πολλαπλασιαστή επί 4, στα 32 bit. Στην έξοδο (ExtImm) των 32 bit της υπομονάδας αυτής λαμβάνεται όποια από τις επεκτάσεις αυτές πραγματοποιείται σε κάθε περίπτωση.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity extend is
port(Instr: in STD_LOGIC_VECTOR(23 downto 0);
ImmSrc: in STD_LOGIC_VECTOR(1 downto 0);
ExtImm: out STD_LOGIC_VECTOR(31 downto 0));
end;


architecture Behavioral of extend is

begin
process(Instr, ImmSrc) begin
case ImmSrc is
when "00" =>
ExtImm <= (X"000000"&Instr(7 downto 0));
when "01" =>
ExtImm <= (X"00000"&Instr(11 downto 0));
when "10" =>
ExtImm <= (Instr(23)& Instr(23)&Instr(23)& Instr(23)&Instr(23)& Instr(23)&Instr(23 downto 0)&"00");
when others =>
ExtImm <= (others=>'-');
end case;
end process;


end Behavioral;
```

Εικόνα 13: Κώδικας Extend



Εικόνα 14: RTL Schematic Extend

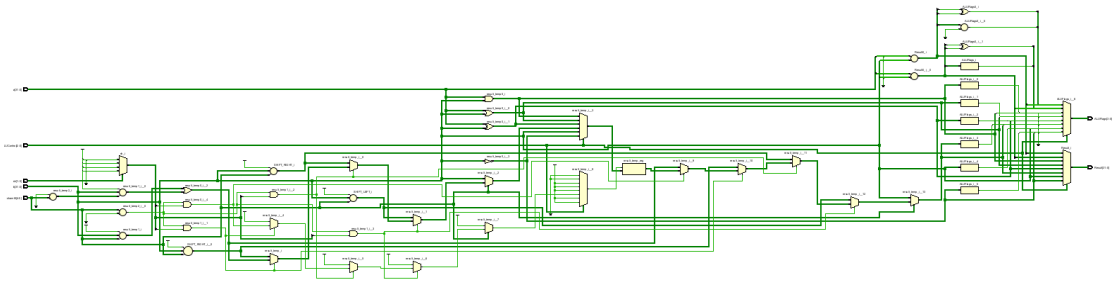
8) Αριθμητική και λογική μονάδα (ALU): η οποία δέχεται στις εισόδους (α), (b) δύο λέξεις μεγέθους 32 bit. Επιπλέον εισέρχεται στην ALU ένα σήμα ελέγχου (ALUControl) των 3 bit το οποίο θα ελέγχει το είδος της πράξης που θα πραγματοποιείται μεταξύ των λέξεων που θα βρεθούν στις εισόδους (α), (b). Αν ALUControl =’000’ πραγματοποιείται η πράξη της πρόσθεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’001’ πραγματοποιείται η πράξη της αφαίρεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’010’ πραγματοποιείται η λογική πράξη and (εντολή AND) μεταξύ των δύο ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’011’ πραγματοποιείται η λογική πράξη or (εντολή ORR) μεταξύ των δύο ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’100’ πραγματοποιείται η λογική πράξη xor (εντολή EOR) μεταξύ των δύο ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b). Επιπροσθέτως αν ALUControl =’101’ πραγματοποιείται ένας από τους τέσσερεις τύπους ολίσθησης (εντολές LSL, LSR, ASR, ROR) στον ακέραιο που αναπαρίσταται στην λέξη που λαμβάνεται από την είσοδο (b). Ο τύπος της ολίσθησης που θα πραγματοποιηθεί, καθορίζεται από ένα σήμα ελέγχου (sh) των 2 bit που εισέρχεται στην ALU και έτσι αν sh=’00’ πραγματοποιείται λογική ολίσθηση προς τα αριστερά της λέξης που λαμβάνεται από την είσοδο (b), αν sh=’01’ πραγματοποιείται λογική ολίσθηση προς τα δεξιά της λέξης που λαμβάνεται από την είσοδο (b), αν sh=’10’ πραγματοποιείται αριθμητική ολίσθηση προς τα δεξιά της λέξης που λαμβάνεται από την είσοδο (b) ενώ αν sh=’11’ πραγματοποιείται περιστροφή προς τα δεξιά της λέξης που λαμβάνεται από την είσοδο (b).

Η σταθερή ποσότητα ολίσθησης της λέξης που λαμβάνεται από την είσοδο (b) καθορίζεται από ένα σήμα εισόδου (shamt5) των 5 bit που εισέρχεται στην ALU. Ακόμη αν ALUControl =’111’ πραγματοποιείται λογική πράξη not (εντολή MVN) για την λέξη που λαμβάνεται από την είσοδο (b) ενώ αν ALUControl =’110’ θα μπορούσε να πει κάποιος πως πραγματοποιείται λογικά η πράξη της πρόσθεσης της λέξης που λαμβάνεται από την είσοδο (b) με τον αριθμό ‘0’ που ουσιαστικά σαν αποτέλεσμα έχει την ίδια λέξη που προέρχεται από την είσοδο (b) (εντολή MOV). Στην μία από τις δύο έξοδούς της υπομονάδας αυτής λαμβάνεται ένα σήμα εξόδου (Result) των 32 bit το οποίο περιλαμβάνει το αποτέλεσμα από την όποια πράξη πραγματοποιείται σε κάθε περίπτωση αναλόγως τα σήματα (ALUControl) και (sh) ενώ από την άλλη έξοδο (ALUFLAGS) των 4 bit αφού πραγματοποιηθεί η εκάστοτε πράξη ενημερώνονται οι λεγόμενες σημαίες συνθήκης που υλοποιεί η αρχιτεκτονική ARM. Έτσι τα bit του σήματος εξόδου (ALUFLAGS) από λιγότερο σημαντικό μέχρι περισσότερο σημαντικό αντιπροσωπεύουν αν υπάρχει Carry out από μια πρόσθεση ή αφαίρεση, αν υπάρχει Overflow από μια πρόσθεση ή αφαίρεση, αν το αποτέλεσμα από οποιαδήποτε πράξη είναι μηδενικό (Zero) και τέλος αν το αποτέλεσμα από οποιαδήποτε πράξη είναι αρνητικό (Negative) αντίστοιχα.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity alu is
generic (WIDTH : positive := 32);
port(a, b: in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
ALUControl: in STD_LOGIC_VECTOR(2 downto 0);
Result: out STD_LOGIC_VECTOR(WIDTH-1 downto 0);
ALUFlags: out STD_LOGIC_VECTOR(3 downto 0);
shamt5: in STD_LOGIC_VECTOR(4 downto 0);
sh: in STD_LOGIC_VECTOR(1 downto 0));
end alu;

architecture Behavioral of alu is

begin
process(a, b, ALUControl)
variable A_s, B_s, S_s: SIGNED (WIDTH+1 downto 0);
variable result_temp: std_logic_vector (WIDTH-1 downto 0);
begin
case ALUControl is
when "000" =>-- addition
A_s := signed('0'&A(WIDTH-1)&A); -- numeric_std
B_s := signed('0'&B(WIDTH-1)&B); -- numeric_std
S_s := A_s + B_s; -- numeric_std
Result <= std_logic_vector(S_s(WIDTH-1 downto 0)); -- numeric_std
ALUFlags(0) <= std_logic(S_s(WIDTH) xor S_s(WIDTH-1));
ALUFlags(1) <= std_logic(S_s(WIDTH+1));
if(S_s(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
ALUFlags(3)<= '0';
end if;
if(S_s(WIDTH-1 downto 0) = 0) then


ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
when "001" =>-- subtraction
A_s := signed('0'&A(WIDTH-1)&A); -- numeric_std
B_s := signed('0'&B(WIDTH-1)&B); -- numeric_std
S_s := A_s - B_s; -- numeric_std
Result <= std_logic_vector(S_s(WIDTH-1 downto 0)); -- numeric_std
ALUFlags(0) <= std_logic(S_s(WIDTH) xor S_s(WIDTH-1));
ALUFlags(1) <= std_logic(S_s(WIDTH+1));
if(S_s(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
ALUFlags(3)<= '0';
end if;
if(S_s(WIDTH-1 downto 0) = 0) then
ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
 when "010" =>
result_temp := a and b; -- AND gate
ALUFlags(0) <= '0';
ALUFlags(1) <= '0';
if(result_temp(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
ALUFlags(3)<= '0';
end if;
if(unsigned(result_temp) = 0) then
ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
Result <= result_temp;
 when "011" =>
result_temp := a or b; -- OR gate
ALUFlags(0) <= '0';
ALUFlags(1) <= '0';
if(result_temp(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
```

```
ALUFlags(3)<= '0';
end if;
if(unsigned(result_temp) = 0) then
ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
Result <= result_temp;
when "100" =>
result_temp := a xor b;-- XOR gate
ALUFlags(0) <= '0';
ALUFlags(1) <= '0';
if(result_temp(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
ALUFlags(3)<= '0';
end if;
if(unsigned(result_temp) = 0) then
ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
Result <= result_temp;
when "101" =>
if(unsigned(shamt5) = 0) then
result_temp := b;-- SCRB gate
elsif((unsigned(shamt5) /= 0) and (sh ="00")) then
result_temp := std_logic_vector(shift_left(unsigned(b),to_integer(unsigned(shamt5))));-- shift left gate
elsif((unsigned(shamt5) /= 0) and (sh ="01")) then
result_temp := std_logic_vector(shift_right(unsigned(b),to_integer(unsigned(shamt5))));-- shift right gate
elsif((unsigned(shamt5) /= 0) and (sh ="10")) then
result_temp := std_logic_vector(shift_right(signed(b),to_integer(unsigned(shamt5))));-- arithmetic shift right gate
elsif((unsigned(shamt5) /= 0) and (sh ="11")) then
result_temp := std_logic_vector(unsigned(b) ror (to_integer(unsigned(shamt5))));-- rotate gate
end if;
ALUFlags(0) <= '0';
ALUFlags(1) <= '0';
if(result_temp(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
ALUFlags(3)<= '0';
end if;
if(unsigned(result temp) = 0) then


ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
Result <= result_temp;
when "110" =>
result_temp := b;-- SCRB gate
ALUFlags(0) <= '0';
ALUFlags(1) <= '0';
if(result_temp(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
ALUFlags(3)<= '0';
end if;
if(unsigned(result_temp) = 0) then
ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
Result <= result_temp;
when "111" =>
result_temp := (not b);-- NOT(SRCB) gate
ALUFlags(0) <= '0';
ALUFlags(1) <= '0';
if(result_temp(WIDTH-1) = '1') then
ALUFlags(3)<= '1';
else
ALUFlags(3)<= '0';
end if;
if(unsigned(result_temp) = 0) then
ALUFlags(2)<= '1';
else
ALUFlags(2)<= '0';
end if;
Result <= result_temp;
when others =>
NULL;
end case;
end process;
end Behavioral;
```

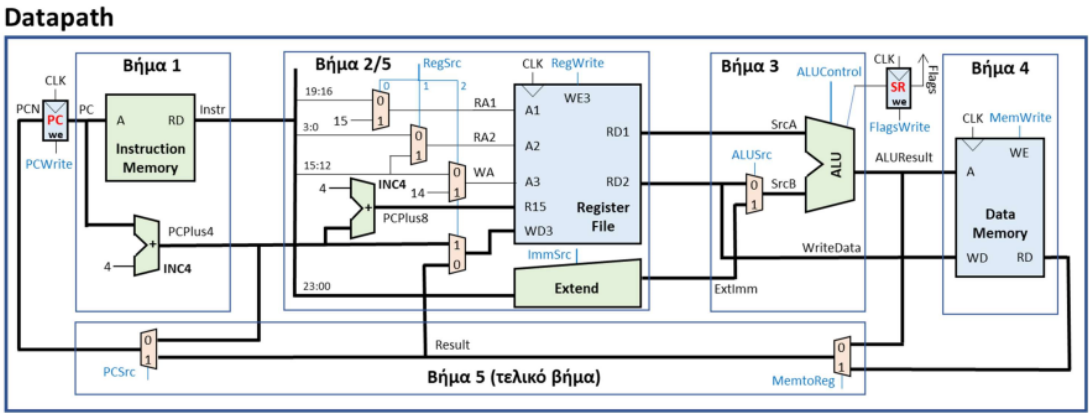
Εικόνα 15: Κώδικας ALU



Εικόνα 16: RTL Schematic ALU

Περιγραφή της δομή της διαδρομής δεδομένων (datapath) του επεξεργαστή στα ανώτερα ιεραρχικά επίπεδα

Για την υλοποίηση της δομής της διαδρομή δεδομένων της αρχιτεκτονικής ARM χρησιμοποιήθηκαν τα ψηφιακά δομικά στοιχεία (components) τα οποία περιεγράφηκαν παραπάνω. Πιο συγκεκριμένα υλοποιήθηκε η δομή της διαδρομή δεδομένων που φαίνεται εικόνα που ακολουθεί αξιοποιώντας σε ένα ανώτερο ιεραρχικά επίπεδο τα ψηφιακά δομικά στοιχεία (components) προαναφέρθηκαν.



Εικόνα 17: Δομή Datapath

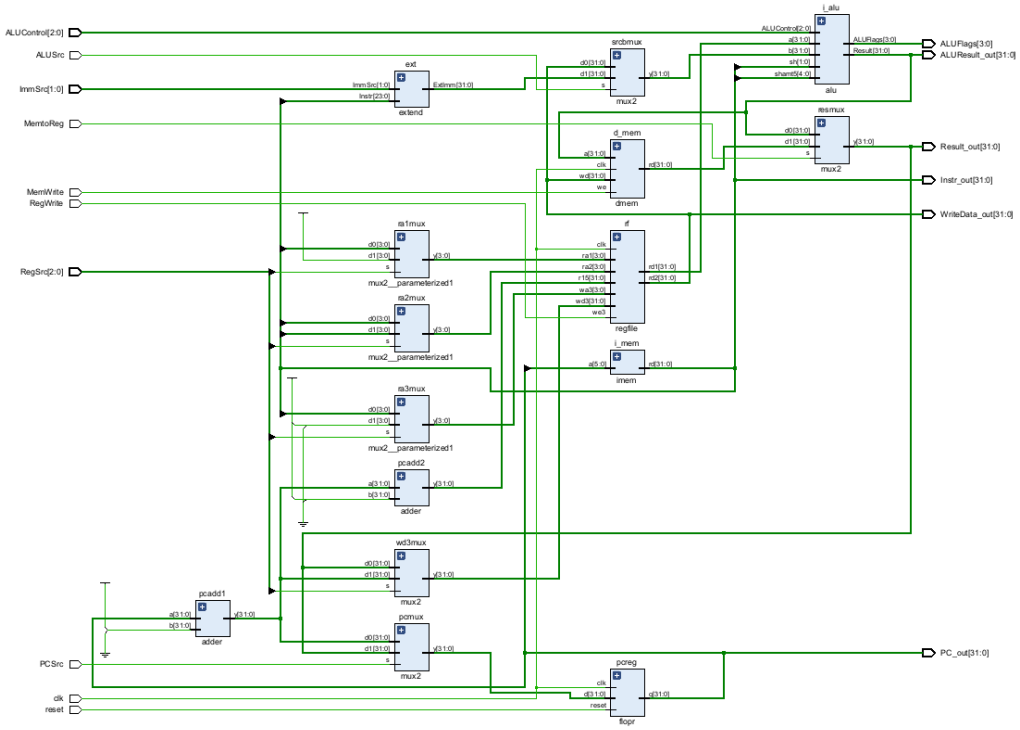
Για την υλοποίηση σε γλώσσα περιγραφής υλικού VHDL του datapath χρειάστηκαν 7 Πολυπλέκτες 2 σε 1 (mux 2 to 1) (3 των 4 bit και 4 των 32 bit), 2 Αθροιστές μη προσημασμένων αριθμών σε σειρά για να παραχθεί η απαραίτητη στην αρχιτεκτονική ARM διεύθυνση της μεθεπόμενης εντολής στην instruction memory που καταλήγει στην θύρα R15 του Register file. Ακόμη απαιτήθηκαν ένας καταχωρητής μετρητή προγράμματος, ένα αρχείο καταχωρητών, μια μνήμη εντολών, μια μνήμη δεδομένων, μια μονάδα επέκτασης μηδενός ή πρόσημου και μια αριθμητική και λογική μονάδα. Για την αρμονική και σωστή σύμφωνα με την αρχιτεκτονική ARM σύνδεση όλων αυτών των ψηφιακών δομικών στοιχείων χρησιμοποιήθηκαν σε γλώσσα περιγραφής υλικού VHDL αρκετά εσωτερικά σήματα που έχουν τα ονόματα που διαθέτουν τα ‘καλώδια’ που παρατηρούνται στην εικόνα που ακολουθεί. Ωστόσο πέρα από τα εσωτερικά σήματα που έχουν ως σκοπό την ένωση μεταξύ των ψηφιακών δομικών στοιχείων που χρησιμοποιεί η διαδρομή δεδομένων υπάρχουν και κάποια σήματα εισόδου της μονάδας διαδρομής δεδομένων. Σήματα εισόδου στην μονάδα διαδρομής δεδομένων όπως αυτά του clock (clk) ή του reset 1 bit χρησιμοποιούνται από την μονάδα για τον συγχρονισμό και την αρχικοποίηση αντίστοιχα των διαφόρων δομικών ψηφιακών μονάδων από τις οποίες αποτελείται η διαδρομή δεδομένων. Από την άλλη μεριά σήματα εισόδου στην διαδρομή δεδομένων που προέρχονται από την μονάδα ελέγχου όπως το RegSrc των 3 bit, ALUSrc του 1 bit, MemtoReg του 1 bit, PCSrc του 1 bit που αξιοποιούνται από το datapath σαν σήματα ελέγχου επιλογής εισόδου 1 bit σε πολυπλέκτες ενώ τα σήματα MemWrite του 1 bit, RegWrite του 1 χρησιμοποιούνται σαν σήματα ελέγχου σύγχρονης εγγραφής wright enable στην data memory και στο register file αντίστοιχα. Επιπροσθέτως τα σήματα εισόδου στην διαδρομή δεδομένων που προέρχονται από την μονάδα ελέγχου όπως ALUControl των 3 bit και ImmSrc των 2 bit αξιοποιούνται ως ‘controllers’ ώστε να αποφασιστεί ποια πράξη θα εκτελεστεί στην αριθμητική και λογική μονάδα και ποια λειτουργία επέκτασης μηδενός ή πρόσημου θα πραγματοποιηθεί στην μονάδα επέκτασης μηδενός ή πρόσημου.

Σαν έξοδο από την διαδρομή δεδομένων λαμβάνεται το σήμα PC_out (32 bit) που περιέχει την διεύθυνση της τρέχουσας εντολής στην Instruction Memory, το σήμα Instr_out (32 bit) που περιέχει την τρέχουσα εντολή που βγαίνει από την έξοδο της instruction memory, το σήμα ALUResult_out (32 bit) που περιέχει το αποτέλεσμα που προκύπτει από την εκάστοτε πράξη στην έξοδο της ALU, το σήμα WriteData_out (32 bit) που περιέχει τα δεδομένα που προέρχονται από την θύρα RD2 (32 bit) του Register file η οποία πραγματοποιεί ανάγνωση των δεδομένων του καταχωρητή που υποδεικνύει η θύρα A2(4 bit) και το σήμα Result_out (32 bit) που περιέχει είτε το αποτέλεσμα που προκύπτει από την εκάστοτε πράξη στην έξοδο της ALU είτε τα δεδομένα που πραγματοποιείται ανάγνωση που προέρχονται από την θύρα RD (32 bit) της Data memory η οποία πραγματοποιεί ανάγνωση των δεδομένων της διεύθυνσης που υποδεικνύει η θύρα A(4 bit) ανάλογα την τιμή του σήματος ελέγχου επιλογής εισόδου του πολυπλέκτη που ως σήμα ελέγχου επιλογής εισόδου λαμβάνει το σήμα MemtoReg που προέρχεται από την μονάδα ελέγχου. Τέλος ένα σήμα εξόδου που καταλήγει στην μονάδα ελέγχου και συγκεκριμένα στον καταχωρητή κατάστασης της μονάδας ελέγχου είναι αυτό με το όνομα ALUFlags το οποίο περιέχει τις λεγόμενες σημαίες συνθήκης που υλοποιεί η αρχιτεκτονική ARM και ενημερώνονται κάθε φορά που πραγματοποιείται μια πράξη στην ALU αλλά η αποθήκευση τους πραγματοποιείται αν χρειαστεί στον καταχωρητή κατάστασης όπως προαναφέρθηκε. Όπως είναι εύκολα παρατηρήσιμο η δομή της διαδρομή δεδομένων που φαίνεται στην εικόνα παραπάνω ταιριάζει απόλυτα τα προκύπτοντα σχηματικά διαγράμματα στο επίπεδο RTL της διαδρομής δεδομένων με την διαφορά πως δεν υπάρχουν οι έξοδοι PC_out, Instr_out, ALUResult_out, WriteData_out και Result_out που ουσιαστικά τοποθετήθηκαν έτσι ώστε να ελεγχθεί η σωστή λειτουργία του επεξεργαστή.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity datapath is
port(clk, reset: in STD_LOGIC;
RegSrc: in STD_LOGIC_VECTOR(2 downto 0);
RegWrite: in STD_LOGIC;
ImmSrc: in STD_LOGIC_VECTOR(1 downto 0);
ALUSrc: in STD_LOGIC;
ALUControl: in STD_LOGIC_VECTOR(2 downto 0);
MemtoReg: in STD_LOGIC;
PCSrc: in STD_LOGIC;
ALUFlags: out STD_LOGIC_VECTOR(3 downto 0);
MemWrite : in std_logic;
PC_out: out STD_LOGIC_VECTOR(31 downto 0);
Instr_out: out STD_LOGIC_VECTOR(31 downto 0);
ALUResult_out:out STD_LOGIC_VECTOR(31 downto 0);
WriteData_out:out STD_LOGIC_VECTOR(31 downto 0);
Result_out:out STD_LOGIC_VECTOR(31 downto 0));
--ReadData: in STD_LOGIC_VECTOR(31 downto 0));
end datapath;


architecture Behavioral of datapath is
component alu
generic (WIDTH : positive := 32);
port(a, b: in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
ALUControl: in STD_LOGIC_VECTOR(2 downto 0);
Result: out STD_LOGIC_VECTOR(WIDTH-1 downto 0);
ALUFlags: out STD_LOGIC_VECTOR(3 downto 0);
shamt5: in STD_LOGIC_VECTOR(4 downto 0);
sh: in STD_LOGIC_VECTOR(1 downto 0));
end component;
component regfile
```

```
port(clk: in STD_LOGIC;
we3: in STD_LOGIC;
ra1, ra2, wa3: in STD_LOGIC_VECTOR(3 downto 0);
wd3, r15: in STD_LOGIC_VECTOR(31 downto 0);
rd1, rd2: out STD_LOGIC_VECTOR(31 downto 0));
end component;
component adder
port(a, b: in STD_LOGIC_VECTOR(31 downto 0);
y: out STD_LOGIC_VECTOR(31 downto 0));
end component;
component extend
port(Instr: in STD_LOGIC_VECTOR(23 downto 0);
ImmSrc: in STD_LOGIC_VECTOR(1 downto 0);
ExtImm: out STD_LOGIC_VECTOR(31 downto 0));
end component;
component flopr generic(width:  positive := 32);
port(clk, reset: in STD_LOGIC;
d: in STD_LOGIC_VECTOR(width-1 downto 0);
q: out STD_LOGIC_VECTOR(width-1 downto 0));
end component;
component mux2 generic(width:  positive := 32);
port(d0, d1: in STD_LOGIC_VECTOR(width-1 downto 0);
s: in STD_LOGIC;
y: out STD_LOGIC_VECTOR(width-1 downto 0));
end component;
component dmem is -- data memory
port(clk, we: in STD_LOGIC;
a, wd: in STD_LOGIC_VECTOR(31 downto 0);
rd: out STD_LOGIC_VECTOR(31 downto 0));
end component;
component imem is -- instruction memory
generic(
    N : positive := 6; --address length
    M : positive := 32); --data word length
port(a: in STD_LOGIC_VECTOR(N-1 downto 0);
rd: out STD_LOGIC_VECTOR(M-1 downto 0));
end component;
signal PCNext, PCPlus4,PCPlus8: STD_LOGIC_VECTOR(31 downto 0);
signal PC: STD_LOGIC_VECTOR(31 downto 0):="00000000000000000000000000000000";
signal ExtImm, Result, WD3: STD_LOGIC_VECTOR(31 downto 0);
signal SrcA, SrcB: STD_LOGIC_VECTOR(31 downto 0);
signal RA1, RA2, RA3: STD_LOGIC_VECTOR(3 downto 0);
signal ALUResult: STD_LOGIC_VECTOR(31 downto 0);


signal WriteData: STD_LOGIC_VECTOR(31 downto 0);
signal ReadData:  STD_LOGIC_VECTOR(31 downto 0);
signal Instr:  STD_LOGIC_VECTOR(31 downto 0);
begin
-- next PC logic
PC_out<=PC;
pcmux: mux2  generic map (width => 32)
port map(PCPlus4, Result, PCSrc, PCNext);
pcreg: flopr generic map(width => 32) port map(clk, reset, PCNext, PC);
pcadd1: adder port map(PC, "00000000000000000000000000000100", PCPlus4);
pcadd2: adder port map(PCPlus4, "00000000000000000000000000000100", PCPlus8);
-- register file logic
ra1mux: mux2 generic map (width => 4) port map(Instr(19 downto 16), "1111", RegSrc(0), RA1);
ra2mux: mux2 generic map (width => 4) port map(Instr(3 downto 0),Instr(15 downto 12), RegSrc(1), RA2);
ra3mux: mux2 generic map (width => 4) port map(Instr(15 downto 12),"1110",RegSrc(2), RA3);
wd3mux: mux2 generic map (width => 32) port map(Result, PCPlus4, RegSrc(2), WD3);
rf: regfile port map(clk, RegWrite, RA1, RA2,RA3, WD3,PCPlus8, SrcA, WriteData);
WriteData_out<=WriteData;
resmux: mux2 generic map(width => 32)
port map(ALUResult, ReadData, MemtoReg, Result);
Result_out<=Result;
ext: extend port map(Instr(23 downto 0), ImmSrc, ExtImm);
-- ALU logic
srcbmux: mux2 generic map(width => 32)
port map(WriteData, ExtImm, ALUSrc, SrcB);
i_alu: alu generic map(width => 32) port map(SrcA, SrcB, ALUControl, ALUResult, ALUFlags,Instr(11 downto 7),Instr(6 downto 5));
ALUResult_out<=ALUResult;
-- dmem logic
d_mem: dmem port map(clk, MemWrite, ALUResult, WriteData, ReadData);
-- dmem logic
i_mem: imem port map(PC(7 downto 2), Instr);
Instr_out<=Instr;
end Behavioral;
```

Εικόνα 18: Κώδικας Datapath



Εικόνα 19: RTL Schematic Datapath

Περιγραφή των ψηφιακών δομικών στοιχείων της μονάδας ελέγχου (control unit or controller)

Όπως προαναφέρθηκε η μονάδας ελέγχου αποτελείται από κάποια ψηφιακά δομικά στοιχεία (components) τα οποία ευθέως αμέσως θα σκιαγραφηθούν και στην συνέχεια θα δοθεί ο κώδικας στην γλώσσα περιγραφής υλικού VHDL για την υλοποίηση τους:

1) Αποκωδικοποιητής (Decoder) εντολής, σημάτων έγκρισης εγγραφής και λογικής επιλογής διεύθυνσης επόμενης εντολής: ο οποίος δέχεται ως είσοδό op (των 2 bit) στην οποία διοχετεύεται το $26^{ο}$ και το $27^{ο}$ bit από το σήμα εξόδου Instr_out της διαδρομής δεδομένων το οποίο ουσιαστικά περιέχει την τρέχουσα εντολή που βγαίνει από την έξοδο της instruction memory, αφού βέβαια το σήμα Instr_out περάσει πρώτα από το ανώτερο ιεραρχικά επίπεδο του controller. Αντίστοιχα δέχεται ως είσοδό funct (των 6 bit) στην οποία διοχετεύεται από το $20^{ο}$ και το $25^{ο}$ bit του σήματος εξόδου Instr_out καθώς και το σήμα εισόδου Rd (των 4 bit) από το $12^{ο}$ και το $15^{ο}$ bit του σήματος εξόδου Instr_out. Τα op, funct, Rd αναπαριστούν πεδία της αρχιτεκτονικής ARM που περιεγράφηκαν παραπάνω. Έτσι λοιπόν η υπομονάδα του αποκωδικοποιητή παράγει τα σήματα επιλογής πολυπλεκτών που ρυθμίζουν τη ροή δεδομένων και τα σήματα ελέγχου συνδυαστικής λογικής στη διαδρομή δεδομένων, καθώς και τα σήματα έγκρισης εγγραφής στο Register file ή την data memory. Επίσης ο αποκωδικοποιητής ενεργοποιεί το σήμα επιλογής ελέγχου της εισόδου του πολυπλέκτη διεύθυνσης επόμενης εντολής εάν απαιτείται εγγραφή στον R15 (PC) ή εκτελείται εντολή διακλάδωσης και απενεργοποιεί σε αντίθετη περίπτωση. Έτσι σαν έξοδο του ο αποκωδικοποιητής παράγει τα σήματα ελέγχου επιλογής εισόδου πολυπλεκτών RegSrc των 3 bit, ALUSrc του 1 bit και MemtoReg του 1 bit καθώς και τα σήματα ελέγχου λειτουργίας της αριθμητικής και λογικής μονάδας ALUControl των 3 bit και της μονάδας επέκτασης μηδενός ή πρόσημου ImmSrc των 2 bit. Τα σήματα εξόδου που περιεγράφηκαν παραπάνω αφού περάσουν πρώτα από το ανώτερο ιεραρχικά επίπεδο της μονάδας ελέγχου καταλήγουν κατευθείαν στη διαδρομή δεδομένων. Επιπρόσθετα σήματα εξόδου του αποκωδικοποιητή είναι τα RegW του 1 bit, MemW του 1 bit, FlagW των 2 bit τα οποία είναι σήματα έγκρισης εγγραφής (για το Register file, την Data Memory και τον καταχωρητή κατάστασης αντίστοιχα) και θα περάσουν πρώτα από την υπομονάδα της μονάδας ελέγχου που ονομάζεται μονάδα λογικής ελέγχου συνθήκης ώστε να γίνει έλεγχος πρώτα αν η συνθήκη της συγκεκριμένης εντολής ικανοποιείται με βάση την κατάσταση του καταχωρητή κατάστασης (ο καταχωρητή κατάστασης βρίσκεται στην μονάδα λογικής ελέγχου συνθήκης) στον οποίον αποθηκεύονται και οι σημαίες συνθήκης. Αν ικανοποιηθεί η συνθήκη τα σήματα έγκρισης εγγραφής θα αποσταλούν ατόφια μέσω του ανώτερου ιεραρχικά επιπέδου της μονάδας ελέγχου, στη μονάδα διαδρομής δεδομένων, ενώ αν η συνθήκη δεν ικανοποιηθεί η διαδρομή δεδομένων θα λάβει σήματα που θα σημαίνουν την μη έγκριση εγγραφής. Τέλος υπάρχει άλλο ένα σήμα εξόδου PCS του 1 bit, το οποίο είναι ένα σήμα επιλογής ελέγχου της εισόδου του πολυπλέκτη διεύθυνσης επόμενης εντολής το οποίο επίσης θα ακολουθήσει την πορεία των σημάτων έγκρισης εγγραφής περνώντας πρώτα από την υπομονάδα της μονάδας ελέγχου που ονομάζεται μονάδα λογικής ελέγχου συνθήκης και καταλήγοντας εν τέλη στην διαδρομή δεδομένων δίνοντας την κατάλληλη τιμή στο σήμα επιλογής ελέγχου της εισόδου του πολυπλέκτη διεύθυνσης επόμενης εντολής.

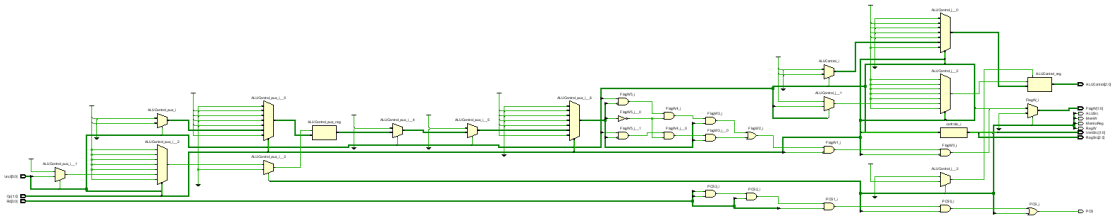
Εντολή	Instr27:26 :op	Instr 25:20 :funct	Τύπος	RegSrc	ALUSrc	Imm Src	Control ALU	Memto Reg	RegW	MemW	Flags W	Instr15:12 :Rd	PCS
ADD	00	1 0100 1	DP Imm	000	1	00	000	0	1	0	1	0000 - 1110	0
ADD	00	0 0100 1	DP Reg	000	0	00	000	0	1	0	1	0000 - 1110	0
ADD	00	1 0100 0	DP Imm	000	1	00	000	0	1	0	0	0000 - 1110	0
ADD	00	0 0100 0	DP Reg	000	0	00	000	0	1	0	0	0000 - 1110	0
ADD	00	1 0100 1	DP Imm	000	1	00	000	0	1	0	1	1111	1
ADD	00	0 0100 1	DP Reg	000	0	00	000	0	1	0	1	1111	1
ADD	00	1 0100 0	DP Imm	000	1	00	000	0	1	0	0	1111	1
ADD	00	0 0100 0	DP Reg	000	0	00	000	0	1	0	0	1111	1
SUB	00	1 0010 1	DP Imm	000	1	00	001	0	1	0	1	0000 - 1110	0
SUB	00	0 0010 1	DP Reg	000	0	00	001	0	1	0	1	0000 - 1110	0
SUB	00	1 0010 0	DP Imm	000	1	00	001	0	1	0	0	0000 - 1110	0
SUB	00	0 0010 0	DP Reg	000	0	00	001	0	1	0	0	0000 - 1110	0
SUB	00	1 0010 1	DP Imm	000	1	00	001	0	1	0	1	1111	1
SUB	00	0 0010 1	DP Reg	000	0	00	001	0	1	0	1	1111	1
SUB	00	1 0010 0	DP Imm	000	1	00	001	0	1	0	0	1111	1
SUB	00	0 0010 0	DP Reg	000	0	00	001	0	1	0	0	1111	1
CMP	00	1 1010 1	DP Imm	000	1	00	010	0	0	0	1	XXXX	0
CMP	00	0 1010 1	DP Reg	000	0	00	010	0	0	0	1	XXXX	0
CMP	00	1 1010 1	DP Imm	000	1	00	010	0	0	0	1	XXXX	0
CMP	00	0 1010 1	DP Reg	000	0	00	010	0	0	0	1	XXXX	0
AND	00	1 0000 1	DP Imm	000	1	00	010	0	1	0	1	0000 - 1110	0
AND	00	0 0000 1	DP Reg	000	0	00	010	0	1	0	1	0000 - 1110	0
AND	00	1 0000 0	DP Imm	000	1	00	010	0	1	0	0	0000 - 1110	0
AND	00	0 0000 0	DP Reg	000	0	00	010	0	1	0	0	0000 - 1110	0
AND	00	1 0000 1	DP Imm	000	1	00	010	0	1	0	1	1111	1
AND	00	0 0000 1	DP Reg	000	0	00	010	0	1	0	1	1111	1
AND	00	1 0000 0	DP Imm	000	1	00	010	0	1	0	0	1111	1
AND	00	0 0000 0	DP Reg	000	0	00	010	0	1	0	0	1111	1
ORR	00	1 1100 1	DP Imm	000	1	00	011	0	1	0	1	0000 - 1110	0
ORR	00	0 1100 1	DP Reg	000	0	00	011	0	1	0	1	0000 - 1110	0
ORR	00	1 1100 0	DP Imm	000	1	00	011	0	1	0	0	0000 - 1110	0
ORR	00	0 1100 0	DP Reg	000	0	00	011	0	1	0	0	0000 - 1110	0
ORR	00	1 1100 1	DP Imm	000	1	00	011	0	1	0	1	1111	1
ORR	00	0 1100 1	DP Reg	000	0	00	011	0	1	0	1	1111	1
ORR	00	1 1100 0	DP Imm	000	1	00	011	0	1	0	0	1111	1
ORR	00	0 1100 0	DP Reg	000	0	00	011	0	1	0	0	1111	1
E0R	00	1 0001 1	DP Imm	000	1	00	100	0	1	0	1	0000 - 1110	0
E0R	00	0 0001 1	DP Reg	000	0	00	100	0	1	0	1	0000 - 1110	0
E0R	00	1 0001 0	DP Imm	000	1	00	100	0	1	0	0	0000 - 1110	0
E0R	00	0 0001 0	DP Reg	000	0	00	100	0	1	0	0	0000 - 1110	0
E0R	00	1 0001 1	DP Imm	000	1	00	100	0	1	0	1	1111	1
E0R	00	0 0001 1	DP Reg	000	0	00	100	0	1	0	1	1111	1
E0R	00	1 0001 0	DP Imm	000	1	00	100	0	1	0	0	1111	1
E0R	00	0 0001 0	DP Reg	000	0	00	100	0	1	0	0	1111	1
MOV,LSL,LSR,ASR,ROR	00	1 1101 1	DP Imm	000	1	00	110	0	1	0	1	0000 - 1110	0
MOV,LSL,LSR,ASR,ROR	00	0 1101 1	DP Reg	000	0	00	101	0	1	0	1	0000 - 1110	0
MOV,LSL,LSR,ASR,ROR	00	1 1101 0	DP Imm	000	1	00	110	0	1	0	0	0000 - 1110	0
MOV,LSL,LSR,ASR,ROR	00	0 1101 0	DP Reg	000	0	00	101	0	1	0	0	0000 - 1110	0
MOV,LSL,LSR,ASR,ROR	00	1 1101 1	DP Imm	000	1	00	110	0	1	0	1	1111	1
MOV,LSL,LSR,ASR,ROR	00	0 1101 1	DP Reg	000	0	00	101	0	1	0	1	1111	1
MOV,LSL,LSR,ASR,ROR	00	1 1101 0	DP Imm	000	1	00	110	0	1	0	0	1111	1
MOV,LSL,LSR,ASR,ROR	00	0 1101 0	DP Reg	000	0	00	101	0	1	0	0	1111	1
MVN	00	1 1111 1	DP Imm	000	1	00	111	0	1	0	1	0000 - 1110	0
MVN	00	0 1111 1	DP Reg	000	0	00	111	0	1	0	1	0000 - 1110	0
MVN	00	1 1111 0	DP Imm	000	1	00	111	0	1	0	0	0000 - 1110	0
MVN	00	0 1111 0	DP Reg	000	0	00	111	0	1	0	0	0000 - 1110	0
MVN	00	1 1111 1	DP Imm	000	1	00	111	0	1	0	1	1111	1
MVN	00	0 1111 1	DP Reg	000	0	00	111	0	1	0	1	1111	1
MVN	00	1 1111 0	DP Imm	000	1	00	111	0	1	0	0	1111	1
MVN	00	0 1111 0	DP Reg	000	0	00	111	0	1	0	0	1111	1
LDR	01	0 1100 1	M Imm +	000	1	01	000	1	1	0	0	0000 - 1110	0
LDR	01	0 1000 1	M Imm –	000	1	01	001	1	1	0	0	0000 - 1110	0
LDR	01	0 1100 1	M Imm +	000	1	01	000	1	1	0	0	1111	1
LDR	01	0 1000 1	M Imm –	000	1	01	001	1	1	0	0	1111	1
STR	01	0 1100 0	M Imm +	010	1	01	000	0	0	1	0	XXXX	0
STR	01	0 1000 0	M Imm –	010	1	01	001	0	0	1	0	XXXX	0
STR	01	0 1100 0	M Imm +	010	1	01	000	0	0	1	0	XXXX	0
STR	01	0 1000 0	M Imm –	010	1	01	001	0	0	1	0	XXXX	0
B	10	1 0XXX X	B Imm +	001	1	10	000	0	0	0	0	XXXX	1
B	10	1 0XXX X	B Imm +	001	1	10	000	0	0	0	0	XXXX	1
BL	10	1 1XXX X	B Imm +	101	1	10	000	0	1	0	0	XXXX	1
BL	10	1 1XXX X	B Imm +	101	1	10	000	0	1	0	0	XXXX	1

Εικόνα 20: Πίνακας αλήθειας Decoder (όπου X δεν μας ενδιαφέρει η τιμή που λαμβάνεται)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity decoder is
port(Op: in STD_LOGIC_VECTOR(1 downto 0);
Funct: in STD_LOGIC_VECTOR(5 downto 0);
Rd: in STD_LOGIC_VECTOR(3 downto 0);
FlagW: out STD_LOGIC_VECTOR(1 downto 0);
PCS, RegW,MemW: out STD_LOGIC;
MemtoReg, ALUSrc: out STD_LOGIC;
ImmSrc: out STD_LOGIC_VECTOR(1 downto 0);
RegSrc: out STD_LOGIC_VECTOR(2 downto 0);
ALUControl: out STD_LOGIC_VECTOR(2 downto 0));
end decoder;

architecture Behavioral of decoder is
signal controls: STD_LOGIC_VECTOR(10 downto 0);
signal ALUOp, Branch,RegW_aux: STD_LOGIC;
signal op2: STD_LOGIC_VECTOR(7 downto 0);
begin

op2 <= (Op&Funct);
process(Op,Funct,Rd,op2)

begin -- Main Decoder

case (op2) is
when "00001000" => controls <= "00000001001"; --add cases
when "00001001" => controls <= "00000001001";
when "00101000" => controls <= "00000101001";
when "00101001" => controls <= "00000101001";


when "00000100" => controls <= "00000001001"; --sub cases
when "00000101" => controls <= "00000001001";
when "00100100" => controls <= "00000101001";
when "00100101" => controls <= "00000101001";

when "00000000" => controls <= "00000001001"; --and cases
when "00000001" => controls <= "00000001001";
when "00100000" => controls <= "00000101001";
when "00100001" => controls <= "00000101001";

when "00000010" => controls <= "00000001001"; --xor cases
when "00000011" => controls <= "00000001001";
when "00100010" => controls <= "00000101001";
when "00100011" => controls <= "00000101001";

when "00011010" => controls <= "00000001001"; --srcB, LSL, LSR, ASR, ROR cases
when "00011011" => controls <= "00000001001";
when "00111010" => controls <= "00000101001";
when "00111011" => controls <= "00000101001";

when "00011110" => controls <= "00000001001"; --not(srcB) cases
when "00011111" => controls <= "00000001001";
when "00111110" => controls <= "00000101001";
when "00111111" => controls <= "00000101001";

when "00011000" => controls <= "00000001001"; --or cases
when "00011001" => controls <= "00000001001";
when "00111000" => controls <= "00000101001";
when "00111001" => controls <= "00000101001";

when "00110101" => controls <= "00000100001";--cmp cases
when "00010101" => controls <= "00000000001";

when "01011000" => controls <= "01001110101"; --store +imm12
when "01010000" => controls <= "01001110101"; --store -imm12

when "01011001" => controls <= "00001111001";--load +imm12
when "01010001" => controls <= "00001111001";--load -imm12

when "10100000" => controls <= "00110100011";--branch +imm24 cases
when "10100001" => controls <= "00110100011";
when "10100010" => controls <= "00110100011";
when "10100011" => controls <= "00110100011";


when "10100100" => controls <= "00110100011";
when "10100101" => controls <= "00110100011";
when "10100110" => controls <= "00110100011";
when "10100111" => controls <= "00110100011";

when "10101000" => controls <= "00110100011";
when "10101001" => controls <= "00110100011";
when "10101010" => controls <= "00110100011";
when "10101011" => controls <= "00110100011";

when "10101100" => controls <= "00110100011";
when "10101101" => controls <= "00110100011";
when "10101110" => controls <= "00110100011";
when "10101111" => controls <= "00110100011";

when "10110000" => controls <= "10110101011";--branch link +imm24 cases
when "10110001" => controls <= "10110101011";
when "10110010" => controls <= "10110101011";
when "10110011" => controls <= "10110101011";

when "10110100" => controls <= "10110101011";
when "10110101" => controls <= "10110101011";
when "10110110" => controls <= "10110101011";
when "10110111" => controls <= "10110101011";

when "10111000" => controls <= "10110101011";
when "10111001" => controls <= "10110101011";
when "10111010" => controls <= "10110101011";
when "10111011" => controls <= "10110101011";

when "10111100" => controls <= "10110101011";
when "10111101" => controls <= "10110101011";
when "10111110" => controls <= "10110101011";
when "10111111" => controls <= "10110101011";

when others => controls <= "-----------";
end case;
end process;
(RegSrc(2) , RegSrc(1), RegSrc(0), ImmSrc(1), ImmSrc(0), ALUSrc, MemtoReg, RegW, MemW,Branch, ALUOp) <= controls;
RegW_aux<= controls(3);
process(Op,Funct,Rd,ALUOp)
variable ALUControl_aux: STD_LOGIC_VECTOR(2 downto 0);
begin -- ALU Decoder
```

```
if (ALUOp='1') then
case (opsFunct(4 downto 1)) is
when "000100"|"011100"|"100000"|"100001"|"100010"|"100011"|"100100"|"100101"|"100110"|"100111"|"101000"|"101001"|"101010"|"101011"|"101100"|"101101"|"101110"|"101111" => ALUControl <= "000"; ALUControl_aux :="000" ;
when "000010"|"011000"|"001010" => ALUControl <= "001"; ALUControl_aux :="001" ;-- SUB
when "000000" => ALUControl <= "010"; ALUControl_aux :="010" ;-- AND
when "001100" => ALUControl <= "011"; ALUControl_aux :="011" ;-- ORR
when "000001" => ALUControl <= "100"; ALUControl_aux :="100" ;-- XOR
when "001101" =>
if (funct(5)='0') then
ALUControl <= "101"; ALUControl_aux :="101" ;-- SRCB LSR ASR with I=0
elsif (funct(5)='1') then
ALUControl <= "110"; ALUControl_aux :="110" ;-- SRCB with I=1
end if;
when "001111" => ALUControl <= "111"; ALUControl_aux :="111" ;-- not(SRCB)
when others => ALUControl <= "---"; ALUControl_aux :="---" ;-- unimplemented
end case;
FlagW(1) <= Funct(0);
FlagW(0) <= ((Funct(0) and (not ALUControl_aux(1))and (not ALUControl_aux(2)) and ALUControl_aux(0))or(Funct(0) and (not ALUControl_aux(0))and (not ALUControl_aux(2)) and ALUControl_aux(1)))and (not op(0))and (not o
else
ALUControl <= "000";
FlagW <= "00";
end if;
end process;
PCS <= (Rd(0)and Rd(1)and Rd(2)and Rd(3)and RegW_aux) or Branch;


end Behavioral;
```

Εικόνα 21: Κωδικας Decoder



Εικόνα 22: RTL Schematic Decoder

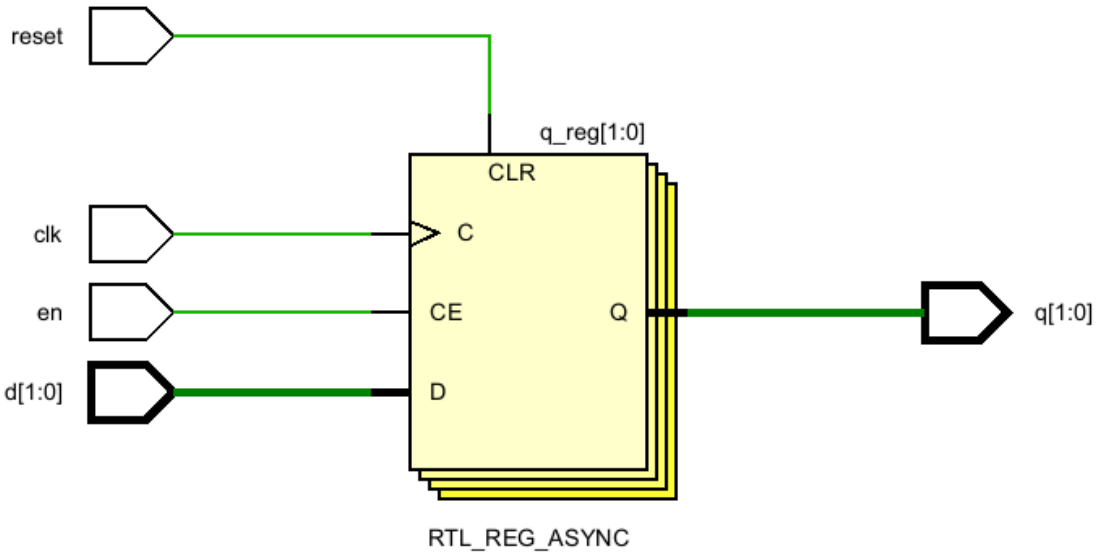
2) Καταχωρητής με wright enable των 2 bit : ο οποίος δεν είναι κάτι άλλο από έναν καταχωρητή 2 bit, που στην είσοδο του δέχεται τις ένα σήμα (d)(2 bit) ,το οποίο σήμα περνά στην έξοδο (q) κατά την ανερχόμενη ακμή του (1) bit σήματος ελέγχου εισόδου clock (clk) καθώς και μόνο όταν το σήμα ελέγχου εγγραφής wright enable (en) (1 bit) έχει την τιμή 1 έτσι ώστε να γίνεται σύγχρονη εγγραφή των δεδομένων ενώ η έξοδος του αρχικοποιείται στην τιμή 0 με το (1) bit σήματος αρχικοποίησης εξόδου reset.

d	clk	reset	enable (en)	q
XX	X	1	X	00
XX	0	0	X	no change
00	1	0	1	00
01	1	0	1	01
10	1	0	1	10
11	1	0	1	11
XX	1	0	0	no change

Εικόνα 23: Πίνακας αλήθειας καταχωρητή με wright enable των 2 bit (όπου X δεν μας ενδιαφέρει η τιμή που λαμβάνεται)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity flopenr is -- flip-flop with enable and synchronous reset
generic(width: positive := 2);
port(clk, reset, en: in STD_LOGIC;
d: in STD_LOGIC_VECTOR(width-1 downto 0);
q: out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture Behavioral of flopenr is

begin
process(clk, reset) begin
if reset='1' then q <= (others => '0');
elsif rising_edge(clk) then
if en='1' then
q <= d;
end if;
end if;
end process;

end Behavioral;
```

Εικόνα 24: Κώδικας καταχωρητή με wright enable των 2 bit



Εικόνα 25: RTL Schematic καταχωρητή με wright enable των 2 bit

3) Μονάδα ελέγχου συνθήκης (Condcheck): η οποία δέχεται ως είσοδό το σήμα cond (των 4 bit) στο οποίο διοχετεύεται από το 28^{ο} και το 31^{ο} bit το σήμα εξόδου Instr_out της διαδρομής δεδομένων το οποίο ουσιαστικά περιέχει την τρέχουσα εντολή που βγαίνει από την έξοδο της instruction memory, αφού βέβαια το σήμα Instr_out περάσει πρώτα από το ανώτερο ιεραρχικά επίπεδο του controller και στην συνέχεια στην λογική μονάδα ελέγχου συνθήκης που θα περιγράφει παρακάτω.

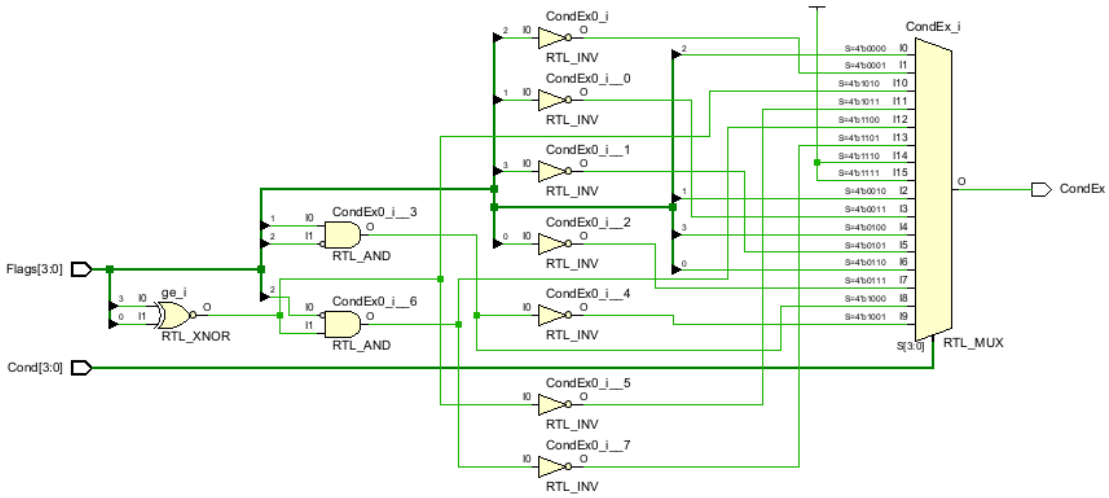
Το cond αναπαριστά το αντίστοιχο πεδίο της αρχιτεκτονικής ARM που περιεγράφηκε παραπάνω και έχει να κάνει με την συνθήκη υπό την οποία πρέπει να εκτελεστεί η εντολή. Μια ακόμα είσοδος στην μονάδα ελέγχου συνθήκης είναι το σήμα Flags (των 4 bit) στο οποίο διοχετεύεται από τον καταχωρητή κατάστασης (ο οποίος αποτελείται ουσιαστικά δύο καταχωρητές των 2 bit που φυλάσσουν τις σημαίες συνθήκης που υλοποιεί η αρχιτεκτονική ARM) που βρίσκεται στην ανώτερη ιεραρχικά μονάδα με το όνομα λογική μονάδα ελέγχου συνθήκης (η μονάδα ελέγχου συνθήκης βρίσκεται επίσης στην ανώτερη ιεραρχικά μονάδα με το όνομα λογική μονάδα ελέγχου συνθήκης). Η μονάδα ελέγχου συνθήκης ακολουθώντας την κατάλληλη συνδυαστική λογική συγκρίνει κατάλληλα τις δύο εισόδους και διαπιστώνει αν ικανοποιείται η συνθήκη υπό την οποία πρέπει να εκτελεστεί η εντολή που πάρθηκε από την Instruction memory της διαδρομής δεδομένων. Το αποτέλεσμα ελέγχου της συνθήκης υπό την οποία πρέπει να εκτελεστεί η εντολή περνιέται ως έξοδος σε ένα σήμα με το όνομα Condex του 1 bit το οποίο μπορεί να χαρακτηριστεί σαν λογική μεταβλητή λαμβάνοντας την τιμή ‘0’ αν δεν ικανοποιείται η συνθήκη και την τιμή ‘1’ αν ικανοποιείται η συνθήκη.

Instr 31:28 :cond	Περιγραφή συνθήκης	CondEx
0000	Equal	Z
0001	Not equal	notZ
0010	Carry set / unsigned higher or same	C
0011	Carry clear / unsigned lower	notC
0100	Minus / negative	N
0101	Plus / positive or zero	notN
0110	Overflow / overflow set	V
0111	No overflow / overflow clear	notN
1000	Unsigned higher	(notZ) and C
1001	Unsigned lower or same	Z or (notC)
1010	Signed greater or equal	not(N xor V)
1011	Signed less	(N xor V)
1100	Signed greater	(notZ) and (not(N xor V))
1101	Signed less or equal	(notZ) or (not(N xor V))
1110	Always / unconditional	1
1111	For unconditional instructions	1

Εικόνα 26: Πίνακας αλήθειας Condcheck, όπου N, Z, C, V οι σημαίες συνθήκης που υλοποιεί η αρχιτεκτονική ARM και αντιστοιχούν στα Bit της εισόδου Flags της Condcheck ως εξής: Z->Flags(2), C->Flags(1), V->Flags(0),N->Flags(3)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity condcheck is
port(Cond: in STD_LOGIC_VECTOR(3 downto 0);
Flags: in STD_LOGIC_VECTOR(3 downto 0);
CondEx: out STD_LOGIC);
end condcheck;

architecture Behavioral of condcheck is
signal neg, zero, carry, overflow, ge: STD_LOGIC;
begin
(neg, zero, carry, overflow) <= Flags;
ge <= (neg xnor overflow);
process(Cond,Flags)
begin -- Condition checking
case Cond is
when "0000" => CondEx <= zero;
when "0001" => CondEx <= not zero;
when "0010" => CondEx <= carry;
when "0011" => CondEx <= not carry;
when "0100" => CondEx <= neg;
when "0101" => CondEx <= not neg;
when "0110" => CondEx <= overflow;
when "0111" => CondEx <= not overflow;
when "1000" => CondEx <= carry and (not zero);
when "1001" => CondEx <= not(carry and (not zero));
when "1010" => CondEx <= ge;
when "1011" => CondEx <= not ge;
when "1100" => CondEx <= (not zero) and ge;
when "1101" => CondEx <= not ((not zero) and ge);
when "1110" => CondEx <= '1';
when "1111" => CondEx <= '1';
when others => CondEx <= '-';
end case;
end process;
end Behavioral;
```

Εικόνα 27: Κώδικας Condcheck



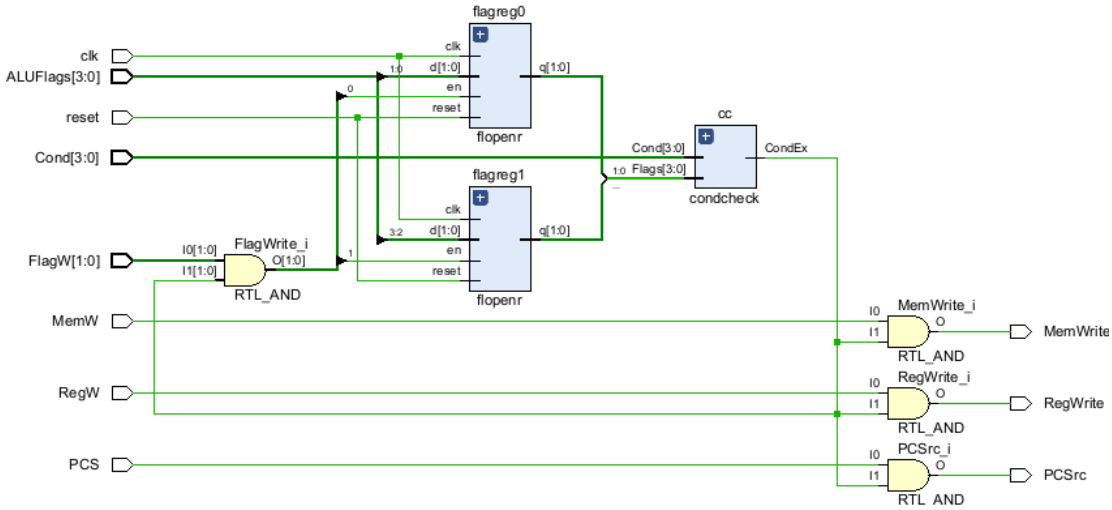
Εικόνα 28: RTL Schematic Condcheck

4) Λογική μονάδα ελέγχου συνθήκης (CondLogic): η οποία δέχεται ως είσοδό το σήμα cond (των 4 bit) στο οποίο διοχετεύεται από το 28^{ο} και το 31^{ο} bit το σήμα εξόδου Instr_out της διαδρομής δεδομένων το οποίο ουσιαστικά περιέχει την τρέχουσα εντολή που βγαίνει από την έξοδο της instruction memory, αφού βέβαια το σήμα Instr_out περάσει πρώτα από το ανώτερο ιεραρχικά επίπεδο του controller. Το cond αναπαριστά το αντίστοιχο πεδίο της αρχιτεκτονικής ARM που περιεγράφηκε παραπάνω και έχει να κάνει με την συνθήκη υπό την οποία πρέπει να εκτελεστεί η εντολή. Μια ακόμα είσοδος στην μονάδα ελέγχου συνθήκης είναι το σήμα AluFlags (των 4 bit) στο οποίο διοχετεύεται το σήμα εξόδου AluFlags της διαδρομής δεδομένων στο οποίο με την δειρά του διοχετεύονται οι τρέχουσες σημαίες συνθήκης που παράγονται μετά από κάθε πράξη στην αριθμητική και λογική μονάδα και συγκεκριμένα το σήμα εξόδου της AluFlags, αφού βέβαια το σήμα AluFlags της διαδρομής δεδομένων περάσει πρώτα από το ανώτερο ιεραρχικά επίπεδο του controller.

Η λογική μονάδα ελέγχου αποτελείται από δύο καταχωρητές με wright enable των 2 bit οι οποίοι ουσιαστικά φυλάσσουν τις λεγόμενες σημαίες συνθήκης (4 bit) που υλοποιεί η αρχιτεκτονική ARM, άρα θα χρειαστούν και ένα σήμα ελέγχου εισόδου clock (clk) (1) bit καθώς και ένα σήμα αρχικοποίησης reset (1) bit. Το σύνολο αυτό των δύο καταχωρητών ονομάζεται καταχωρητής κατάστασης και τα 4 bit του καταχωρητή κατάστασης αντιπροσωπεύουν:1) την ύπαρξη Carry out από μια πρόσθεση ή αφαίρεση, 2) την ύπαρξη Overflow από μια πρόσθεση ή αφαίρεση, 3) αν το αποτέλεσμα από οποιαδήποτε πράξη είναι μηδενικό (Zero) και 4) αν το αποτέλεσμα από οποιαδήποτε πράξη είναι αρνητικό (Negative) αντίστοιχα. Ο ένας καταχωρητής στο λιγότερο σημαντικό bit αποθηκεύει την πληροφορία για την ύπαρξη ή όχι Overflow με το λογικό ‘1’ ή ‘0’ αντίστοιχα ενώ στο άλλο στο άλλο bit αποθηκεύει την πληροφορία για την ύπαρξη ή όχι Carry out με το λογικό ‘1’ ή ‘0’ αντίστοιχα. Ο δεύτερος καταχωρητής στο λιγότερο σημαντικό bit αποθηκεύει την πληροφορία για το αν το δεδομένο είναι μηδενικό ή όχι με το λογικό ‘1’ ή ‘0’ αντίστοιχα ενώ στο άλλο στο άλλο bit αποθηκεύει την πληροφορία για το αν το δεδομένο είναι αρνητικό ή όχι Carry out με το λογικό ‘1’ ή ‘0’ αντίστοιχα. Οι έξοδος του καταχωρητή κατάστασης όπως και η είσοδος cond διοχετεύονται στην μονάδα ελέγχου συνθήκης (Condcheck) που περιέχεται στην Condlogic και έτσι ελέγχεται η συνθήκη υπό την οποία πρέπει να εκτελεστεί η εντολή και έτσι λαμβάνεται η λογική έξοδος Condex 1 bit της Condcheck. Επιπρόσθετα σαν σήματα εισόδου της Condlogic είναι τα RegW του 1 bit, MemW του 1 bit, FlagW των 2 bit, στα οποία διοχετεύονται οι αντίστοιχες έξοδοι των σημάτων έγκρισης εγγραφής RegW του 1 bit, MemW του 1 bit, FlagW των 2 bit του αποκωδικοποιητή ενώ άλλο σήματα εισόδου της Condlogic είναι το PCS του 1 bit στο οποίο διοχετεύεται η αντίστοιχη έξοδος του σήματος επιλογής ελέγχου της εισόδου του πολυπλέκτη διεύθυνσης επόμενης εντολής PCS του 1 bit, του αποκωδικοποιητή. Αξιοποιώντας αυτά τα RegW του 1 bit, MemW του 1 bit, FlagW των 2 bit, PCS του 1 bit και την λογική έξοδο Condex παράγονται τα κατάλληλα σήματα εξόδου RegWrite του 1 bit, MemWrite του 1 bit, τα οποία είναι σήματα έγκρισης εγγραφής για το Register file, την Data Memory αντίστοιχα καθώς και το σήμα εξόδου PCSrc του 1 bit το οποίο είναι σήμα επιλογής ελέγχου της εισόδου του πολυπλέκτη διεύθυνσης επόμενης εντολής. Επίσης παράγεται και ένα εσωτερικό σήμα έγκρισης εγγραφής FlagWrite των 2 bit για τον καταχωρητή κατάστασης. Τα σήματα εξόδου που περιεγράφηκαν παραπάνω αφού περάσουν πρώτα από το ανώτερο ιεραρχικά επίπεδο της μονάδας ελέγχου καταλήγουν κατευθείαν στη διαδρομή δεδομένων.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity condlogic is
port(clk, reset: in STD_LOGIC;
Cond: in STD_LOGIC_VECTOR(3 downto 0);
ALUFlags: in STD_LOGIC_VECTOR(3 downto 0);
FlagW: in STD_LOGIC_VECTOR(1 downto 0);
PCS, RegW, MemW: in STD_LOGIC;
PCSrc, RegWrite: out STD_LOGIC;
MemWrite: out STD_LOGIC);
end condlogic;

architecture Behavioral of condlogic is
component condcheck
port(Cond: in STD_LOGIC_VECTOR(3 downto 0);
Flags: in STD_LOGIC_VECTOR(3 downto 0);
CondEx: out STD_LOGIC);
end component;
component flopenr
generic(width: positive := 2);
port(clk, reset, en: in STD_LOGIC;
d: in STD_LOGIC_VECTOR (width-1 downto 0);
q: out STD_LOGIC_VECTOR (width-1 downto 0));
end component;
signal FlagWrite: STD_LOGIC_VECTOR(1 downto 0);
signal Flags: STD_LOGIC_VECTOR(3 downto 0);
signal CondEx: STD_LOGIC;

begin
flagreg1: flopenr generic map (width => 2)
port map(clk, reset, FlagWrite(1), ALUFlags(3 downto 2), Flags(3 downto 2));
flagreg0: flopenr  generic map (width => 2)
port map(clk, reset, FlagWrite(0), ALUFlags(1 downto 0), Flags(1 downto 0));


 cc: condcheck port map(Cond, Flags, CondEx);
 FlagWrite <= FlagW and (CondEx, CondEx);
 RegWrite <= RegW and CondEx;
 MemWrite <= MemW and CondEx;
 PCSrc <= PCS and CondEx;


end Behavioral;
```

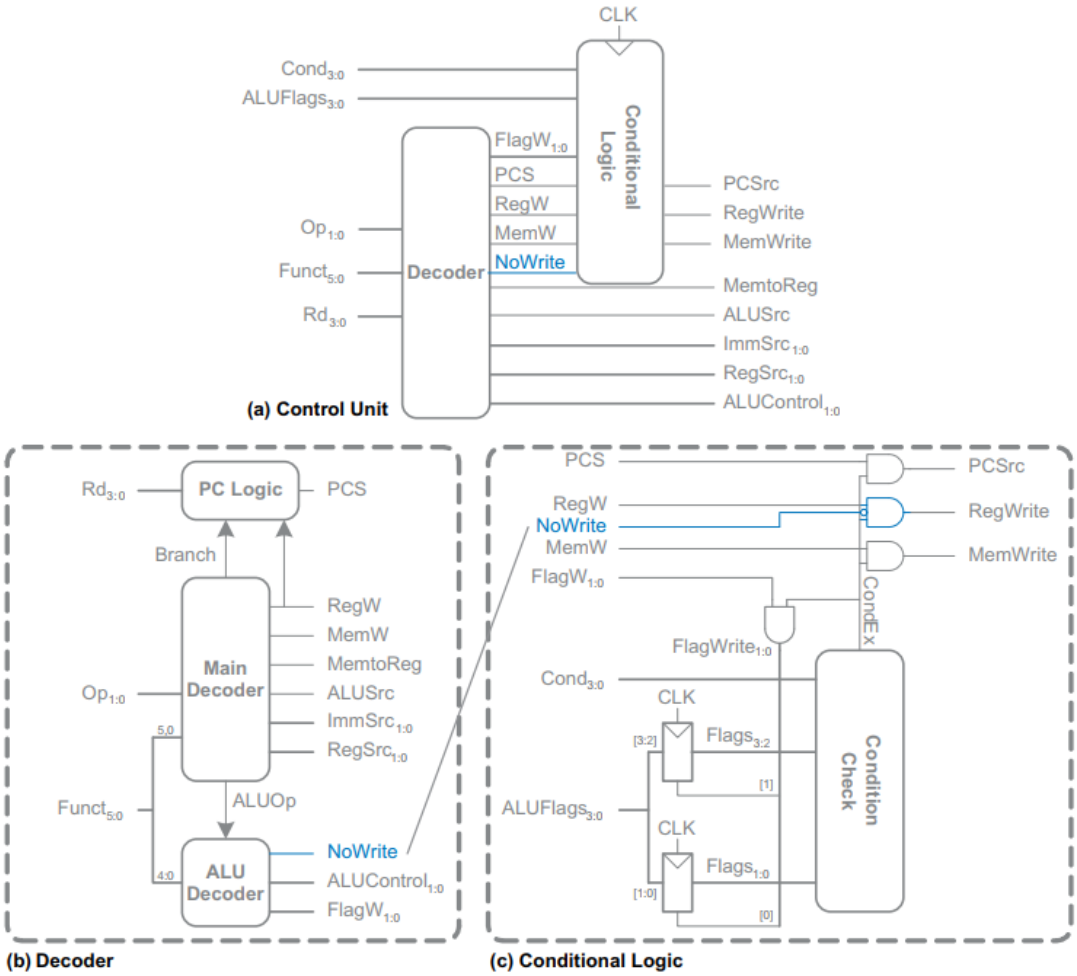
Εικόνα 29: Κώδικας Condlogic



Εικόνα 30: RTL Schematic Condcheck

Περιγραφή της δομή της στοιχείων της μονάδας ελέγχου (control unit or controller) του επεξεργαστή στα ανώτερα ιεραρχικά επίπεδα

Για την υλοποίηση της δομής της μονάδας ελέγχου της αρχιτεκτονικής ARM χρησιμοποιήθηκαν τα ψηφιακά δομικά στοιχεία (components) τα οποία περιεγράφηκαν παραπάνω. Πιο συγκεκριμένα υλοποιήθηκε η δομή της μονάδας ελέγχου που φαίνεται εικόνα που ακολουθεί αξιοποιώντας σε ένα ανώτερο ιεραρχικά επίπεδο τα ψηφιακά δομικά στοιχεία (components) προαναφέρθηκαν.



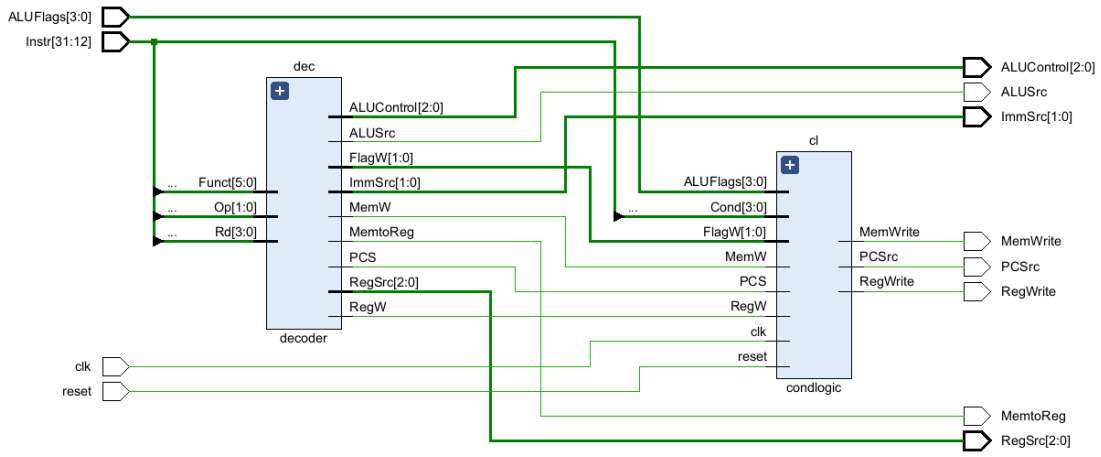
Εικόνα 31: Δομή Controller

Για την υλοποίηση σε γλώσσα περιγραφής υλικού VHDL του controller χρειάστηκαν μια λογική μονάδα ελέγχου συνθήκης (CondLogic) και ένας αποκωδικοποιητής (Decoder). Για την αρμονική και σωστή σύμφωνα με την αρχιτεκτονική ARM σύνδεση όλων αυτών των ψηφιακών δομικών στοιχείων χρησιμοποιήθηκαν σε γλώσσα περιγραφής υλικού VHDL αρκετά εσωτερικά σήματα που έχουν τα ονόματα που διαθέτουν τα 'καλώδια' που παρατηρούνται στην εικόνα που ακολουθεί. Ωστόσο πέρα από τα εσωτερικά σήματα που έχουν ως σκοπό την ένωση μεταξύ των ψηφιακών δομικών στοιχείων που χρησιμοποιεί η διαδρομή δεδομένων υπάρχουν και κάποια σήματα εισόδου της μονάδας διαδρομής δεδομένων. Σήματα εισόδου στην μονάδα ελέγχου όπως αυτά του clock (clk) ή του reset 1 bit χρησιμοποιούνται από την μονάδα για τον συγχρονισμό και την αρχικοποίηση αντίστοιχα των διαφόρων δομικών ψηφιακών μανάδων από τις οποίες αποτελείται η μονάδα ελέγχου. Από την άλλη μεριά σήματα εισόδου στην μονάδας ελέγχου όπως το σήμα Instr (20 bit) στο οποίο διοχετεύονται τα 20 σημαντικότερα bit από το σήμα εξόδου της διαδρομής δεδομένων Instr_out (32 bit) το οποίο περιέχει την τρέχουσα εντολή που βγαίνει από την έξοδο της instruction memory που αξιοποιούνται από την μονάδα ελέγχου σαν σήματα για την αποκωδικοποίηση της εντολής και τον έλεγχο της ικανοποίησης ή όχι της συνθήκης υπό την οποία πρέπει να εκτελεστεί η εντολή .

Επιπροσθέτως άλλο ένα σήμα εισόδου στην μονάδας ελέγχου είναι το σήμα ALUFlags (4bit) στο οποίο διοχετεύονται τα 4 bit από το σήμα εξόδου της διαδρομής δεδομένων ALUFlags (4bit) το οποίο περιέχει τις τρέχουσες σημαίες συνθήκης που παράγονται μετά από κάθε πράξη στην αριθμητική και λογική μονάδα και συγκεκριμένα το σήμα εξόδου της AluFlags. Σαν έξοδο από την μονάδα ελέγχου λαμβάνεται τα σήματα ελέγχου επιλογής εισόδου πολυπλεκτών RegSrc των 3 bit, ALUSrc του 1 bit και MemtoReg του 1 bit καθώς και τα σήματα ελέγχου λειτουργίας της αριθμητικής και λογικής μονάδας ALUControl των 3 bit και της μονάδας επέκτασης μηδενός ή πρόσημου ImmSrc των 2 bit, τα οποία σήματα διοχετεύονται από τα αντίστοιχα (με το ίδιο όνομα) σήματα εξόδου του αποκωδικοποιητή και θα καταλήξουν στην συνέχεια στην διαδρομή δεδομένων. Ακόμη υπάρχουν και τα σήματα εξόδου από την μονάδα ελέγχου RegWrite του 1 bit, MemWrite του 1 bit, τα οποία είναι σήματα έγκρισης εγγραφής για το Register file, την Data Memory αντίστοιχα καθώς και το σήμα εξόδου PCSrc του 1 bit το οποίο είναι σήμα επιλογής ελέγχου της εισόδου του πολυπλέκτη διεύθυνσης επόμενης εντολής, τα οποία σήματα διοχετεύονται από τα αντίστοιχα (με το ίδιο όνομα) σήματα εξόδου της λογικής μονάδας ελέγχου συνθήκης και θα καταλήξουν στην συνέχεια διαδρομή δεδομένων. Όπως είναι εύκολα παρατηρήσιμο η δομή της μονάδας ελέγχου που φαίνεται στην εικόνα παραπάνω ταιριάζει απόλυτα με τα προκύπτοντα σχηματικά διαγράμματα στο επίπεδο RTL της μονάδας ελέγχου.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity controller is
port(clk, reset: in STD_LOGIC;
Instr: in STD_LOGIC_VECTOR(31 downto 12);
ALUFlags: in STD_LOGIC_VECTOR(3 downto 0);
RegSrc: out STD_LOGIC_VECTOR(2 downto 0);
RegWrite: out STD_LOGIC;
ImmSrc: out STD_LOGIC_VECTOR(1 downto 0);
ALUSrc: out STD_LOGIC;
ALUControl: out STD_LOGIC_VECTOR(2 downto 0);
MemWrite: out STD_LOGIC;
MemtoReg: out STD_LOGIC;
PCSrc: out STD_LOGIC);
end controller;

architecture Behavioral of controller is
component decoder
port(Op: in STD_LOGIC_VECTOR(1 downto 0);
Funct: in STD_LOGIC_VECTOR(5 downto 0);
Rd: in STD_LOGIC_VECTOR(3 downto 0);
FlagW: out STD_LOGIC_VECTOR(1 downto 0);
PCS, RegW, MemW: out STD_LOGIC;
MemtoReg, ALUSrc: out STD_LOGIC;
ImmSrc : out STD_LOGIC_VECTOR(1 downto 0);
RegSrc : out STD_LOGIC_VECTOR(2 downto 0);
ALUControl: out STD_LOGIC_VECTOR(2 downto 0));
end component;
component condlogic
port(clk, reset: in STD_LOGIC;
Cond: in STD_LOGIC_VECTOR(3 downto 0);
ALUFlags: in STD_LOGIC_VECTOR(3 downto 0);
FlagW: in STD_LOGIC_VECTOR(1 downto 0);
PCS, RegW, MemW: in STD_LOGIC;
PCSrc, RegWrite: out STD_LOGIC;
MemWrite: out STD_LOGIC);
end component;
signal FlagW: STD_LOGIC_VECTOR(1 downto 0);
signal PCS, RegW, MemW: STD_LOGIC;

begin
dec: decoder port map(Instr(27 downto 26), Instr(25 downto 20), Instr(15 downto 12), FlagW, PCS, RegW, MemW, MemtoReg, ALUSrc, ImmSrc, RegSrc, ALUControl);
cl: condlogic port map(clk, reset, Instr(31 downto 28), ALUFlags, FlagW, PCS, RegW, MemW, PCSrc, RegWrite, MemWrite);

end Behavioral;
```

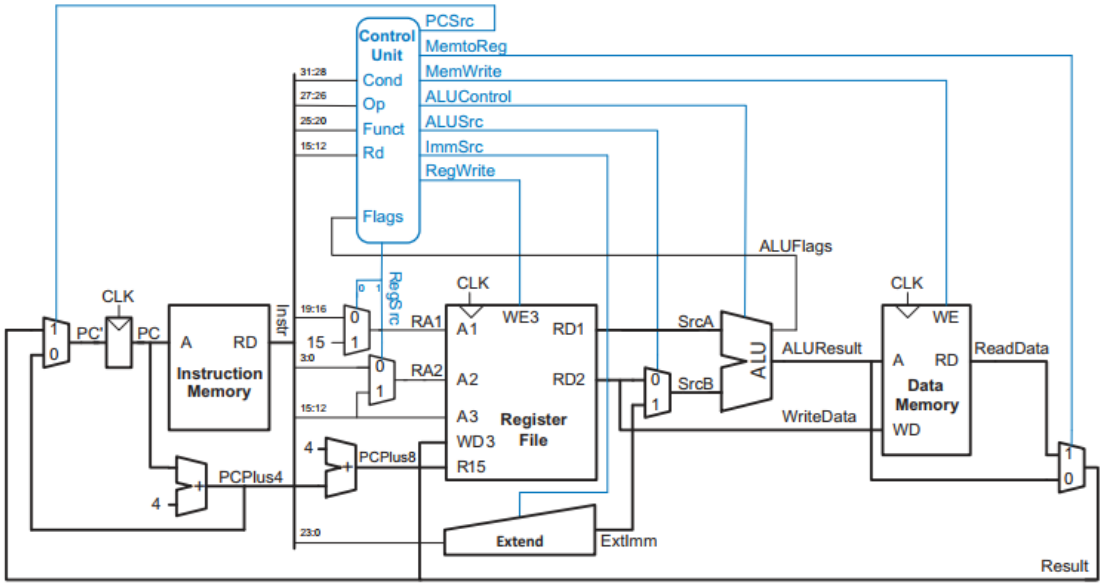
Εικόνα 32: Κώδικας Controller



Εικόνα 33: RTL Schematic Controller

Περιγραφή της δομή της στοιχείων του επεξεργαστή (processor) του επεξεργαστή στα ανώτερα ιεραρχικά επίπεδα

Για την υλοποίηση της δομής του επεξεργαστή της αρχιτεκτονικής ARM χρησιμοποιήθηκαν τις δύο κύριες υπομονάδες οι οποίες περιεγράφηκαν παραπάνω. Πιο συγκεκριμένα υλοποιήθηκε η δομή του επεξεργαστή που φαίνεται στην εικόνα που ακολουθεί αξιοποιώντας σε ένα ανώτερο ιεραρχικά επίπεδο τις δύο κύριες υπομονάδες προαναφέρθηκαν.



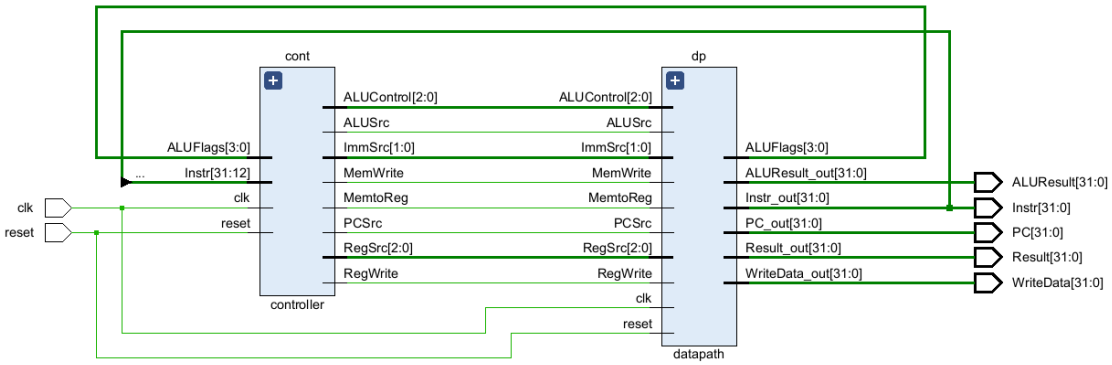
Εικόνα 34: Δομή CPU

Για την υλοποίηση σε γλώσσα περιγραφής υλικού VHDL του επεξεργαστή χρειάστηκε μια μονάδα ελέγχου συνθήκης (Control unit or Controller) και μια υπομονάδα διαδρομής δεδομένων (Datapath). Για την αρμονική και σωστή σύμφωνα με την αρχιτεκτονική ARM σύνδεση όλων αυτών των ψηφιακών δομικών στοιχείων χρησιμοποιήθηκαν σε γλώσσα περιγραφής υλικού VHDL αρκετά εσωτερικά σήματα που έχουν τα ονόματα που διαθέτουν τα ‘καλώδια’ που παρατηρούνται στην εικόνα που ακολουθεί. Ωστόσο πέρα από τα εσωτερικά σήματα που έχουν ως σκοπό την ένωση μεταξύ των ψηφιακών δομικών στοιχείων που χρησιμοποιεί η διαδρομή δεδομένων υπάρχουν και κάποια σήματα εισόδου της μονάδας διαδρομής δεδομένων.

Σήματα εισόδου στην μονάδα ελέγχου όπως αυτά του clock (clk) ή του reset 1 bit χρησιμοποιούνται από την μονάδα για τον συγχρονισμό και την αρχικοποίηση αντίστοιχα των διαφόρων ψηφιακών υπομονάδων από τις οποίες αποτελείται ο επεξεργαστής. Σαν έξοδο από την επεξεργαστή λαμβάνεται το σήμα PC (32 bit) που διοχετεύεται από το σήμα εξόδου PC_out (32 bit) της διαδρομής δεδομένων που περιέχει την διεύθυνση της τρέχουσας εντολής στην Instruction Memory, το σήμα Instr (32 bit) που διοχετεύεται από το σήμα εξόδου Instr_out (32 bit) της διαδρομής δεδομένων που περιέχει την τρέχουσα εντολή που βγαίνει από την έξοδο της instruction memory, το σήμα ALUResult (32 bit) που διοχετεύεται από το σήμα εξόδου ALUResult_out της διαδρομής δεδομένων (32 bit) που περιέχει το αποτέλεσμα που προκύπτει από την εκάστοτε πράξη στην έξοδο της ALU, το σήμα WriteData (32 bit) που διοχετεύεται από το σήμα εξόδου WriteData_out(32 bit) της διαδρομής δεδομένων που περιέχει τα δεδομένα που προέρχονται από την θύρα RD2 (32 bit) του Register file η οποία πραγματοποιεί ανάγνωση των δεδομένων του καταχωρητή που υποδεικνύει η θύρα A2(4 bit) και το σήμα Result(32 bit) που διοχετεύεται από το σήμα εξόδου Result_out(32 bit) της διαδρομής δεδομένων που περιέχει είτε το αποτέλεσμα που προκύπτει από την εκάστοτε πράξη στην έξοδο της ALU είτε τα δεδομένα που πραγματοποιείται ανάγνωση που προέρχονται από την θύρα RD (32 bit) της Data memory η οποία πραγματοποιεί ανάγνωση των δεδομένων της διεύθυνσης που υποδεικνύει η θύρα A(4 bit) ανάλογα την τιμή του σήματος ελέγχου επιλογής εισόδου του πολυπλέκτη που ως σήμα ελέγχου επιλογής εισόδου λαμβάνει το σήμα MemtoReg που προέρχεται από την μονάδα ελέγχου. Όπως είναι εύκολα παρατηρήσιμο η δομή του επεξεργαστή που φαίνεται στην εικόνα παραπάνω ταιριάζει απόλυτα τα προκύπτοντα σχηματικά διαγράμματα στο επίπεδο RTL της διαδρομής δεδομένων με την διαφορά πως δεν υπάρχουν οι έξοδοι PC, Instr, ALUResult, WriteData και Result που ουσιαστικά τοποθετήθηκαν έτσι ώστε να ελεγχθεί η σωστή λειτουργία του επεξεργαστή.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity arm is -- single cycle processor
port(clk, reset: in STD_LOGIC;
PC: out STD_LOGIC_VECTOR(31 downto 0);
Instr: out STD_LOGIC_VECTOR(31 downto 0);
ALUResult:out STD_LOGIC_VECTOR(31 downto 0);
WriteData:out STD_LOGIC_VECTOR(31 downto 0);
Result:out STD_LOGIC_VECTOR(31 downto 0));
end arm;

architecture Behavioral of arm is
component controller
port(clk, reset: in STD_LOGIC;
Instr: in STD_LOGIC_VECTOR(31 downto 12);
ALUFlags: in STD_LOGIC_VECTOR(3 downto 0);
RegSrc: out STD_LOGIC_VECTOR(2 downto 0);
RegWrite: out STD_LOGIC;
ImmSrc: out STD_LOGIC_VECTOR(1 downto 0);
ALUSrc: out STD_LOGIC;
ALUControl: out STD_LOGIC_VECTOR(2 downto 0);
MemWrite: out STD_LOGIC;
MemtoReg: out STD_LOGIC;
PCSrc: out STD_LOGIC);
end component;
component datapath
port(clk, reset: in STD_LOGIC;
RegSrc: in STD_LOGIC_VECTOR(2 downto 0);
RegWrite: in STD_LOGIC;
ImmSrc: in STD_LOGIC_VECTOR(1 downto 0);
ALUSrc: in STD_LOGIC;
ALUControl: in STD_LOGIC_VECTOR(2 downto 0);
MemtoReg: in STD_LOGIC;


PCSrc: in STD_LOGIC;
ALUFlags: out STD_LOGIC_VECTOR(3 downto 0);
MemWrite : in std_logic;
PC_out: out STD_LOGIC_VECTOR(31 downto 0);
Instr_out: out STD_LOGIC_VECTOR(31 downto 0);
ALUResult_out:out STD_LOGIC_VECTOR(31 downto 0);
WriteData_out:out STD_LOGIC_VECTOR(31 downto 0);
Result_out:out STD_LOGIC_VECTOR(31 downto 0));
end component;
signal RegWrite, ALUSrc, MemtoReg, PCSrc, MemWrite: STD_LOGIC;
signal ImmSrc: STD_LOGIC_VECTOR (1 downto 0);
signal RegSrc, ALUControl : STD_LOGIC_VECTOR (2 downto 0);
signal ALUFlags: STD_LOGIC_VECTOR(3 downto 0);
signal Instr_aux: STD_LOGIC_VECTOR(31 downto 0);

begin
cont: controller port map(clk, reset, Instr_aux(31 downto 12), ALUFlags, RegSrc, RegWrite, ImmSrc, ALUSrc, ALUControl, MemWrite, MemtoReg, PCSrc);
dp: datapath port map(clk, reset, RegSrc, RegWrite, ImmSrc, ALUSrc, ALUControl, MemtoReg, PCSrc, ALUFlags, MemWrite, PC, Instr_aux, ALUResult, WriteData, Result);
Instr<=Instr_aux;
end Behavioral;
```

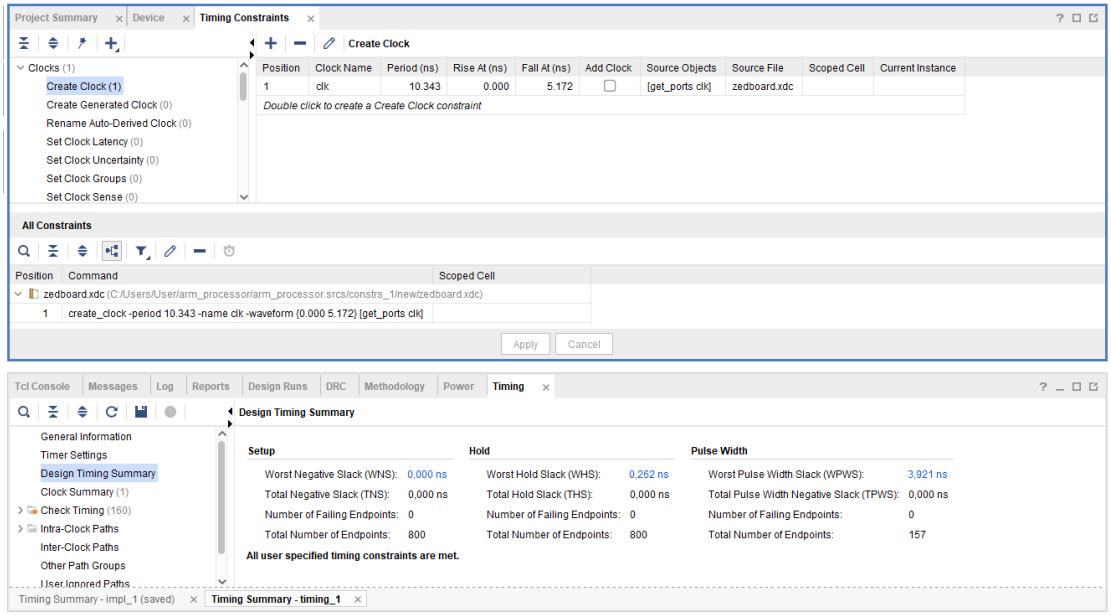
Εικόνα 35: Κώδικας CPU



Εικόνα 36: RTL Schematic CPU

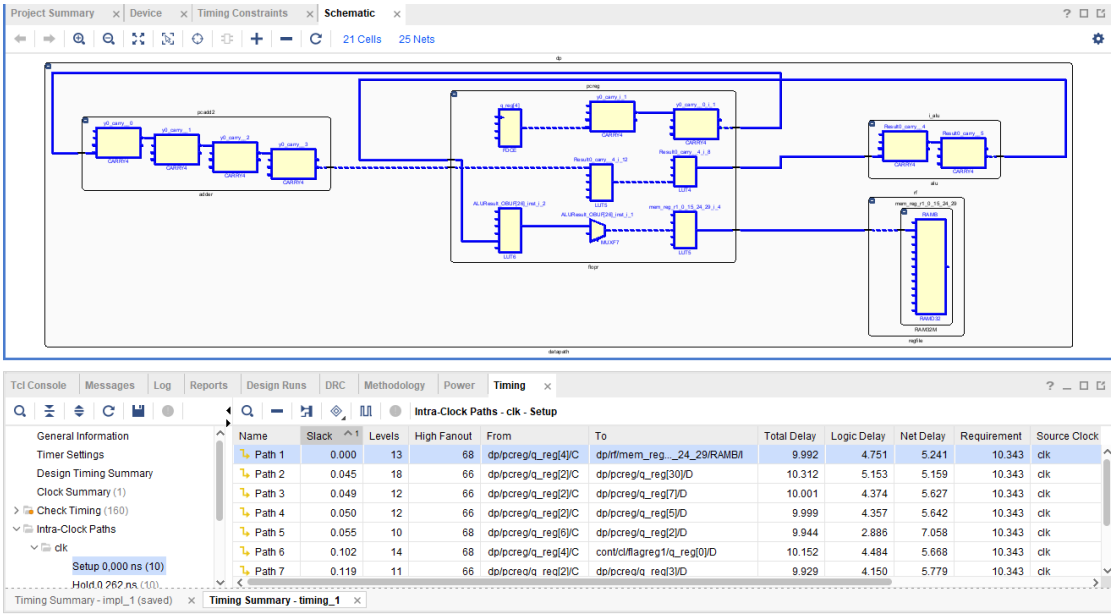
Ποια είναι η μέγιστη συχνότητα λειτουργίας στο επίπεδο του επεξεργαστή (processor), καθώς και ποιά η χειρότερη κρίσιμη διαδρομή και τη χειρότερη σύντομη διαδρομή.

Έχοντας κάνει τα βήματα της synthesis και του implementation η μέγιστη συχνότητα λειτουργίας του κυκλώματος επιτυγχάνεται όταν το περιθώριο (Slack) του χειρότερου μονοπατιού με βάση τον χρόνο setup (path 1) προσεγγίζει το μηδέν ή και ακόμα γίνει μηδέν χωρίς να έχουμε κανένα αριθμό από failing endpoints. Άρα η μέγιστη περίοδος λειτουργίας παρατηρούμε πως είναι 10.343 ns και η μέγιστη συχνότητα λειτουργίας $0{,}966 \times 10^8$ Hz.



Εικόνα 37: Η μέγιστη περίοδος λειτουργίας έχει την τιμή 10.343 ns

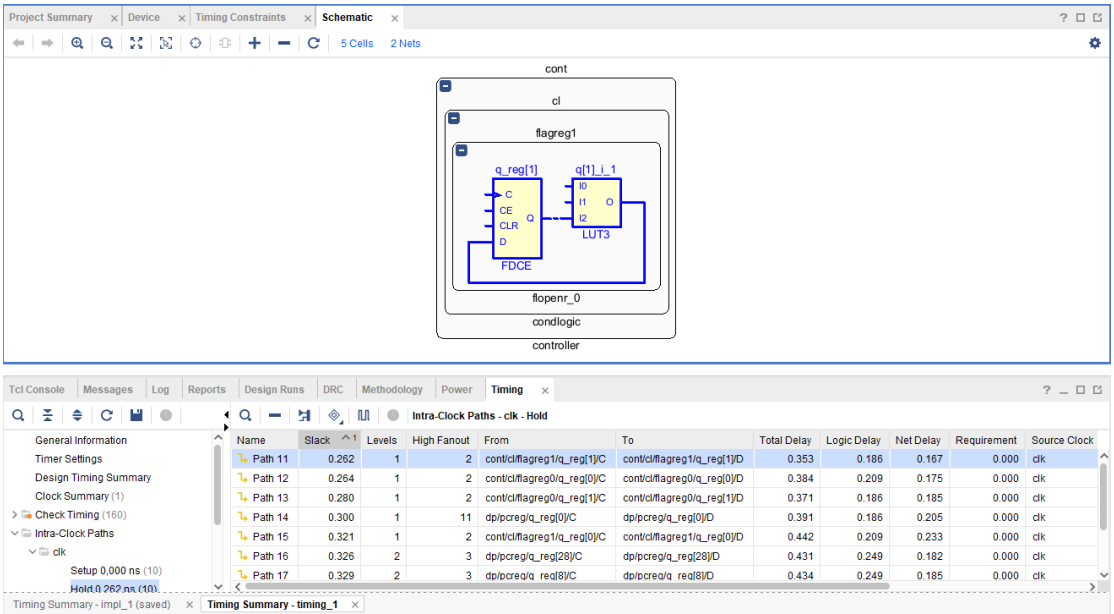
Η χειρότερη κρίσιμη διαδρομή ταυτίζεται με το χειρότερο μονοπάτι με βάση το περιθώριο (Slack) για τον χρόνο setup (path 1)



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	0.000	13	68	dp/pcreg/q_reg[4]/C	dp/rf/mem_reg..._24_29/RAMB/I	9.992	4.751	5.241	10.343	clk
Path 2	0.045	18	66	dp/pcreg/q_reg[2]/C	dp/pcreg/q_reg[30]/D	10.312	5.153	5.159	10.343	clk
Path 3	0.049	12	66	dp/pcreg/q_reg[2]/C	dp/pcreg/q_reg[7]/D	10.001	4.374	5.627	10.343	clk
Path 4	0.050	12	66	dp/pcreg/q_reg[2]/C	dp/pcreg/q_reg[5]/D	9.999	4.357	5.642	10.343	clk
Path 5	0.055	10	68	dp/pcreg/q_reg[6]/C	dp/pcreg/q_reg[2]/D	9.944	2.886	7.058	10.343	clk
Path 6	0.102	14	68	dp/pcreg/q_reg[4]/C	cont/cl/flagreg1/q_reg[0]/D	10.152	4.484	5.668	10.343	clk
Path 7	0.119	11	66	dp/pcreg/q_reg[2]/C	dp/pcreg/q_reg[3]/D	9.929	4.150	5.779	10.343	clk

Εικόνα 38: Η χειρότερη κρίσιμη διαδρομή ταυτίζεται με το χειρότερο μονοπάτι με βάση το περιθώριο (Slack) για τον χρόνο setup (path 1)

Η χειρότερη σύντομη διαδρομή ταυτίζεται με το χειρότερο μονοπάτι με βάση το περιθώριο (Slack) για τον χρόνο hold (path 11)



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	0.262	1	2	cont/cl/flagreg1/q_reg[1]/C	cont/cl/flagreg1/q_reg[1]/D	0.353	0.186	0.167	0.000	clk
Path 12	0.264	1	2	cont/cl/flagreg0/q_reg[0]/C	cont/cl/flagreg0/q_reg[0]/D	0.384	0.209	0.175	0.000	clk
Path 13	0.280	1	2	cont/cl/flagreg0/q_reg[1]/C	cont/cl/flagreg0/q_reg[1]/D	0.371	0.186	0.185	0.000	clk
Path 14	0.300	1	11	dp/pcreg/q_reg[0]/C	dp/pcreg/q_reg[0]/D	0.391	0.186	0.205	0.000	clk
Path 15	0.321	1	2	cont/cl/flagreg1/q_reg[0]/C	cont/cl/flagreg1/q_reg[0]/D	0.442	0.209	0.233	0.000	clk
Path 16	0.326	2	3	dp/pcreg/q_reg[28]/C	dp/pcreg/q_reg[28]/D	0.431	0.249	0.182	0.000	clk
Path 17	0.329	2	3	dp/pcreg/q_reg[8]/C	dp/pcreg/q_reg[8]/D	0.434	0.249	0.185	0.000	clk

Εικόνα 39: Η χειρότερη σύντομη διαδρομή ταυτίζεται με το χειρότερο μονοπάτι με βάση το περιθώριο (Slack) για τον χρόνο hold (path 11)

Επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή (processor).

Περιγραφή της διαδρομής που ακολουθούν οι εντολές στην διαδρομή δεδομένων

Καταρχήν κοινή λειτουργία όλων των εντολών είναι πως ο μετρητής προγράμματος PC (ο οποίος προέρχεται από τον καταχωρητή μετρητής προγράμματος PC) που περιέχει τη διεύθυνση της τρέχουσας εντολής διοχετεύεται στην είσοδο A της μνήμης εντολών και στην έξοδο Rd της μνήμης εντολών διοχετεύεται η εντολή που βρίσκεται στην διεύθυνση που 'έδειξε' ο μετρητής προγράμματος. Επίσης κοινή λειτουργία όλων των εντολών είναι πως υπολογίζεται ταυτόχρονα με την εκτέλεση της τρέχουσας εντολής η διεύθυνση της αμέσως επόμενης εντολής, δηλαδή ο μετρητής προγράμματος PC+4 μέσω ενός αθροιστή κατά 4 και το αποτέλεσμα διοχετεύεται στον πολυπλέκτη επιλογής διεύθυνσης επόμενης εντολής. Έπειτα έχουμε τις παρακάτω μορφές εντολών:

1. εντολές επεξεργασίας δεδομένων ADD(S), SUB(S), AND(S), OR(S), XOR(S), MOV(S), MVN (S), LSL(S), LSR(S), ASR(S), ROR(S).
   Για τις εντολές αυτές αφού ληφθεί η εντολή από την μνήμη εντολών, το πεδίο Rn της εντολής διοχετεύεται στην είσοδο ανάγνωσης A1 του Register file έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης Rn του αρχείου καταχωρητών (R0-R14) από την θύρα ανάγνωσης RD1. Αν το πεδίο Rn της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προέλευσης δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε διαβιβάζεται στην θύρα ανάγνωσης RD1 η διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της διεύθυνσης που λαμβάνουν. Τα σήματα ελέγχου επιλογής εισόδου πολυπλέκτη RegScr πρέπει να λάβουν την τιμή RegScr[2:0]=000. Αν ο δεύτερος τελεστής προέλευσης είναι άμεσος τότε το πεδίο src2 της εντολής περνά από τη μονάδα Extend όπου εκτελείται επέκταση μηδενός στα 32 bit του μη προσημασμένου άμεσου τελεστέου των 8 bit και το αποτέλεσμα εμφανίζεται στην έξοδο του Extend την ExtImm. Το σήμα ελέγχου είδους επέκτασης λαμβάνει την τιμή Immsrc=00. Αν ο δεύτερος τελεστής προέλευσης είναι έμμεσος τότε το υποπεδίο Rm του πεδίου src2 της εντολής περνά στην είσοδο ανάγνωσης A2 του Register file έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης Rm του αρχείου καταχωρητών (R0-R14) από την θύρα ανάγνωσης RD2. Αν το πεδίο Rm της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προέλευσης δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε διαβιβάζεται στην θύρα ανάγνωσης RD2 η διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της που λαμβάνουν.

Έπειτα αν έχουμε άμεσο τελεστέου το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέο πρέπει να λάβει την τιμή ALUSrc= 1 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της θύρας εξόδου ExtImm του Extend. Αντίθετα αν έχουμε έμμεσο τελεστέο το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέο πρέπει να λάβει την τιμή ALUSrc= 0 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της από την θύρα ανάγνωσης του Register file RD2. Στην συνέχεια στην ALU ανάλογα το σήμα ελέγχου (ALUControl) των 3 bit που προέρχεται από την μονάδα ελέγχου και το οποίο θα ελέγχει το είδος της πράξης που θα πραγματοποιείται μεταξύ των λέξεων που θα βρεθούν στις εισόδους (α), (b) της ALU. Αν ALUControl =’000’ πραγματοποιείται η πράξη της πρόσθεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’001’ πραγματοποιείται η πράξη της αφαίρεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’010’ πραγματοποιείται η λογική πράξη and μεταξύ των δύο ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’011’ πραγματοποιείται η λογική πράξη or μεταξύ των δύο ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b), αν ALUControl =’100’ πραγματοποιείται η λογική πράξη xor μεταξύ των δύο ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b) που χρησιμοποιείται από την εντολή EOR. Επιπροσθέτως αν ALUControl =’101’ πραγματοποιείται ένας από τους τέσσερεις τύπους ολίσθησης στον ακέραιο που αναπαρίσταται στην λέξη που λαμβάνεται από την είσοδο (b). Ο τύπος της ολίσθησης που θα πραγματοποιηθεί, καθορίζεται από ένα σήμα ελέγχου (sh) (προέρχεται από το πεδίο sh της εντολής) των 2 bit που εισέρχεται στην ALU και έτσι αν sh=’00’ πραγματοποιείται λογική ολίσθηση προς τα αριστερά της λέξης που λαμβάνεται από την είσοδο (b), αν sh=’01’ πραγματοποιείται λογική ολίσθηση προς τα δεξιά της λέξης που λαμβάνεται από την είσοδο (b), αν sh=’10’ πραγματοποιείται αριθμητική ολίσθηση προς τα δεξιά της λέξης που λαμβάνεται από την είσοδο (b) ενώ αν sh=’11’ πραγματοποιείται περιστροφή προς τα δεξιά της λέξης που λαμβάνεται από την είσοδο (b). Η σταθερή ποσότητα ολίσθησης, της λέξης που λαμβάνεται από την είσοδο (b), καθορίζεται από ένα σήμα εισόδου (shamt5) (προέρχεται από το πεδίο sh της εντολής) των 5 bit που εισέρχεται στην ALU. Ακόμη αν ALUControl =’111’ πραγματοποιείται λογική πράξη not για την λέξη που λαμβάνεται από την είσοδο (b) που χρησιμοποιείται από την εντολή MVN ενώ αν ALUControl =’110’ θα μπορούσε να πει κάποιος πως πραγματοποιείται λογικά η πράξη της πρόσθεσης της λέξης που λαμβάνεται από την είσοδο (b) με τον αριθμό ‘0’ που ουσιαστικά σαν αποτέλεσμα έχει την ίδια λέξη που προέρχεται από την είσοδο (b) που χρησιμοποιείται από την εντολή MOV. Από την έξοδο της ALU λαμβάνεται ένα σήμα εξόδου (Result) των 32 bit το οποίο περιλαμβάνει το αποτέλεσμα από την όποια πράξη πραγματοποιείται σε κάθε περίπτωση αναλόγως τα σήματα (ALUControl) και (sh) το οποίο στην συνέχεια θα πρέπει να περάσει από ένα πολυπλέκτη του οποίου το σήμα ελέγχου επιλογής εισόδου MemtoReg πρέπει να λάβει την τιμή MemtoReg=0 και συνεχίζοντας για να γίνει εγγραφή στον καταχωρητή προορισμού Rd το πεδίο Rd της εντολής διοχετεύεται στην είσοδο A3 του Register file έτσι ώστε να γίνει εγγραφή του δεδομένου στον καταχωρητή προορισμού Rd του αρχείου καταχωρητών (R0-R14) από την θύρα εγγραφής WD3 έχοντας βέβαια και το σήμα έγκρισης εγγραφής στο Register file RegWrite να λάβει τη τιμή RegWrite =1. Αν το πεδίο Rd της εντολής λάβει

την τιμή 15, δηλαδή ως καταχωρητής προορισμού δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε πρέπει το σήμα έγκρισης εγγραφής στο Register file RegWrite να λάβει τη τιμή RegWrite =0 για να μην γίνει εγγραφή και το δεδομένο να διαβιβαστεί στον πολυπλέκτη επιλογής διεύθυνσης επόμενης εντολής του οποίου αν το σήμα ελέγχου επιλογής εισόδου PCSrc λάβει την τιμή PCSrc=1 η διεύθυνση της επόμενης εντολής θα είναι αυτή που έχει το δεδομένο ενώ αν PCSrc=0 τότε η διεύθυνση της αμέσως επόμενης εντολής θα είναι ο μετρητής προγράμματος προσαυξημένος κατά τέσσερα δηλαδή ο PC+4.Εάν το πεδίο S της εντολής είναι 1, τότε ενημερώνεται ο καταχωρητής κατάστασης που βρίσκεται στη μονάδα ελέγχου με τις νέες σημαίες συνθήκης N, Z, C, V (C = 0 και V = 0 στις λογικές πράξεις) που παράγονται από πράξεις στην ALU στην επόμενη ακμή του CLK αν ικανοποιούνται και οι συνθήκες εκτέλεσης της εντολής.

2. Εντολή επεξεργασίας δεδομένων CMP.
   Για την εντολή αυτή αφού ληφθεί η εντολή από την μνήμη εντολών, το πεδίο Rn της εντολής διοχετεύεται στην είσοδο ανάγνωσης A1 του Register file έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης Rn του αρχείου καταχωρητών (R0-R14) από την θύρα ανάγνωσης RD1. Αν το πεδίο Rn της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προέλευσης δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε διαβιβάζεται στην θύρα ανάγνωσης RD1 η διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της διεύθυνσης που λαμβάνουν. Τα σήματα ελέγχου επιλογής εισόδου πολυπλέκτη RegScr πρέπει να λάβουν την τιμή RegScr[2:0]=000. Αν ο δεύτερος τελεστής προέλευσης είναι άμεσος τότε το πεδίο src2 της εντολής περνά από τη μονάδα Extend όπου εκτελείται επέκταση μηδενός στα 32 bit του μη προσημασμένου άμεσου τελεστέου των 8 bit και το αποτέλεσμα εμφανίζεται στην έξοδο του Extend την ExtImm. Το σήμα ελέγχου είδους επέκτασης λαμβάνει την τιμή Immsrc=00. Αν ο δεύτερος τελεστής προέλευσης είναι έμμεσος τότε το υποπεδίο Rm του πεδίου src2 της εντολής περνά στην είσοδο ανάγνωσης A2 του Register file έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης Rm του αρχείου καταχωρητών (R0-R14) από την θύρα ανάγνωσης RD2. Αν το πεδίο Rm της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προέλευσης δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε διαβιβάζεται στην θύρα ανάγνωσης RD2 η διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της που λαμβάνουν. Έπειτα αν έχουμε άμεσο τελεστέου το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέου πρέπει να λάβει την τιμή ALUSrc= 1 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της θύρας εξόδου ExtImm του Extend. Αντίθετα αν έχουμε έμμεσο τελεστέου το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέου πρέπει να λάβει την τιμή ALUSrc= 0 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της από την θύρα ανάγνωσης του Register file RD2.

Στην συνέχεια στην ALU το σήμα ελέγχου (ALUControl) των 3 bit που προέρχεται από την μονάδα ελέγχου και το οποίο θα ελέγχει το είδος της πράξης που θα πραγματοποιείται μεταξύ των λέξεων που θα βρεθούν στις εισόδους (α), (b) της ALU, θα πάρει την τιμή ALUControl =’001’ ώστε να πραγματοποιηθεί η πράξη της αφαίρεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b). Από την έξοδο της ALU λαμβάνεται ένα σήμα εξόδου (Result) των 32 bit το οποίο περιλαμβάνει το αποτέλεσμα από την πράξη της αφαίρεσης το οποίο από σχεδιαστική επιλογή για την συγκεκριμένη στην συνέχεια θα περάσει από ένα πολυπλέκτη του οποίου το σήμα ελέγχου επιλογής εισόδου MemtoReg πρέπει να λάβει την τιμή MemtoReg=0 και συνεχίζοντας επειδή κατά την συγκεκριμένη εντολή το αποτέλεσμα δεν πρέπει να εγγραφεί στο register file αφού το συγκεκριμένο σήμα δεν θα χρησιμοποιηθεί περαιτέρω για την λειτουργία της εντολής, δεν θα πρέπει να πραγματοποιηθεί εγγραφή στον καταχωρητή προορισμού Rd από το πεδίο Rd της εντολής που διοχετεύεται στην είσοδο ανάγνωσης A3 του Register file έτσι ώστε να γίνει εγγραφή του δεδομένου στον καταχωρητή προορισμού Rd του αρχείου καταχωρητών (R0-R14) από την θύρα εγγραφής WD3, επομένως θα πρέπει το σήμα έγκρισης εγγραφής στο Register file RegWrite λάβει την τιμή RegWrite =0. Επίσης κατά την εκτέλεση αυτής της εντολής το σήμα ελέγχου επιλογής εισόδου PCSrc λαμβάνει πάντα την τιμή PCSrc=0 ώστε η διεύθυνση της αμέσως επόμενης εντολής να είναι πάντα ο μετρητής προγράμματος προσαυξημένος κατά τέσσερα δηλαδή ο PC+4. Το πεδίο S της εντολής πάντα έχει την τιμή 1, επομένως ενημερώνεται ο καταχωρητής κατάστασης που βρίσκεται στη μονάδα ελέγχου με τις νέες σημαίες συνθήκης N, Z, C, V (C = 0 και V = 0 στις λογικές πράξεις) που παράγονται από πράξεις στην ALU στην επόμενη ακμή του CLK αν ικανοποιούνται και οι συνθήκες εκτέλεσης της εντολής.

3. Εντολή μνήμης LDR.

   Για την εντολή αυτή αφού ληφθεί η εντολή από την μνήμη εντολών, το πεδίο Rn της εντολής διοχετεύεται στην είσοδο ανάγνωσης A1 του Register file έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης Rn του αρχείου καταχωρητών (R0-R14) από την θύρα ανάγνωσης RD1. Αν το πεδίο Rn της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προέλευσης δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος, τότε στην θύρα ανάγνωσης RD1 διαβιβάζεται η διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της διεύθυνσης που λαμβάνουν. Τα σήματα ελέγχου επιλογής εισόδου πολυπλέκτη RegScr πρέπει να λάβουν την τιμή RegScr[2:0]=000. Αν ο δεύτερος τελεστέος προέλευσης είναι άμεσος άρα το πεδίο src2 της εντολής περνά από τη μονάδα Extend όπου εκτελείται επέκταση μηδενός στα 32 bit του μη προσημασμένου άμεσου τελεστέου των 12 bit και το αποτέλεσμα εμφανίζεται στην έξοδο του Extend την ExtImm. Το σήμα ελέγχου είδους επέκτασης λαμβάνει την τιμή Immsrc=01. Έπειτα επειδή έχουμε άμεσο τελεστέο το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέο πρέπει να λάβει την τιμή ALUSrc= 1 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της θύρας εξόδου ExtImm του Extend. Στην συνέχεια στην ALU ανάλογα το σήμα ελέγχου (ALUControl) των 3 bit που προέρχεται από την μονάδα ελέγχου και το οποίο θα ελέγχει το είδος της πράξης (πρόσθεση ή αφαίρεση) που θα πραγματοποιείται μεταξύ των λέξεων που θα βρεθούν στις εισόδους (α), (b) της ALU.

Αν ALUControl =’000’ πραγματοποιείται η πράξη της πρόσθεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b) ενώ αν ALUControl =’001’ πραγματοποιείται η πράξη της αφαίρεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b). Από την έξοδο της ALU λαμβάνεται ένα σήμα εξόδου (ALUResult) των 32 bit το οποίο περιλαμβάνει το αποτέλεσμα, από την όποια πράξη (πρόσθεση ή αφαίρεση) πραγματοποιείται σε κάθε περίπτωση αναλόγως τα σήματα (ALUControl), από το οποίο σήμα στην συνέχεια τα bit 2 έως 7 ALUResult[7:2] θα περαστούν στην είσοδο της μνήμης δεδομένων (A) ως διεύθυνση 6 bit όπου υποδεικνύει εσωτερικά στην μνήμη δεδομένων την θέση που πρέπει διαβαστεί ένα δεδομένο των 32 bit μέσω της θύρας read data (RD) (32 bit) ασύγχρονα χωρίς την έγκριση κάποιου σήματος ενώ το σήμα έγκρισης εγγραφής στο Data memory Memwrite έχει τιμή MemWrite =0. Έπειτα το δεδομένο περνά από ένα πολυπλέκτη του οποίου το σήμα ελέγχου επιλογής εισόδου MemtoReg πρέπει να λάβει την τιμή MemtoReg=1 και συνεχίζοντας για να γίνει εγγραφή στον καταχωρητή προορισμού Rd το πεδίο Rd της εντολής διοχετεύεται στην είσοδο A3 του Register file έτσι ώστε να γίνει εγγραφή του δεδομένου στον καταχωρητή προορισμού Rd του αρχείου καταχωρητών (R0-R14) από την θύρα εγγραφής WD3 θα πρέπει το σήμα έγκρισης εγγραφής στο Register file RegWrite να λάβει την τιμή RegWrite =1. Αν το πεδίο Rd της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προορισμού δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε πρέπει το σήμα έγκρισης εγγραφής στο Register file RegWrite να λάβει την τιμή RegWrite =0 για να μην γίνει εγγραφή και το δεδομένο να διαβιβαστεί στον πολυπλέκτη επιλογής διεύθυνσης επόμενης εντολής του οποίου αν το σήμα ελέγχου επιλογής εισόδου PCSrc λάβει την τιμή PCSrc=1 η διεύθυνση της επόμενης εντολής θα είναι αυτή που έχει το δεδομένο ενώ αν PCSrc=0 τότε η διεύθυνση της αμέσως επόμενης εντολής θα είναι ο μετρητής προγράμματος προσαυξημένος κατά τέσσερα δηλαδή ο PC+4. Ανεξάρτητα από την τιμή του πεδίο S της εντολής, δεν ενημερώνεται ο καταχωρητής κατάστασης που βρίσκεται στη μονάδα ελέγχου με τις νέες σημαίες συνθήκης N, Z, C, V (C = 0 και V = 0 στις λογικές πράξεις) που παράγονται από πράξεις στην ALU στην επόμενη ακμή του CLK.

4. Εντολή μνήμης STR.
   Για την εντολή αυτή αφού ληφθεί η εντολή από την μνήμη εντολών, το πεδίο Rn της εντολής διοχετεύεται στην είσοδο ανάγνωσης A1 του Register file έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης Rn του αρχείου καταχωρητών (R0-R14) από την θύρα ανάγνωσης RD1. Αν το πεδίο Rn της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προέλευσης δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε διαβιβάζεται στην θύρα ανάγνωσης RD1 η διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της διεύθυνσης που λαμβάνουν. Το σήμα ελέγχου επιλογής εισόδου πολυπλέκτη RegScr πρέπει να λάβουν την τιμή RegScr[2:0]=010. Ο δεύτερος τελεστέος προέλευσης είναι άμεσος άρα το πεδίο src2 της εντολής περνά από τη μονάδα Extend όπου εκτελείται επέκταση μηδενός στα 32 bit του μη προσημασμένου άμεσου τελεστέου των 12 bit και το αποτέλεσμα εμφανίζεται στην έξοδο του Extend την ExtImm.

Το σήμα ελέγχου είδους επέκτασης λαμβάνει την τιμή Immsrc=01. Επίσης το υποπεδίο Rd της εντολής περνά στην είσοδο ανάγνωσης A2 του Register file έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης Rd του αρχείου καταχωρητών (R0-R14) από την θύρα ανάγνωσης RD2 του οποίου το περιεχόμενο στην συνέχεια θα αποθηκευτεί στην Data memory. Αν το πεδίο Rd της εντολής λάβει την τιμή 15, δηλαδή ως καταχωρητής προέλευσης δηλωθεί ο R15 που είναι ο καταχωρητής μετρητή προγράμματος τότε διαβιβάζεται στην θύρα ανάγνωσης RD2 η διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της που λαμβάνουν. Έπειτα επειδή έχουμε άμεσο τελεστέο το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέο πρέπει να λάβει την τιμή ALUSrc= 1 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της θύρας εξόδου ExtImm του Extend. Στην συνέχεια στην ALU ανάλογα το σήμα ελέγχου (ALUControl) των 3 bit που προέρχεται από την μονάδα ελέγχου και το οποίο θα ελέγχει το είδος της πράξης (πρόσθεση ή αφαίρεση) που θα πραγματοποιείται μεταξύ των λέξεων που θα βρεθούν στις εισόδους (α), (b) της ALU. Αν ALUControl =’000’ πραγματοποιείται η πράξη της πρόσθεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b) ενώ αν ALUControl =’001’ πραγματοποιείται η πράξη της αφαίρεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b). Από την έξοδο της ALU λαμβάνεται ένα σήμα εξόδου (ALUResult) των 32 bit το οποίο περιλαμβάνει το αποτέλεσμα, από την όποια πράξη (πρόσθεση ή αφαίρεση) πραγματοποιείται σε κάθε περίπτωση αναλόγως τα σήματα (ALUControl), από το οποίο σήμα στην συνέχεια τα bit 2 έως 7 ALUResult[7:2] θα περαστούν στην είσοδο της μνήμης δεδομένων (A) ως διεύθυνση 6 bit όπου υποδεικνύει εσωτερικά στην μνήμη δεδομένων την θέση που πρέπει να εγγραφεί το δεδομένο των 32 bit που προέρχεται από την θύρα ανάγνωσης RD2 του Register file μέσω της θύρας write data (WD) (32 bit) σύγχρονα με την έγκριση του σήματος εγγραφής MemWrite να παίρνει την τιμή MemWrite=1 (το σήμα έγκρισης εγγραφής στο Register file RegWrite έχει τιμή RegWrite =0). Το σήμα ελέγχου επιλογής εισόδου PCSrc λαμβάνει την τιμή PCSrc=0 δηλαδή η διεύθυνση της επόμενης εντολής θα είναι αυτή της αμέσως επόμενης εντολής, δηλαδή θα είναι ο μετρητής προγράμματος προσαυξημένος κατά τέσσερα δηλαδή ο PC+4. Ανεξάρτητα από την τιμή του πεδίο S της εντολής, δεν ενημερώνεται ο καταχωρητής κατάστασης που βρίσκεται στη μονάδα ελέγχου με τις νέες σημαίες συνθήκης N, Z, C, V (C = 0 και V = 0 στις λογικές πράξεις) που παράγονται από πράξεις στην ALU στην επόμενη ακμή του CLK.

5. Εντολή μνήμης B.

   Για την εντολή αυτή αφού ληφθεί η εντολή από την μνήμη εντολών, επειδή τα σήματα ελέγχου επιλογής εισόδου πολυπλέκτη RegScr πρέπει να λάβουν την τιμή RegScr[2:0]=001 διοχετεύεται στην είσοδο ανάγνωσης A1 του Register file η τιμή 15 έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης R15 που είναι ο καταχωρητής μετρητή προγράμματος από την θύρα ανάγνωσης RD1,η οποία θα εξάγει την διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της διεύθυνσης που λαμβάνουν.

Ο δεύτερος τελεστέος προέλευσης είναι άμεσος άρα το πεδίο src2 της εντολής περνά από τη μονάδα Extend όπου εκτελείται επέκταση μηδενός στα 32 bit του μη προσημασμένου άμεσου τελεστέου των 26 bit και το αποτέλεσμα εμφανίζεται στην έξοδο του Extend την ExtImm. Το σήμα ελέγχου είδους επέκτασης λαμβάνει την τιμή Immsrc=10. Έπειτα επειδή έχουμε άμεσο τελεστέο το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέο πρέπει να λάβει την τιμή ALUSrc= 1 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της θύρας εξόδου ExtImm του Extend. Στην συνέχεια στην ALU ανάλογα το σήμα ελέγχου (ALUControl) των 3 bit που προέρχεται από την μονάδα ελέγχου και το οποίο θα ελέγχει το είδος της πράξης (πρόσθεση) που θα πραγματοποιείται μεταξύ των λέξεων που θα βρεθούν στις εισόδους (α), (b) της ALU. Έτσι επειδή στην συγκεκριμένη εντολή ALUControl =’000’ πραγματοποιείται η πράξη της πρόσθεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b). Από την έξοδο της ALU λαμβάνεται ένα σήμα εξόδου (ALUResult) των 32 bit το οποίο περιλαμβάνει το αποτέλεσμα, από την πράξη της πρόσθεσης που πραγματοποιείται, το οποίο στην συνέχεια θα πρέπει να περάσει από ένα πολυπλέκτη του οποίου το σήμα ελέγχου επιλογής εισόδου MemtoReg πρέπει να λάβει την τιμή MemtoReg=0 και συνεχίζοντας πρέπει να δοθεί στο σήμα έγκρισης εγγραφής στο Register file RegWrite η τιμή RegWrite =0 για να μην γίνει εγγραφή στο αρχείο καταχωρητών και το δεδομένο να συνεχίσει την πορεία του ως τον πολυπλέκτη επιλογής διεύθυνσης επόμενης εντολής του οποίου το σήμα ελέγχου επιλογής εισόδου PCSrc λάβει την τιμή PCSrc=1 ώστε η διεύθυνση της επόμενης εντολής να είναι αυτή που έχει το δεδομένο. Ανεξάρτητα από την τιμή του πεδίο S της εντολής, δεν ενημερώνεται ο καταχωρητής κατάστασης που βρίσκεται στη μονάδα ελέγχου με τις νέες σημαίες συνθήκης N, Z, C, V (C = 0 και V = 0 στις λογικές πράξεις) που παράγονται από πράξεις στην ALU στην επόμενη ακμή του CLK.

6. Εντολή μνήμης B.

   Για την εντολή αυτή αφού ληφθεί η εντολή από την μνήμη εντολών, επειδή τα σήματα ελέγχου επιλογής εισόδου πολυπλέκτη RegScr πρέπει να λάβουν την τιμή RegScr[2:0]=101 διοχετεύεται στην είσοδο ανάγνωσης A1 του Register file η τιμή 15 έτσι ώστε να γίνει ανάγνωση του τελεστέου που είναι αποθηκευμένος στον καταχωρητή προέλευσης R15 που είναι ο καταχωρητής μετρητή προγράμματος από την θύρα ανάγνωσης RD1,η οποία θα εξάγει την διεύθυνση της τρέχουσας εντολής προσαυξημένη κατά 8 αφού έχει περάσει πριν καταλήξει στην είσοδο R15 από δυο αθροιστές που προσθέτουν την τιμή 4 ο καθένας στην τιμή της διεύθυνσης που λαμβάνουν. Ο δεύτερος τελεστέος προέλευσης είναι άμεσος άρα το πεδίο src2 της εντολής περνά από τη μονάδα Extend όπου εκτελείται επέκταση μηδενός στα 32 bit του μη προσημασμένου άμεσου τελεστέου των 26 bit και το αποτέλεσμα εμφανίζεται στην έξοδο του Extend την ExtImm. Το σήμα ελέγχου είδους επέκτασης λαμβάνει την τιμή Immsrc=10. Έπειτα επειδή έχουμε άμεσο τελεστέο το σήμα ελέγχου επιλογής εισόδου ALUSrc του πολυπλέκτη που επιλέγει τον άμεσο ή έμμεσο τελεστέο πρέπει να λάβει την τιμή ALUSrc= 1 έτσι ώστε στην είσοδο της ALU να φτάσουν τα περιεχόμενα από την θύρα ανάγνωσης του Register file RD1 και της θύρας εξόδου ExtImm του Extend. Στην συνέχεια στην ALU ανάλογα το σήμα ελέγχου (ALUControl) των 3 bit που προέρχεται από την μονάδα ελέγχου και το οποίο θα ελέγχει το είδος της πράξης (πρόσθεση) που θα πραγματοποιείται μεταξύ των λέξεων που θα βρεθούν στις εισόδους (α), (b) της ALU.

Έτσι επειδή στην συγκεκριμένη εντολή ALUControl =’000’ πραγματοποιείται η πράξη της πρόσθεσης των δύο προσημασμένων ακεραίων που αναπαριστούν οι λέξεις στις εισόδους (α), (b). Από την έξοδο της ALU λαμβάνεται ένα σήμα εξόδου (ALUResult) των 32 bit το οποίο περιλαμβάνει το αποτέλεσμα, από την πράξη της πρόσθεσης που πραγματοποιείται, το οποίο στην συνέχεια θα πρέπει να περάσει από ένα πολυπλέκτη του οποίου το σήμα ελέγχου επιλογής εισόδου MemtoReg πρέπει να λάβει την τιμή MemtoReg=0 ώστε το δεδομένο να διαβιβαστεί στον πολυπλέκτη επιλογής διεύθυνσης επόμενης εντολής του οποίου το σήμα ελέγχου επιλογής εισόδου PCSrc λάβει την τιμή PCSrc=1 ώστε η διεύθυνση της επόμενης εντολής να είναι αυτή που έχει το δεδομένο. Επίσης πρέπει να δοθεί στο σήμα έγκρισης εγγραφής στο Register file RegWrite η τιμή RegWrite =1 έτσι ώστε αφού η τιμή 14 διοχετεύεται στην είσοδο A3 του Register file να γίνει εγγραφή στον καταχωρητή προορισμού R14 του αρχείου καταχωρητών από την θύρα εγγραφής WD3 του μετρητή προγράμματος προσαυξημένου κατά 4 PC+4 αφού θα έχει περάσει πριν καταλήξει στην θύρα εγγραφής WD3 από ένα αθροιστή που προσθέτει την τιμή 4 στην τιμή της διεύθυνσης του μετρητή προγράμματος που λαμβάνει. Ανεξάρτητα από την τιμή του πεδίο S της εντολής, δεν ενημερώνεται ο καταχωρητής κατάστασης που βρίσκεται στη μονάδα ελέγχου με τις νέες σημαίες συνθήκης N, Z, C, V (C = 0 και V = 0 στις λογικές πράξεις) που παράγονται από πράξεις στην ALU στην επόμενη ακμή του CLK.

Δημιουργία προγράμματος σε συμβολική γλώσσα αρχιτεκτονικής ARM

Για τον έλεγχο της σωστής λειτουργίας του επεξεργαστής δημιουργήθηκε ένα πρόγραμμα σε συμβολική γλώσσα αρχιτεκτονικής ARM που περιλαμβάνει όλες τις υλοποιημένες εντολές και ενεργοποιεί όλες τις πιθανές ροές δεδομένων και λειτουργίες στη διαδρομή δεδομένων για όλες τις μορφές εντολών, δηλαδή για τις εντολές επεξεργασίας δεδομένων, τις εντολές μνήμης και τις εντολές διακλάδωσης που περιεγράφηκαν στην ακριβώς προηγούμενη παράγραφο. Οι πιθανές ροές δεδομένων στη διαδρομή δεδομένων έχουν σχέση με την μορφή της εντολής αλλά και την ιδιομορφία που έχουν αρκετές εντολές να εκτελούνται είτε με άμεσο είτε με έμμεσο δεύτερο τελεστέο προέλευσης, ενώ οι λειτουργίες έχουν σχέση καθαρά με το έργο που καλείται να παράξει η κάθε εντολή.

Με γνώμονα την δοκιμή όλων των εντολών και όσο το δυνατόν περισσότερων διαφορετικών ροών δεδομένων δημιουργήθηκε το εξής πρόγραμμα σε συμβολική γλώσσα αρχιτεκτονικής ARM:

```
START: MOV R0, #0;
MVN R1, #0;
STR R1, [R0,#15];
LDR R2, [R0,#15];
LSL R2, R2, #4;
ASR R2, R2, #4;
LSR R2, R2, #4;
ROR R2, R2, #4;
STR R1, [R0,#14];
LDR R3, [R0,#14];
ORR R2, R2, R3;
ADD R4, R0, #2;
SUB R5, R4, #-1;
CMP R1, R3;
SUBEQ R6, R5, R4;
ADDNE R6, R4, R5;
MOV R0, R6;
SUBS R7, R5, #5;
MOVMI R8, R1;
MOVPL R8, R7;
NOP;
MOV R10, R8;
ADDS R9, R8, R1;
BVS START;
EOR R13, R10, R4;
BL ROUTINE;
MOV R0, #1;
ROUTINE: MOV R13, R14;
B START;
```

Για να προχωρήσει η διαδικασία τεκμηρίωσης της ορθής σχεδίασης και λειτουργίας του επεξεργαστή, το πρόγραμμα σε συμβολική γλώσσα μεταφράστηκε μέσω ενός συμβολομεταφραστή σε γλώσσα μηχανής. Έτσι αφού πλέον το πρόγραμμα έχει την μορφή γλώσσας μηχανής διοχετεύτηκε στην Instruction memory ώστε να εκτελεστεί από τον επεξεργαστή μέσω μιας κατάλληλης διαδικασίας προσομοίωση, η οποία προσομοίωση θα διοχετεύσει στον επεξεργαστή τα κατάλληλα σήματα εισόδου όπως αυτά του clock (clk) ή του reset 1 bit που χρησιμοποιούνται για τον συγχρονισμό και την αρχικοποίηση αντίστοιχα των ψηφιακών υπομονάδων του.

Κώδικας σε γλώσσα VHDL που περιγράφει τη συμπεριφορά της μνήμης εντολών καθώς και κώδικας με το απαραίτητο πρόγραμμα δοκιμής στη μέγιστη συχνότητα λειτουργίας.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity imem is -- instruction memory
generic(
    N : positive := 6; --address length
    M : positive := 32); --data word length
port(a: in STD_LOGIC_VECTOR(N-1 downto 0);
rd: out STD_LOGIC_VECTOR(M-1 downto 0));
end imem;

architecture Behavioral of imem is
    type ROM_array is array (0 to (2**N)-1) of STD_LOGIC_VECTOR(M-1 downto 0);
    constant ROM : ROM_array :=(
    X"E3A00000",  -- START: MOV R0, #0
    X"E3E01000",  --        MVN R1, #0
    X"E580100F",  --        STR R1, [R0,#15]
    X"E590200F",  --        LDR R2, [R0,#15]
    X"E1A02202",  --        LSL R2, R2, #4
    X"E1A02242",  --        ASR R2, R2, #4
    X"E1A02222",  --        LSR R2, R2, #4
    X"E1A02262",  --        ROR R2, R2, #4
    X"E580100E",  --        STR R1, [R0,#14]
    X"E590300E",  --        LDR R3, [R0,#14]
    X"E1822003",  --        ORR R2, R2, R3
    X"E2804002",  --        ADD R4, R0, #2
    X"E2845001",  --        SUB R5, R4, #-1
    X"E1510003",  --        CMP R1, R3
    X"00456004",  --        SUBEQ R6, R5, R4
    X"10846005",  --        ADDNE R6, R4, R5
    X"E1A00006",  --        MOV R0, R6
    X"E2557005",  --        SUBS R7, R5, #5
    X"41A08001",  --        MOVMI R8, R1
    X"51A08007",  --        MOVPL R8, R7




  X"E1A00000",  --        NOP
  X"E1A0A008",  --        MOV R10, R8
  X"E0989001",  --        ADDS R9, R8, R1
  X"6AFFFFE7",  --        BVS START
  X"E02AD004",  --        EOR R13, R10, R4
  X"EB000000",  --        BL ROUTINE
  X"E3A00001",  --        MOV R0, #1
  X"E1A0D00E",  --        ROUTINE: MOV R13, R14
  X"EAFFFFE2",  --        B START

 X"00000000",
 X"00000000",
 X"00000000",

 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",

 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",
 X"00000000",


    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000");

begin
rd <= ROM(to_integer(unsigned(a)));
end Behavioral;
```

Εικόνα 40: Κώδικας Instruction memory όπου παρατηρείται πως έχει εισαχθεί το πρόγραμμα σε συμβολική γλώσσα μεταφρασμένο σε γλώσσα μηχανής στο δεκαεξαδικό σύστημα. (Για διευκόλυνση δίπλα από κάθε εντολή σε σχόλια βρίσκεται το ισοδύναμο σε συμβολική γλώσσα)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
use STD.ENV.ALL;
entity arm_tb is
--  Port ( );
end arm_tb;

architecture Behavioral of arm_tb is

component arm is -- single cycle processor
port(clk, reset: in STD_LOGIC;
PC: out STD_LOGIC_VECTOR(31 downto 0);
Instr: out STD_LOGIC_VECTOR(31 downto 0);
ALUResult:out STD_LOGIC_VECTOR(31 downto 0);
WriteData:out STD_LOGIC_VECTOR(31 downto 0);
Result:out STD_LOGIC_VECTOR(31 downto 0));
end component;

signal clk: STD_LOGIC :='0';
signal reset: STD_LOGIC:='1';
signal PC_tb: STD_LOGIC_VECTOR(31 downto 0);
signal Instr_tb: STD_LOGIC_VECTOR(31 downto 0);
signal ALUResult_tb: STD_LOGIC_VECTOR(31 downto 0);
signal WriteData_tb: STD_LOGIC_VECTOR(31 downto 0);
signal Result_tb: STD_LOGIC_VECTOR(31 downto 0);

constant clk_period : time := 10.343ns;

begin
uut: arm
port map(
clk=>clk,
reset=>reset,
PC=>PC_tb,


Instr=>Instr_tb,
ALUResult=>ALUResult_tb,
WriteData=>WriteData_tb,
Result=>Result_tb);

CLK_process: process is
begin
clk<='0';
wait for clk_period/2;
clk<='1';
wait for clk_period/2;
end process;

Test_tb: process is
begin
reset<='1';
wait for 100 ns;
wait until (clk ='0' and clk'event);
reset<='0';

wait for 29*clk_period;


stop(2);
end process Test_tb;
end Behavioral;
```

Εικόνα 41: Κώδικας προσομοίωσης για την εκτέλεση του προγράμματος ελέγχου σωστής λειτουργίας του επεξεργαστή, στον επεξεργαστή.

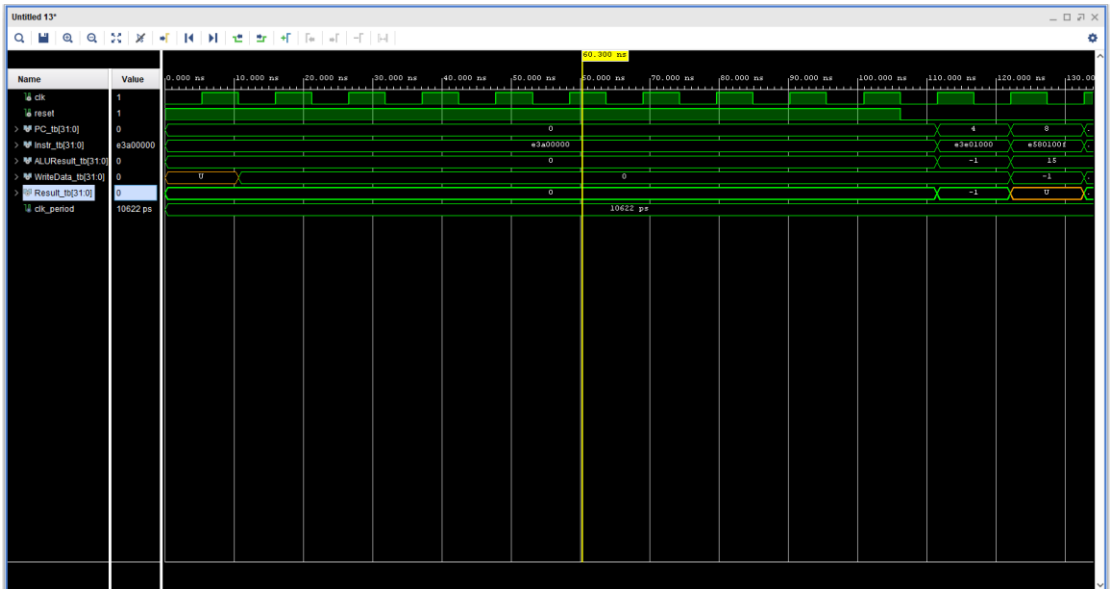
Για να έχει αξία η τεκμηρίωση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή στην συνέχεια δίνεται ένας πίνακας όπου υπολογίστηκε θεωρητικά με βάση την αρχιτεκτονική ARM τι σήματα θα έπρεπε να λαμβάνουμε στις εξόδους του επεξεργαστή, αν το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή εκτελούνταν θεωρητικά με βάση τους κανόνες της ARM και τις ροές των δεδομένων που ακολουθούνται ανάλογα την εντολή που εκτελείται όπως περιεγράφηκε σε προηγούμενη ενότητα. Τα σήματα εξόδου των 32 bit του επεξεργαστή έχουν μετατραπεί σε προσημασμένους ακεραίους τόσο στον πίνακα που ακολουθεί όσο και στα διαγράμματα χρονισμού πραγματικών προσομοιώσεων του επεξεργαστή που θα ακολουθήσουν έτσι ώστε να είναι πιο εύκολη η ανάλυση και η επεξεργασία τους. Εξαίρεση αποτελεί το σήμα εξόδου Instr των 32 bit το οποίο περιγράφει την τρέχουσα εντολή και έτσι αναπαρίσταται σε δεκαεξαδικό σύστημα καθώς πιο πάνω έχουμε κάνει την αντιστοίχιση των εντολών του προγράμματος ελέγχου του επεξεργαστή από συμβολική γλώσσα σε γλώσσα μηχανής με βάση το δεκαεξαδικό σύστημα.

PC	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56
Instr	E3A00000	E3E01000	E580100F	E590200F	E1A02202	E1A02242	E1A02222	E1A02262	E580100E	E590300E	E1822003	E2804002	E2845001	E1510003	00456004
ALUResult	0	-1	15	15	-16	-1	268435455	-251658241	14	14	-1	2	3	0	1
WriteData	X	X	-1	X	X	X	X	X	-1	X	-1	X	X	-1	2
Result	0	-1	X	-1	-16	-1	268435455	-251658241	X	-1	-1	2	3	X	1
PC	60	64	68	72	76	80	84	88	92	96	100	108	112	0	4
Instr	10846005	E1A00006	E2557005	41A08001	51A08007	E1A00000	E1A0A008	E0989001	6AFFFFE7	E02AD004	EB000000	E1A0D00E	EAFFFFE2	E3A00000	E3E01000
ALUResult	5	1	-2	-1	-2	1	-1	-2	0	-3	108	104	0	0	-1
WriteData	3	1	X	-1	-2	1	-1	-1	X	2	X	104	X	X	X
Result	5	1	-2	-1	-2	1	-1	-2	0	-3	108	104	0	0	-1

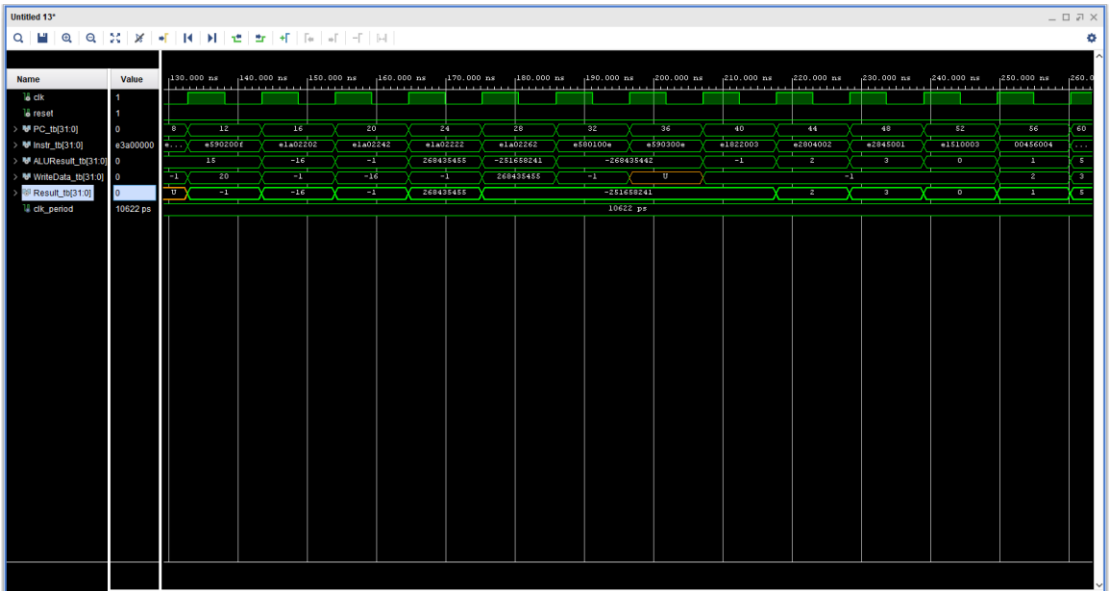
Εικόνα 42: Πίνακας με τις αναμενόμενες θεωρητικές εξόδους του επεξεργαστή σε περίπτωση εκτέλεσης του προγράμματος ελέγχου σωστής λειτουργίας του επεξεργαστή (όπου X δεν μας ενδιαφέρει η τιμή που λαμβάνεται)

Διαγράμματα χρονισμού των προσομοιώσεων του behavioral model, του post-synthesis model και του post-implementation model που προέκυψαν από την εκτέλεση πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή

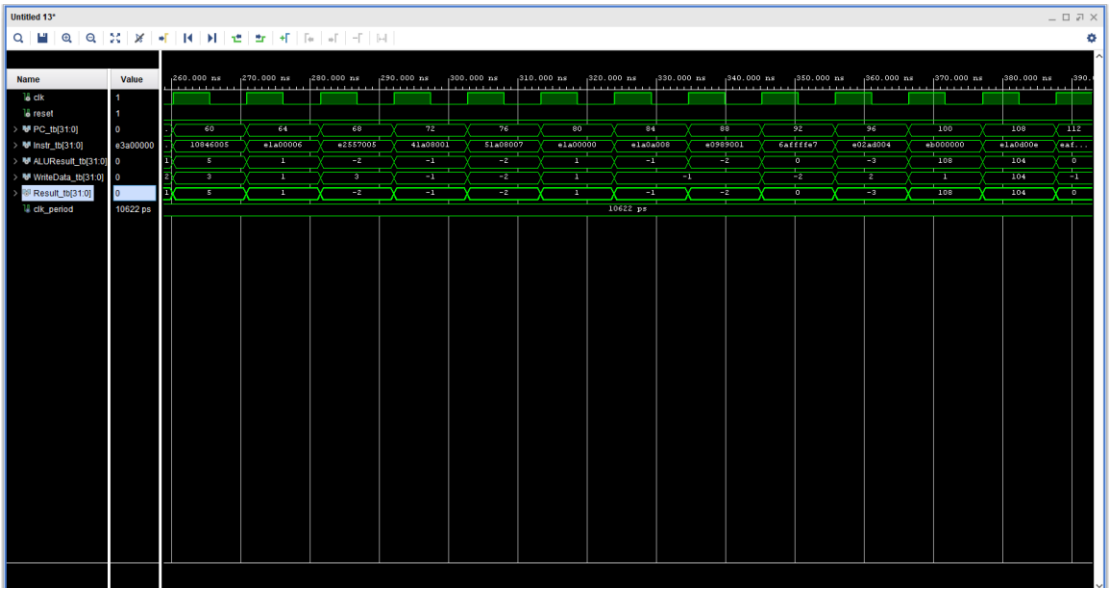
Έπειτα δίνονται τα διαγράμματα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με τα μοντέλα behavioral, του post-synthesis (μόνο λογική προσομοίωση που πρέπει να ταυτίζεται με εκείνη του behavioral) και του post-implementation (μόνο χρονική προσομοίωση).



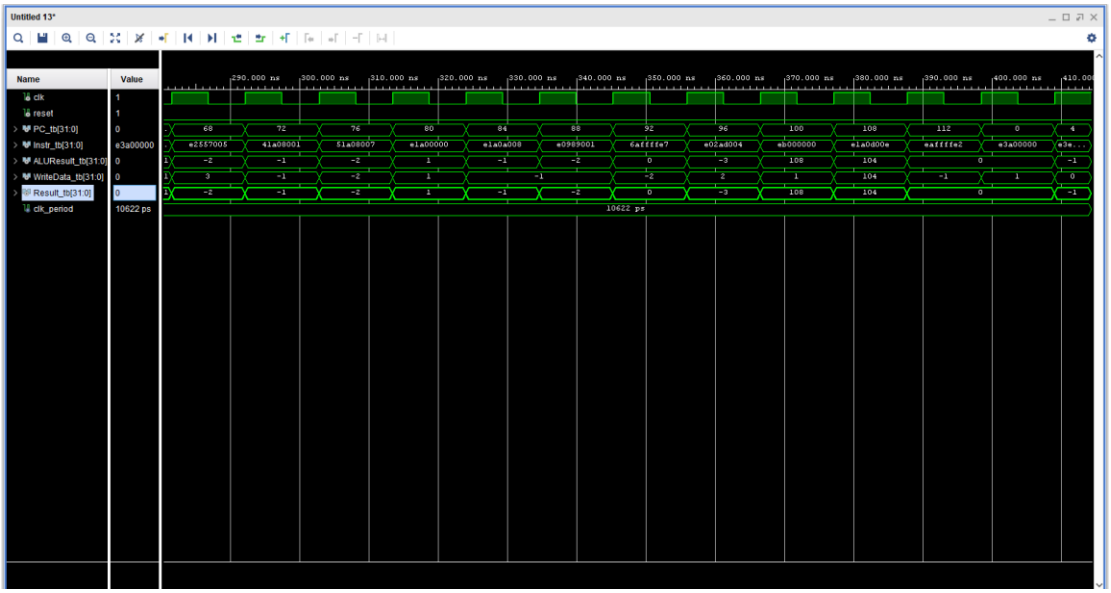
Εικόνα 43: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο behavioral (1)



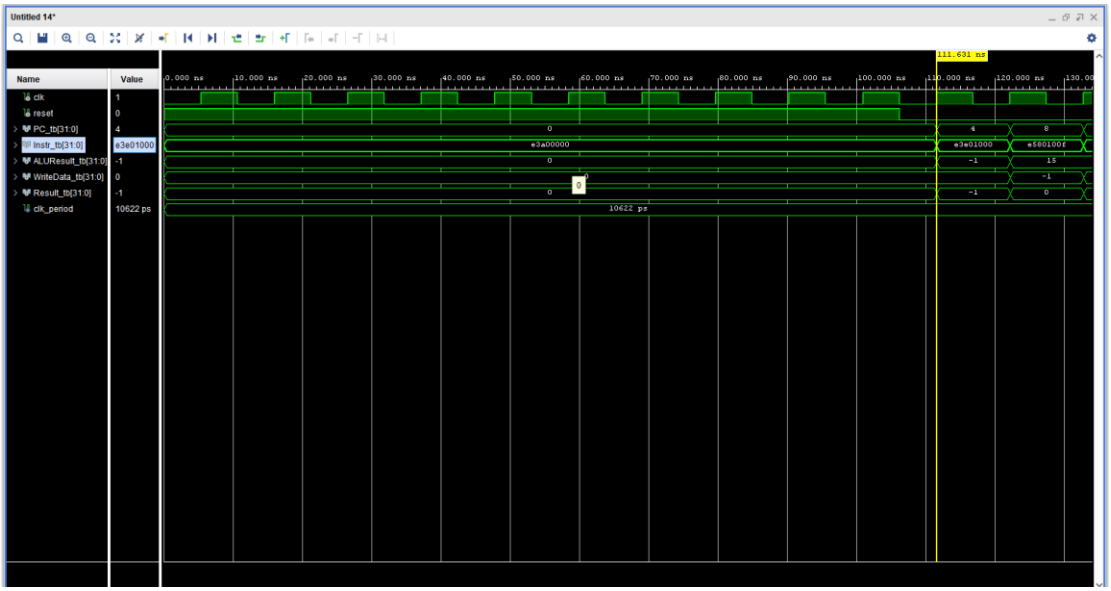
Εικόνα 44: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο behavioral (2)



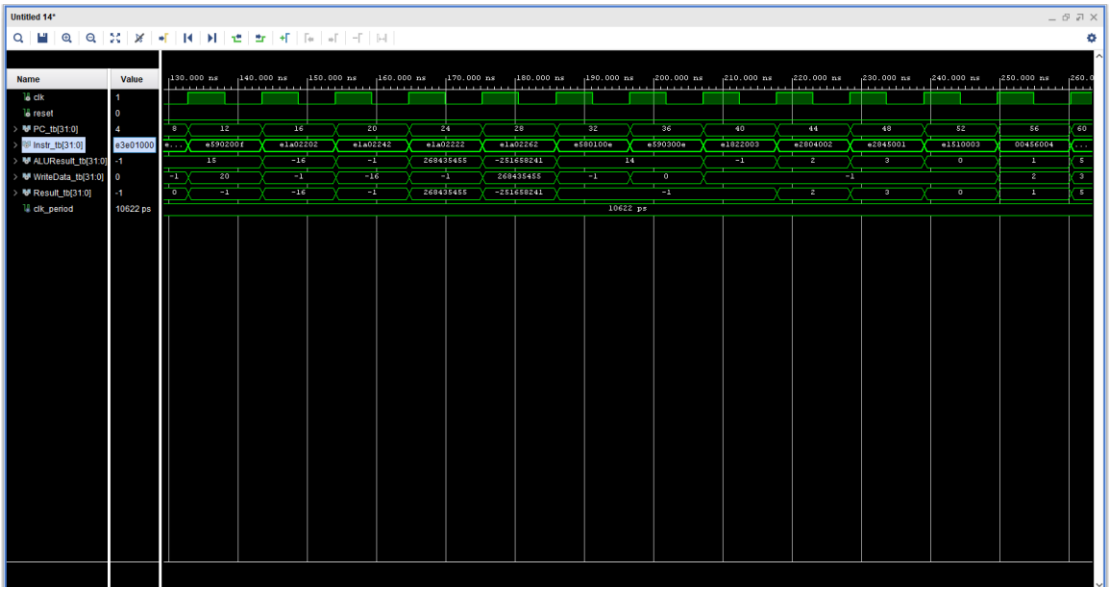
Εικόνα 45: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο behavioral (3)



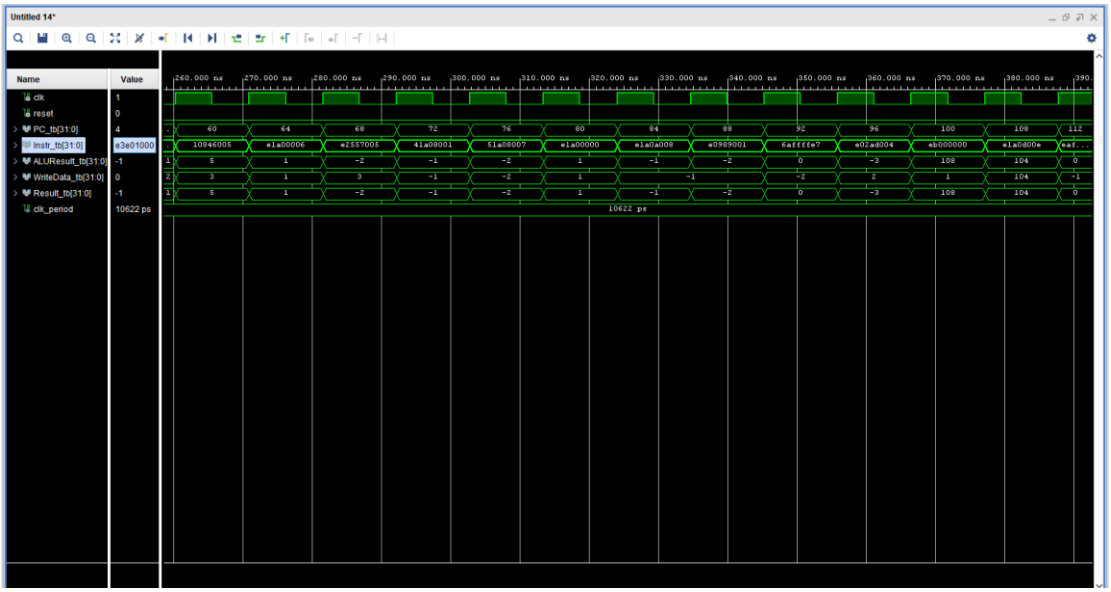
Εικόνα 46: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο behavioral (4)



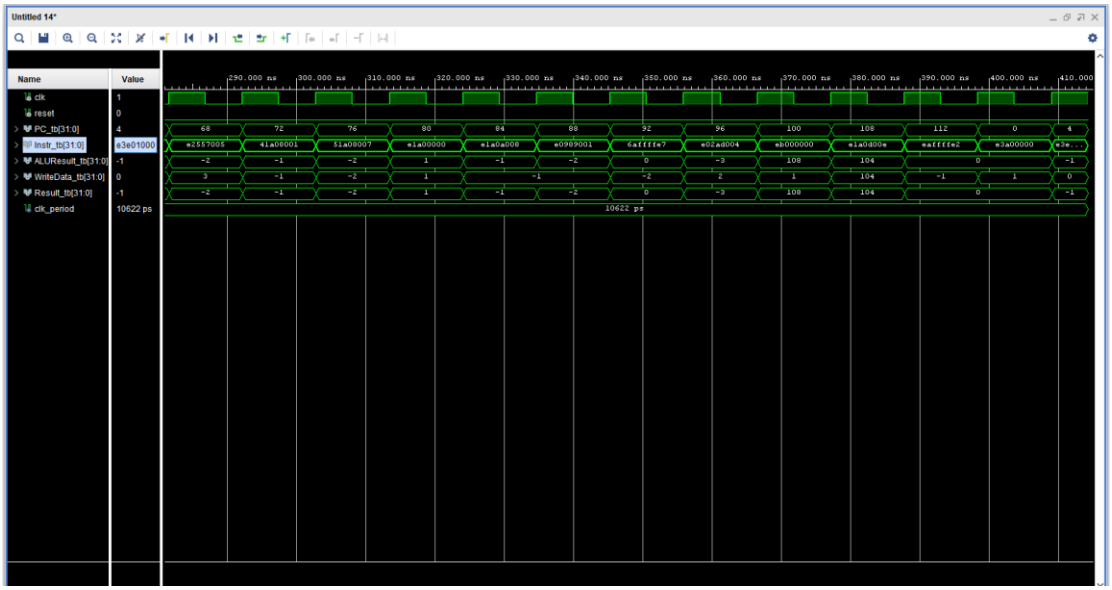
Εικόνα 47: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-synthesis (1)



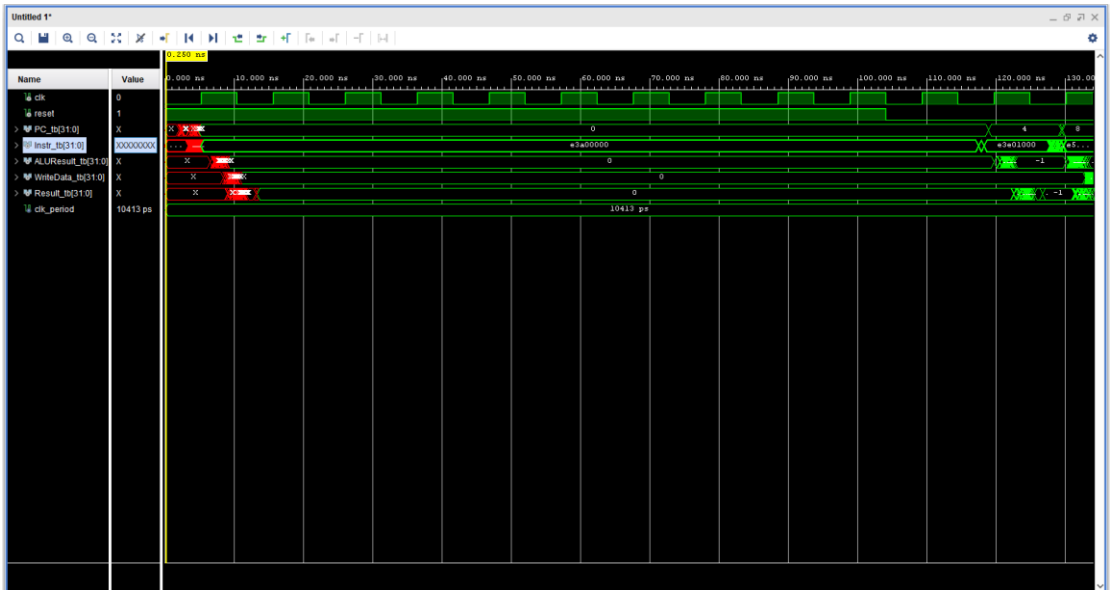
Εικόνα 48: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-synthesis (2)



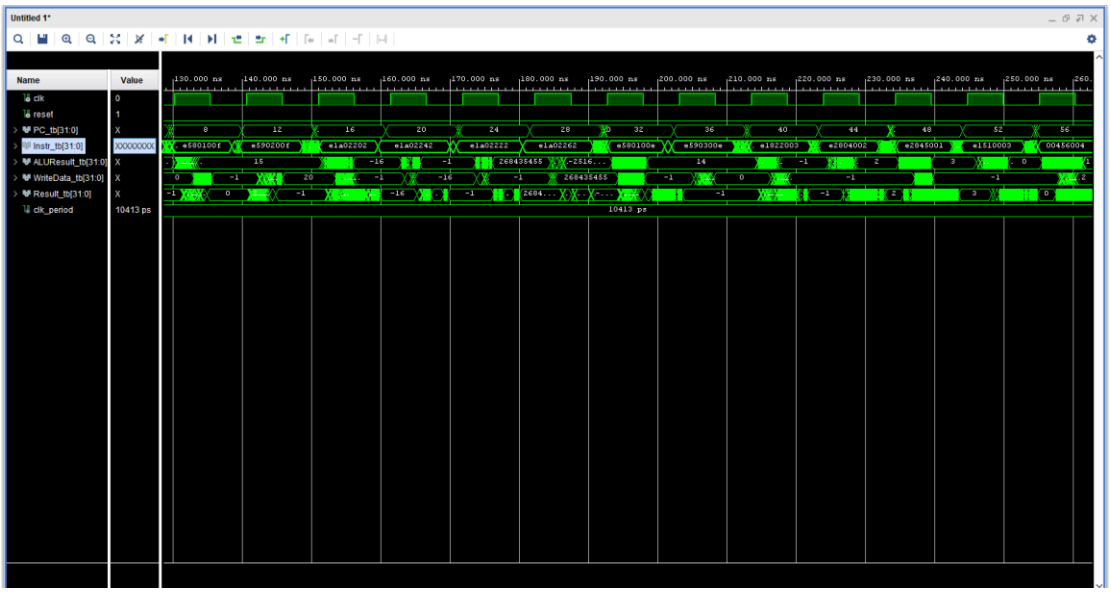
Εικόνα 49: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-synthesis (3)



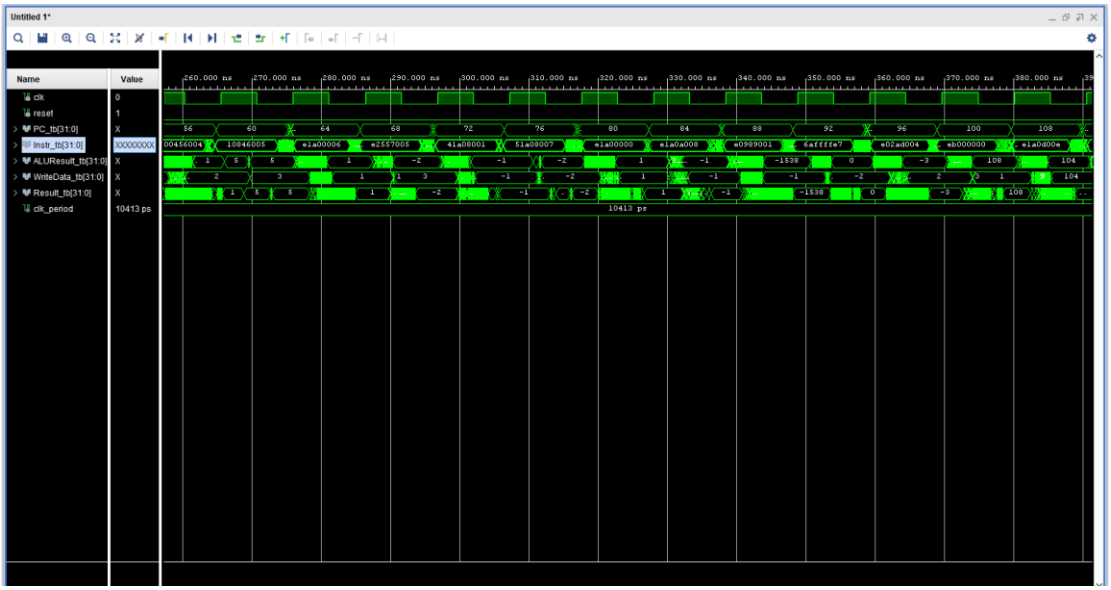
Εικόνα 50: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-synthesis (4)



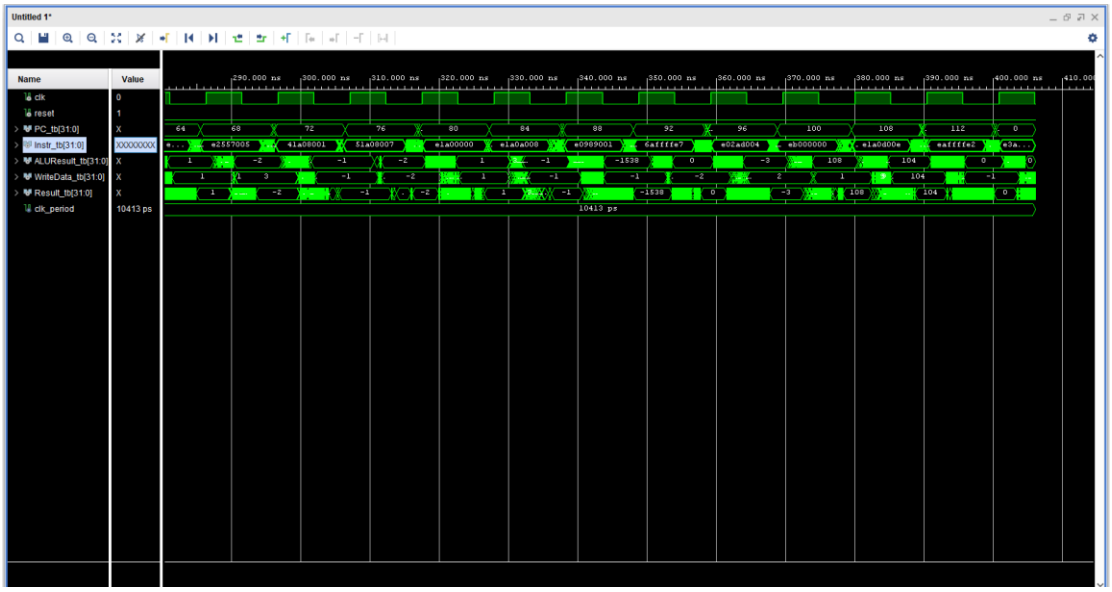
Εικόνα 51: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-implementation (1)



Εικόνα 52: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-implementation (2)



Εικόνα 53: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-implementation (3)



Εικόνα 54: Διάγραμμα χρονισμού των πραγματικών προσομοιώσεων του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του επεξεργαστή σύμφωνα με το μοντέλο post-implementation (4)

Τα αναμενόμενα θεωρητικά σήματα που θα έπρεπε να λαμβάνουμε στις εξόδους του επεξεργαστή με βάση τους κανόνες της αρχιτεκτονικής ARM, τα οποία παράχθηκαν στον πίνακα πιο πάνω, παρατηρούμε πως ταυτίζονται σχεδόν απόλυτα με τις τιμές των σημάτων εξόδου που λάβαμε από τα διαγράμματα χρονισμού κατά την πραγματική προσομοίωση του επεξεργαστή καθώς εκτελείται το πρόγραμμα ελέγχου σωστής λειτουργίας του. Ιδιαίτερα τα μοντέλα προσομοίωσης behavioral και post-synthesis δεν διαφέρουν καθόλου σε σχέση με τις θεωρητικές αναμενόμενες τιμές εξόδου. Από την άλλη το μοντέλο post-implementation όπως είναι ευδιάκριτο καθυστερεί ένα κύκλο με αποτέλεσμα να μην προλάβει να πραγματοποιηθεί η εκτέλεση της τελευταίας εντολής στον χρόνο προσομοίωσης που ορίσαμε, γεγονός που είναι λογικό καθώς το στάδιο του implementation είναι το πιο κοντινό σενάριο στην πραγματικότητα και τα πράγματα μπορεί να διαφοροποιηθούν ελαφρώς από τα προηγούμενα στάδια. Έτσι λοιπόν μπορούμε να εξάγουμε το συμπέρασμα πως ο επεξεργαστής σχεδιάστηκε ορθά και λειτουργεί σύμφωνα με τις προδιαγραφές της αρχιτεκτονικής ARM.

Ανάλυση των αποτελεσμάτων της σύνθεσης και της υλοποίησης του επεξεργαστή

Καταληκτικά μελετώντας το project summary και το report utilization της υλοποίησης του επεξεργαστή παρατηρούμε πως χρησιμοποιούνται αρκετά LUTs για την υλοποίηση της λογικής του επεξεργαστή και αρκετές LUTRAMS που χρησιμοποιούνται συνήθως για στοιχεία RAM/ROM όπως η Instruction memory και η Data memory. Επίσης είναι εμφανής και η χρήση μερικών Flip-Flops καθώς στην υλοποίηση υπάρχουν αρκετοί καταχωρητές, ενώ όπως είναι λογικό αν το σκεφτεί κανείς ο αριθμός 162 για τα I/O είναι όσος ο αριθμός των bits συνολικά των σημάτων εισόδου και εξόδου του επεξεργαστή. Τέλος όσο αφορά το ενεργειακό κομμάτι φαίνεται πως δεν καταναλώνει πολύ ισχύ ενώ η θερμοκρασία που προβλέπεται να φτάνει κατά την λειτουργία του, το chip θα απέχει αρκετά από την μέγιστη κρίσιμή θερμοκρασία που έχει την δυνατότητα να λειτουργήσει χωρίς να καταστραφεί.

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	623	53200	1.17
LUTRAM	76	17400	0.44
FF	36	106400	0.03
IO	162	200	81.00
BUFG	1	32	3.13

Power **Summary** | On-Chip

Total On-Chip Power:	**0.239 W**
Junction Temperature:	**27,8 °C**
Thermal Margin:	57,2 °C (4,8 W)
Effective ϑJA:	11,5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Implemented Power Report

Εικόνα 55:Το project summary και το report utilization της υλοποίησης του επεξεργαστή