

## Project 2: Μεθοδολογία Σχεδίασης Επεξεργαστή Πολλών Κύκλων

Άγγελος Ψημίτης – Χριστοδουλόπουλος

AM : 2019-513

Σχεδίαση Ψηφιακών Συστημάτων

Μάρτιος 2021

*ΔΙΠΜΣ Ηλεκτρονικός Αυτοματισμός*

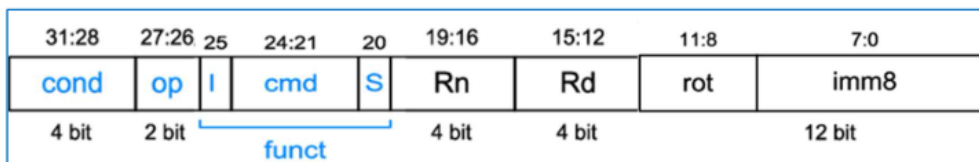
### 1. Περιγραφή των στοιχείων και της δομής του επεξεργαστή

#### 1.1 Εντολές της αρχιτεκτονικής ARM που έχουν υλοποιηθεί στην παρούσα εργασία

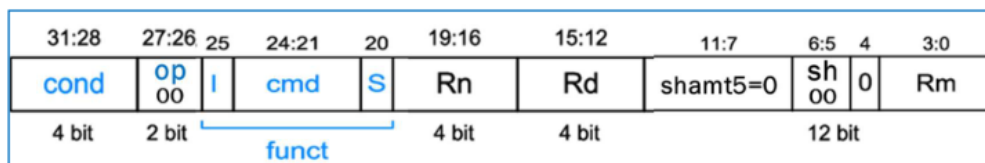
Εντολές επεξεργασίας δεδομένων (DP instructions) :

ADD(S), SUB(S), CMP, AND(S), XOR(S), MOV, MOVN -S =0,  
LSR, ASR -S =0.

Οι DP εντολές κωδικοποιούνται σε λέξεις των 32 bit σύμφωνα με τις επόμενες δύο εικόνες.



*Εντολές επεξεργασίας δεδομένων με άμεση διευθυνσιοδότηση ALU(S) -I*



*Εντολές επεξεργασίας δεδομένων με διευθυνσιοδότηση καταχωρητή ALU(S) -R*

Εντολές Μνήμης (Memory Instructions) :

LDR, STR

Οι εντολές μνήμης κωδικοποιούνται σε λέξεις των 32 bit όπως φαίνεται στην επόμενη εικόνα.



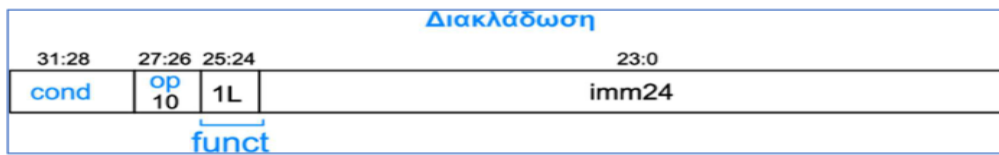
*Εντολές Μνήμης , ο μη προσημασμένος άμεσος τελεστής των 12 bit (imm12) χρησιμοποιείται ως offset με επέκταση μηδενός στα 32 bit.*

LDR Rd, [Rn, #imm12]; Rd = DM[Rn + #imm12] (U=1)  
LDR Rd, [Rn, #-imm12]; Rd = DM[Rn - #imm12] (U=0)

STR Rd, [Rn, #imm12]; DM[Rn + #imm12] = Rd (U=1)  
STR Rd, [Rn, #-imm12]; DM[Rn - #imm12] = Rd (U=0)

## Εντολές Διακλάδωσης (Branch Instructions):

B (Branch), BL (Branch Link)



Κωδικοποίηση εντολής διακλάδωσης στα 32 bit.

B label; PC = BTA = PC + 8 + imm24 x 4 (L = 0)

BL label; PC = BTA = PC + 8 + imm24 x 4; R14 = LR = PC+4 (L = 1)

## 1.2 Περιγραφή των κύριων ψηφιακών δομικών στοιχείων (components) της διαδρομής δεδομένων του επεξεργαστή.

### Αρχείο Καταχωρητών (Register File)

Το αρχείο καταχωρητών είναι μια διάταξη μνήμης RAM με διευθύνσεις των 4 bit για 16 καταχωρητές και δεδομένα (data) των 32 bit.

Διαθέτει σύγχρονη εγγραφή με έγκριση (όταν το σήμα WE = 1), ασύγχρονο διάβασμα, ενώ παρέχει τη δυνατότητα διαβάσματος δύο καταχωρητών ταυτόχρονα. Το RegFile module έχει σχεδιαστεί ως ακολουθιακό κύκλωμα το οποίο συγχρονίζεται με ένα εξωτερικό CLK σήμα καθώς κι ένα σύγχρονο RESET.

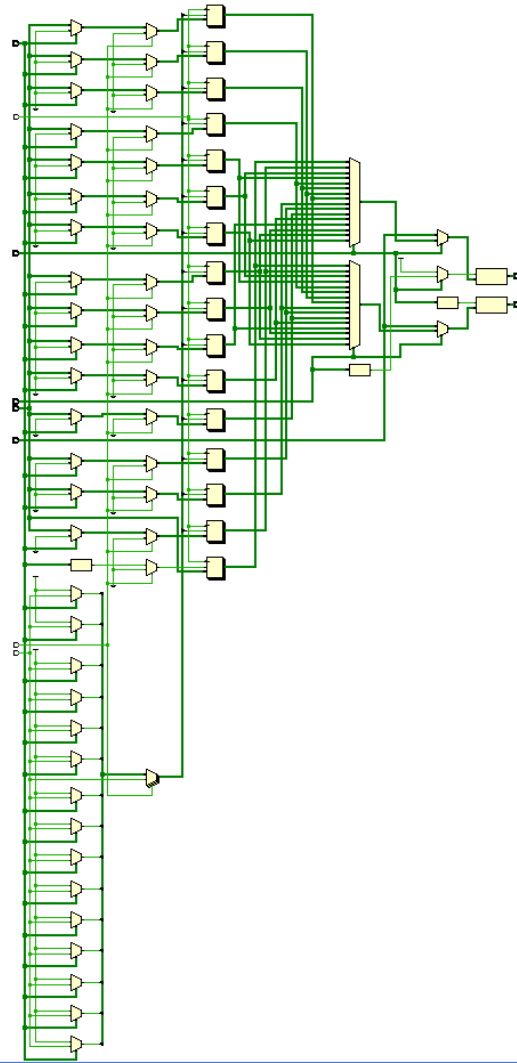
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity REG_FILE is
  Generic( N : positive := 4; -- address length
           M : positive := 32); -- data word length
  Port (
    CLK      : in std_logic;
    RESET    : in std_logic;
    WE3      : in std_logic;
    ADDR_A1  : in std_logic_vector(N-1 downto 0);
    ADDR_A2  : in std_logic_vector(N-1 downto 0);
    ADDR_A3  : in std_logic_vector(N-1 downto 0);
    WD3      : in std_logic_vector(M-1 downto 0);
    R15      : in std_logic_vector(M-1 downto 0);
    RD1      : out std_logic_vector(M-1 downto 0);
    RD2      : out std_logic_vector(M-1 downto 0)
  );
end REG_FILE;
```

Register File Entity

```
architecture Behavioral of REG_FILE is
  type RF_array is array (2**N-1 downto 0)
    of std_logic_vector(M-1 downto 0);
  signal RF : RF_array;
begin
  REG_FILE: process(clk)
  begin
    if(falling_edge(CLK))then
      if(RESET = '1')then
        for I in 0 to 2**N-2 loop
          RF(I) <= (others => '0');
        end loop;
      end if;
      if(WE3 = '1')then
        RF(to_integer(unsigned(ADDR_A3))) <= WD3;
      end if;
    end if;
  end process;
  OUTPUT_PROC: process(ADDR_A1, ADDR_A2, R15)
  begin
    if(ADDR_A1 = "1111")then
      RD1 <= R15;
    elsif(ADDR_A2 = "1111")then
      RD2 <= R15;
    else
      RD1 <= RF(to_integer(unsigned(ADDR_A1)));
      RD2 <= RF(to_integer(unsigned(ADDR_A2)));
    end if;
  end process;
end Behavioral;
```

Register File Architecture



Register File (RTL – Elaborated Design)

### Αριθμητική και Λογική Μονάδα (Arithmetic Logic Unit – ALU)

Πρόκειται για τη συνδυαστική λογική μονάδα που εκτελεί πράξεις σύμφωνα με την τιμή του σήματος ALUControl.

ALUControl	Operation	function	Unit
0000	ALUResult <= A +B	Add A and B	Arithmetic
0001	ALUResult <= A -B	Subtract A and B	Arithmetic
0010	ALUResult <= shiftleft(B)	Logical Shift Left B	Arithmetic
0011	ALUResult <= shiftright(B)	Logical Shift Right B	Arithmetic
0100	ALUResult <= shiftright(B)	Arithmetic Shift Right B	Arithmetic
0101	ALUResult <= ror(B)	Rotate B	Arithmetic
0110	ALUResult <= A	Transfer A	Arithmetic
0111	ALUResult <= B	Transfer B	Arithmetic
1000	ALUResult <= not(A)	Complement A	Logic
1001	ALUResult <= not(B)	Complement B	Logic
1010	ALUResult <= A and B	AND	Logic
1011	ALUResult <= A or B	OR	Logic
1100	ALUResult <= A nand B	NAND	Logic
1101	ALUResult <= A nor B	NOR	Logic
1110	ALUResult <= A xor B	XOR	Logic
1111	ALUResult <= not(A xor B)	XNOR	Logic

Πίνακας 1: Πίνακας αλήθειας για τη λειτουργία της ALU

Η υλοποίηση της ALU είναι structural και περιλαμβάνει 3 submodules:

Arithmetic Unit, Logic Unit κι έναν 2x1 Multiplexer για την επιλογή στην έξοδο της εξόδου της μιας ή της άλλης μονάδας ανάλογα με την τιμή του ALUControl(3) bit.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity ALU is
6  Generic ( Width : positive := 32);
7  Port (
8      CLK          : in  std_logic;
9      SrcA          : in  std_logic_vector(Width-1 downto 0);
10     SrcB          : in  std_logic_vector(Width-1 downto 0);
11     ALUControl    : in  std_logic_vector(3 downto 0);
12     ALUResult     : out std_logic_vector(Width-1 downto 0);
13     Flags         : out std_logic_vector(3 downto 0) -- N,Z,C,V
14 );
15 end ALU;
```

### ALU Entity

```
16
17 architecture Structural of ALU is
18
19     -- submodules declaration
20
21     component Logic_Unit
22     Generic (Width : positive := 32);
23     Port (
24         A : in  std_logic_vector(Width-1 downto 0);
25         B : in  std_logic_vector(Width-1 downto 0);
26         Sel : in std_logic_vector(2 downto 0);
27         S : out std_logic_vector(Width-1 downto 0)
28     );
29 end component;
30
31     component Arithmetic_Unit
32     Generic (Width : positive := 32);
33     Port (
34         A : in  std_logic_vector(Width-1 downto 0);
35         B : in  std_logic_vector(Width-1 downto 0);
36         Sel : in std_logic_vector(2 downto 0);
37         shamt : in std_logic_vector(4 downto 0);
38         S : out std_logic_vector(Width-1 downto 0);
39         Cout : out std_logic;
40         OV : out std_logic;
41         N : out std_logic;
42         Z : out std_logic
43     );
44 end component;
45
46     component MUX2x1
47     Generic (Width : positive := 32);
48     Port (
49         A0 : in  std_logic_vector(width-1 downto 0);
50         A1 : in  std_logic_vector(width-1 downto 0);
51         SEL : in std_logic;
52         Y : out std_logic_vector(width-1 downto 0)
53     );
54 end component;
55
56     signal logic_out_sig : std_logic_vector(Width-1 downto 0) := (others => '0');
57     signal arithmetic_out_sig : std_logic_vector(Width-1 downto 0) := (others => '0');
58     signal mux_out : std_logic_vector(Width-1 downto 0) := (others => '0');
59
60     begin
61
62         --logic unit instantiation
63         logic_inst: Logic_Unit
64         port map(
65             A => SrcA,
66             B => SrcB,
67             Sel => ALUControl(2 downto 0),
68             S => logic_out_sig
69         );
70
71         --arithmetic unit instantiation
72         arithmetic_inst: Arithmetic_Unit
73         port map(
74             A => SrcA,
75             B => SrcB,
76             Sel => ALUControl(2 downto 0),
77             shamt => SrcB(11 downto 7),
78             S => arithmetic_out_sig,
79             Cout => Flags(2),
80             OV => Flags(3),
81             N => Flags(0),
82             Z => Flags(1)
83         );
84
85         --multiplexer instantiation
86         mux_inst: MUX2x1
87         port map(
88             A0 => arithmetic_out_sig,
89             A1 => logic_out_sig,
90             SEL => ALUControl(3),
91             Y => mux_out
92         );
93
94         ALUResult <= mux_out;
95     end Structural;
```

### ALU Architecture

## Arithmetic Unit

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity Arithmetic_Unit is
6  Generic (Width : positive := 32);
7  Port (
8      A      : in  std_logic_vector(Width-1 downto 0);
9      B      : in  std_logic_vector(Width-1 downto 0);
10     Sel     : in  std_logic_vector(2 downto 0);
11     shamt   : in  std_logic_vector(4 downto 0);
12     S       : out std_logic_vector(Width-1 downto 0);
13     Cout    : out std_logic;
14     OV      : out std_logic;
15     N       : out std_logic;
16     Z       : out std_logic;
17 );
18 end Arithmetic_Unit;
```

```
20 architecture Behavioral of Arithmetic_Unit is
21 signal S_sig      : std_logic_vector(Width-1 downto 0) := (others => '0');
22 signal S_sig_temp : std_logic_vector(Width-1 downto 0) := (others => '0');
23 begin
24
25     Arithm_Proc: process(A,B,Sel)
26     variable A_s, B_s, S_s : signed (Width+1 downto 0);
27     variable shamt_n : natural range 0 to 31;
28     variable X_u : unsigned (Width-1 downto 0);
29     Variable X_s : signed (Width-1 downto 0);
30     begin
31         A_s := signed('0' & A(Width-1) & A);
32         B_s := signed('0' & B(Width-1) & B);
33         shamt_n := to_integer(unsigned(shamt));
34         X_u := unsigned(B);
35         X_s := signed(B);
36         case Sel is
37             when "000" =>
38                 S_s := A_s + B_s;
39                 S_sig <= std_logic_vector(S_s(Width-1 downto 0));
40                 Ov    <= S_s(Width) xor S_s(Width-1);
41                 Cout  <= S_s(Width+1);
42             when "001" =>
43                 S_s := A_s - B_s;
44                 S_sig <= std_logic_vector(S_s(Width-1 downto 0));
45                 Ov    <= S_s(Width) xor S_s(Width-1);
46                 Cout  <= S_s(Width+1);
47             when "010" => S_sig <= std_logic_vector(SHIFT_LEFT (X_u, shamt_n));
48             when "011" => S_sig <= std_logic_vector(SHIFT_RIGHT (X_u, shamt_n));
49             when "100" => S_sig <= std_logic_vector(SHIFT_RIGHT (X_s, shamt_n));
50             when "101" => S_sig <= std_logic_vector(ROTATE_RIGHT (X_s, shamt_n));
51             when "110" => S_sig <= A;
52             when "111" => S_sig <= B;
53             when others => S_sig <= (others => '0');
54         end case;
55
56         N <= S_sig(Width-1) or '0';
57         S_sig_temp <= S_sig nor x"00000000";
58         if(S_sig_temp = x"11111111") then
59             Z <= '1';
60         else
61             Z <= '0';
62         end if;
63     end process;
64
65     S <= S_sig;
66
67 end Behavioral;
```

### Arithmetic Unit entity and architecture

Για την υλοποίηση της πρόσθεσης/αφαίρεσης έχει χρησιμοποιηθεί προσημασμένος αθροιστής/αφαιρέτης (των 32 bit) με δημιουργία κρατουμένου (Cout) και σήματος υπερχειλίσσης (Ov). Όλα τα σήματα των σημαιών (N,Z,C,V) υπολογίζονται στην αριθμητική μονάδα προκειμένου να μπορεί η μονάδα ελέγχου να εκτελεί τις εντολές υπό συνθήκη.

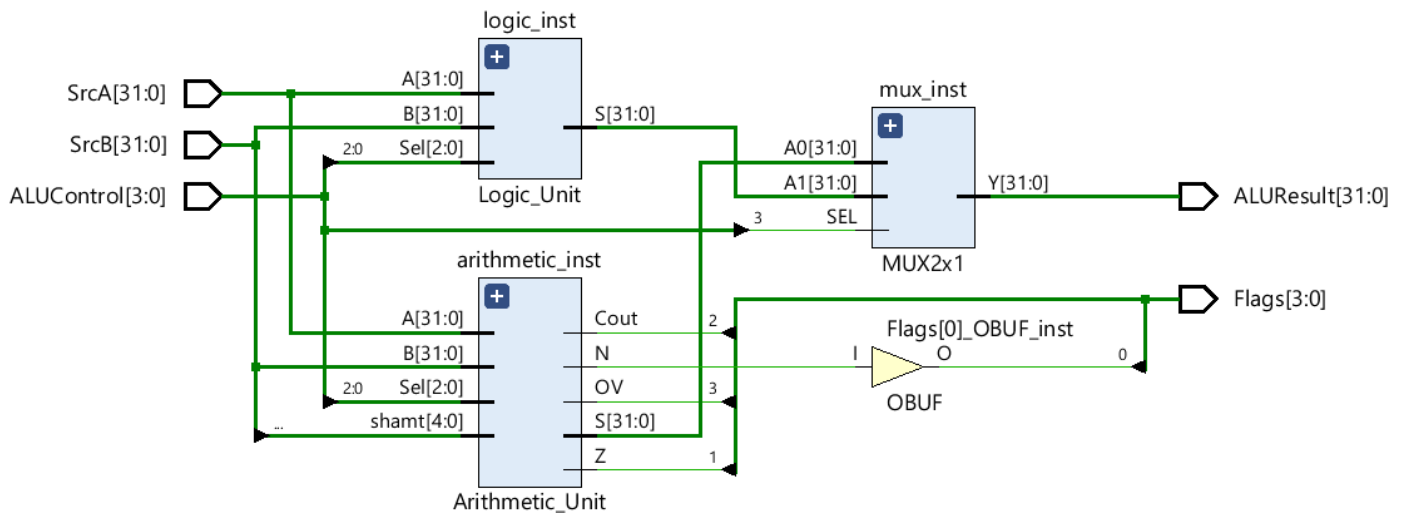
## Logic Unit

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Logic_Unit is
5  Generic (Width : positive := 32);
6  Port (
7      A    : in  std_logic_vector(Width-1 downto 0);
8      B    : in  std_logic_vector(Width-1 downto 0);
9      Sel   : in  std_logic_vector(2 downto 0);
10     S     : out std_logic_vector(Width-1 downto 0)
11 );
12 end Logic_Unit;
13
14 architecture Behavioral of Logic_Unit is
15     signal S_sig : std_logic_vector(Width-1 downto 0) := (others => '0');
16 begin
17     Logic_Unit : process(A,B,Sel)
18     begin
19         case Sel is
20             when "000" => S_sig <= not(A);
21             when "001" => S_sig <= not(B);
22             when "010" => S_sig <= A and B;
23             when "011" => S_sig <= A or B;
24             when "100" => S_sig <= A nand B;
25             when "101" => S_sig <= A nor B;
26             when "110" => S_sig <= A xor B;
27             when others => S_sig <= not(A xor B);
28         end case;
29     end process;
30     S <= S_sig;
31 end Behavioral;

```

*Logic Unit entity and architecture*



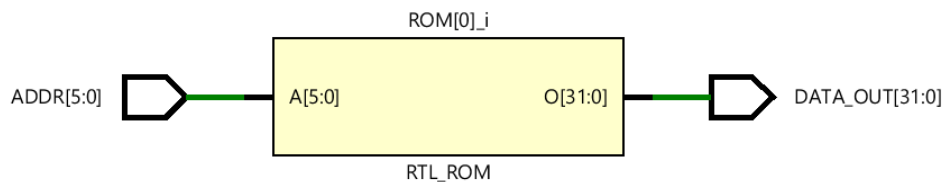
*ALU RTL Elaborated Design*

## Μνήμη Εντολών (Instruction Memory)

Η μνήμη εντολών είναι μια διάταξη μνήμης ROM με διευθύνσεις (address) των 6 bit και δεδομένα (data) των 32 bit, επομένως αποθηκεύει  $2^6$  λέξεις μεγέθους 32 bit. Σε αυτήν αποθηκεύονται τα instructions του προγράμματος που είναι να εκτελεστεί.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity ROM is
6  Generic( N : positive := 6; -- address length
7           M : positive := 32); -- data word length
8  Port (
9      ADDR      : in  std_logic_vector(N-1 downto 0);
10     DATA_OUT  : out std_logic_vector(M-1 downto 0));
11 end ROM;
12
13 architecture Behavioral of ROM is
14     type ROM_array is array (0 to 2**N-1)
15       of std_logic_vector(M-1 downto 0);
16     constant ROM : ROM_array := (
17         X"E3A00000", X"E3E01000", X"E0812000", X"E24230FF",
18         X"E1A00000", X"EAF9FFF9", X"00000000", X"00000000",
19         X"00000000", X"00000000", X"00000000", X"00000000",
20         X"00000000", X"00000000", X"00000000", X"00000000",
21         X"00000000", X"00000000", X"00000000", X"00000000",
22         X"00000000", X"00000000", X"00000000", X"00000000",
23         X"00000000", X"00000000", X"00000000", X"00000000",
24         X"00000000", X"00000000", X"00000000", X"00000000",
25         X"00000000", X"00000000", X"00000000", X"00000000",
26         X"00000000", X"00000000", X"00000000", X"00000000",
27         X"00000000", X"00000000", X"00000000", X"00000000",
28         X"00000000", X"00000000", X"00000000", X"00000000",
29         X"00000000", X"00000000", X"00000000", X"00000000",
30         X"00000000", X"00000000", X"00000000", X"00000000",
31         X"00000000", X"00000000", X"00000000", X"00000000",
32         X"00000000", X"00000000", X"00000000", X"00000000");
33 begin
34     DATA_OUT <= ROM(to_integer(unsigned(ADDR)));
35 end Behavioral;
```

*Instruction Memory (ROM) με αποθηκευμένες τιμές ως εντολές assembly του παραδείγματος που μας δόθηκε.*

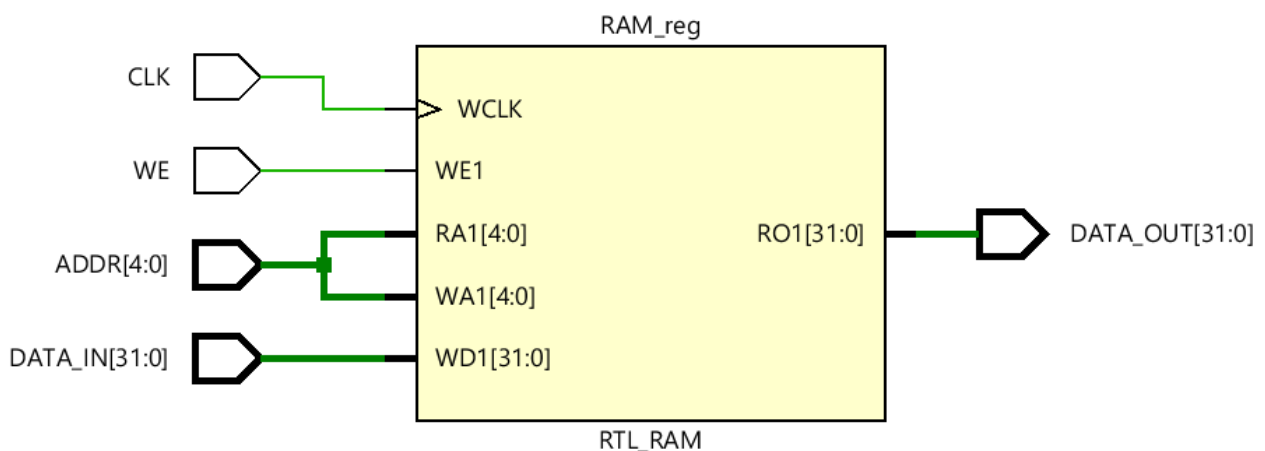


*Instruction Memory RTL*

## Μνήμη Δεδομένων (Data Memory)

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity Data_Memory is
6  Generic ( N : positive := 5; -- Address length
7           M : positive := 32); -- Data word length
8  Port (
9      CLK      : in std_logic;
10     WE       : in std_logic;
11     ADDR     : in std_logic_vector(N-1 downto 0);
12     DATA_IN  : in std_logic_vector(M-1 downto 0);
13     DATA_OUT : out std_logic_vector(M-1 downto 0)
14 );
15 end Data_Memory;
16
17 architecture Behavioral of Data_Memory is
18     type RAM_array is array (2**N-1 downto 0)
19     of std_logic_vector(M-1 downto 0);
20     signal RAM : RAM_array;
21 begin
22     Block_RAM: process (CLK)
23     begin
24         if (falling_edge(CLK)) then
25             if (WE = '1') then
26                 RAM(to_integer(unsigned(ADDR))) <= DATA_IN;
27             end if;
28         end if;
29     end process;
30     DATA_OUT <= RAM(to_integer(unsigned(ADDR)));
31 end Behavioral;
```

Η υλοποίηση έγινε με διάταξη μνήμης RAM χωρίς καταχωρητή εξόδου (Distributed RAM). Η διάταξη περιέχει σύγχρονη εγγραφή και ασύγχρονο διάβασμα.



Data Memory RTL

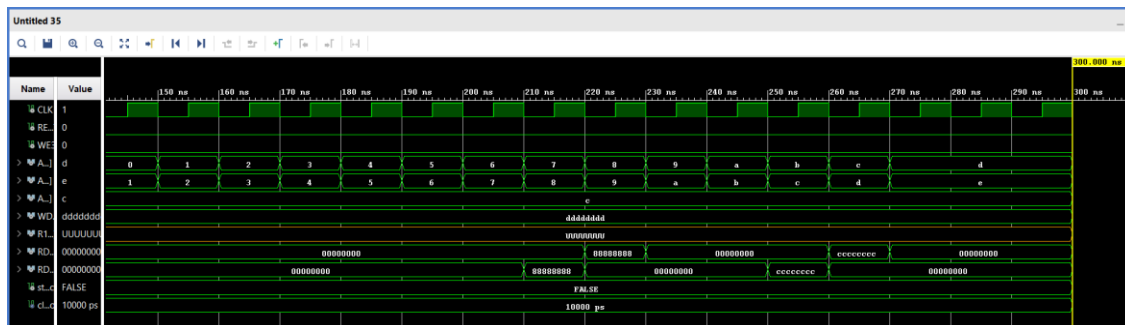


## 1.3 Προγράμματα Δοκιμής – Προσομοιώσεις

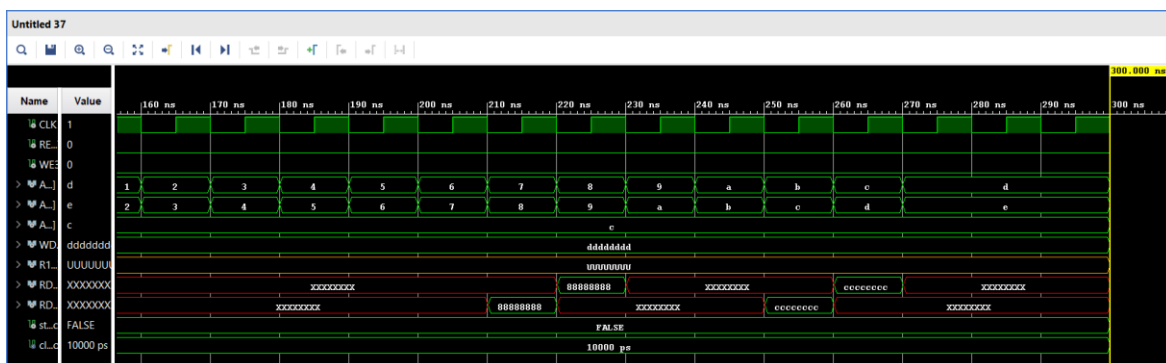
### Register File Testbench

```
57 stimulus: process
58 begin
59     RESET <= '1';
60     wait for 100 ns;
61     wait until (CLK = '0' and CLK'event);
62     RESET <= '0';
63     WE3 <= '1';
64     wait for 1*clock_period;
65     ADDR_A3 <= "1000"; -- write register 8
66     WD3 <= x"88888888";
67     wait for 1*clock_period;
68     ADDR_A3 <= "1100"; -- write register 12
69     WD3 <= x"CCCCCCCC";
70     wait for 1*clock_period;
71     WE3 <= '0';
72     ADDR_A3 <= "1100";
73     WD3 <= x"DDDDDDDD"; -- should not write register 12 because WE3 = 0
74     wait for 1*clock_period;
75 -- Test Reading of all registers except R15
76     for I in 0 to 13 loop
77         ADDR_A1 <= std_logic_vector(to_unsigned(I,4));
78         ADDR_A2 <= std_logic_vector(to_unsigned(I+1,4));
79         wait for 1*clock_period;
80     end loop;
81     wait for 2*clock_period;
82     -- Message and simulation end
83     report "TESTS COMPLETED";
84     stop(2);
85 end process;
```

Στο stimulus process του TB γράφουμε τον R8, στη συνέχεια τον R12, ενώ προσπαθούμε να ξαναγράψουμε τον R12 με το σήμα WE κλειστό. Τελικά διαβάζουμε τις τιμές όλων των registers εκτός του R15 στον οποίο δεν έχουμε δώσει τιμή.

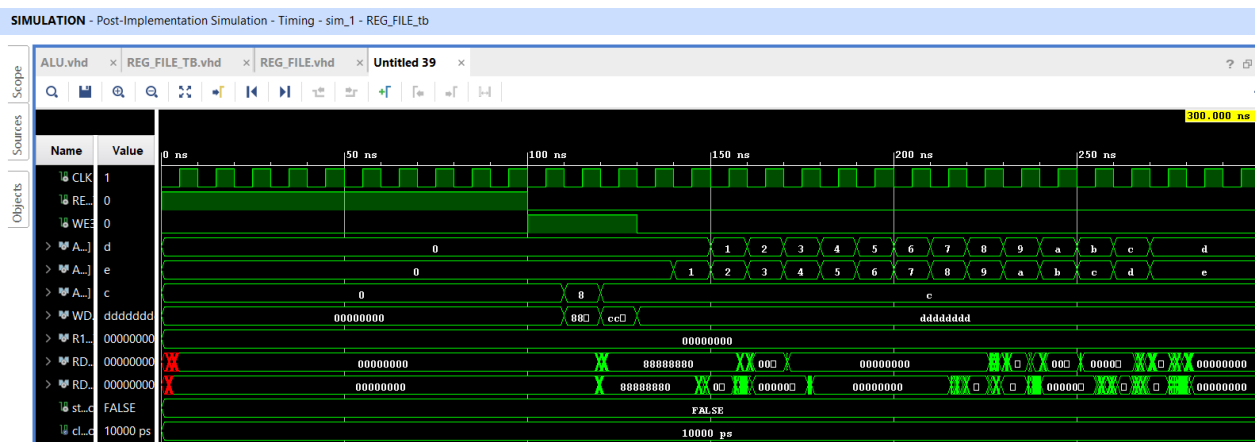


Behavioral Simulation του Register File.



Post Synthesis Simulation του Register File.

Παρατηρούμε ότι το Behavioral Simulation δίνει τα αποτελέσματα που περιμέναμε σύμφωνα με το testbench που δημιουργήσαμε. Όταν διαβάζουμε τους Registers, ο R8 έχει την τιμή x"88888888" ενώ ο R12 την τιμή x"CCCCCCCC", επιβεβαιώνοντας ότι οι καταχωρητές δεν γράφονται όταν το σήμα WE = '0'. Το post synthesis functional simulation ταυτίζεται με το behavioral model επιβεβαιώνοντας την σχεδίαση.



Post Implementation Timing Simulation for RegFile

## ALU Testbench

```

54 stimulus: process
55 begin
56     wait for 20 ns;
57     SrcA <= x"A0000004";
58     SrcB <= x"20000208";
59     wait for 1*CLK_period;
60     ALUControl <= "0000";
61     wait for 1*CLK_period;
62     ALUControl <= "0001";
63     wait for 1*CLK_period;
64     ALUControl <= "0010";
65     wait for 1*CLK_period;
66     ALUControl <= "0011";
67     wait for 1*CLK_period;
68     ALUControl <= "0100";
69     wait for 1*CLK_period;
70     ALUControl <= "0101";
71     wait for 1*CLK_period;
72     ALUControl <= "0110";
73     wait for 1*CLK_period;
74     ALUControl <= "0111";
75     wait for 1*CLK_period;
76     ALUControl <= "1000";
77     wait for 1*CLK_period;
78     ALUControl <= "1001";
79     wait for 1*CLK_period;
80     ALUControl <= "1010";
81     wait for 1*CLK_period;
82     ALUControl <= "1011";
83     wait for 1*CLK_period;
84     ALUControl <= "1100";
85     wait for 1*CLK_period;
86     ALUControl <= "1101";
87     wait for 1*CLK_period;
88     ALUControl <= "1110";
89     wait for 1*CLK_period;
90     ALUControl <= "1110";
91     wait for 1*CLK_period;
92     ALUControl <= "1111";
93     wait for 2*CLK_period;

```

Αριστερά φαίνεται το stimulus process που υλοποιήθηκε στο testbench της ALU. Εκτελούμε όλες τις αριθμητικές και λογικές πράξεις του πίνακα 1 μεταξύ των αριθμών x"A0000004" και x"20000208" δίνοντας τιμές στο σήμα ALUControl από το "0000" έως το "1111" αυξάνοντας κάθε φορά κατά 1. Όπως φαίνεται και από το behavioral simulation της ALU, οι τιμές στην έξοδο είναι οι αναμενόμενες ενώ οι σημαίες ανταποκρίνονται στην λογική με την οποία έχει σχεδιαστεί η μονάδα.

Συγκεκριμένα το σήμα Flags[3:0] κωδικοποιείται ως :

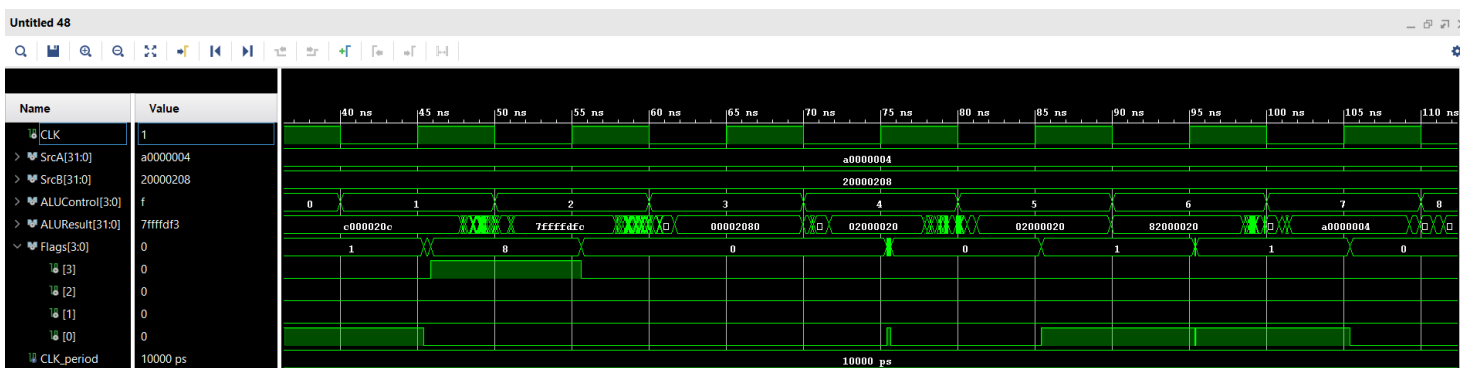
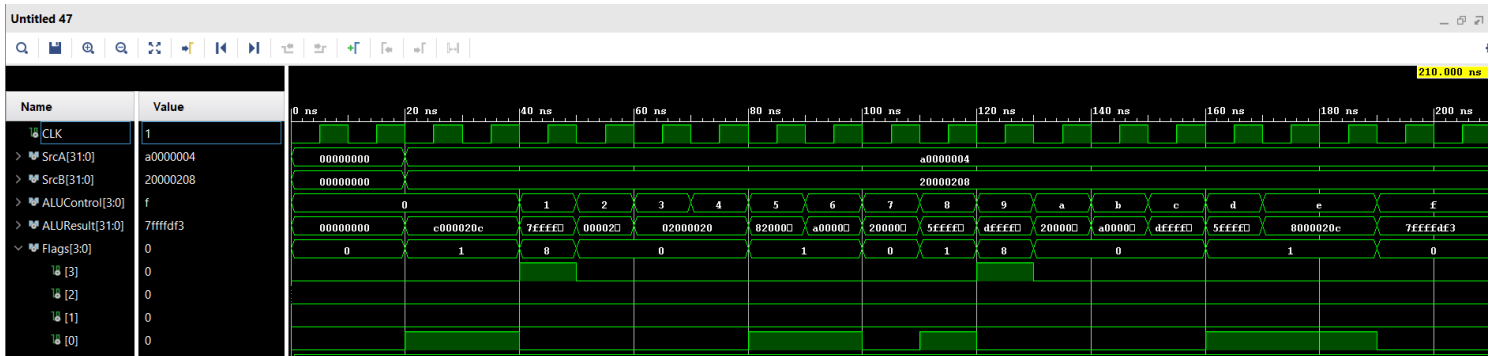
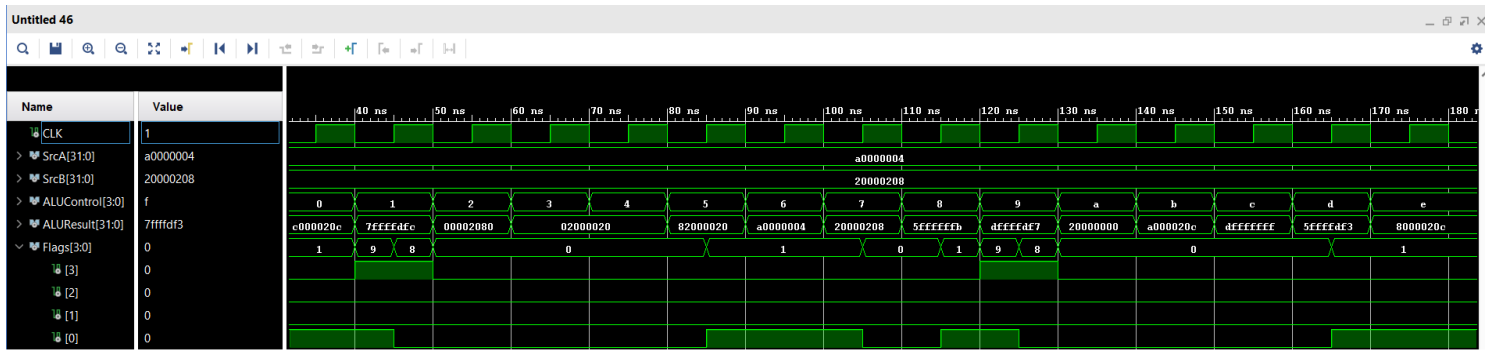
Flags(0) = N (Negative)

Flags(1) = Z (Zero)

Flags(2) = C (Cout)

Flags(3) = V (Overflow)

Παρατηρούμε για παράδειγμα ότι όταν προστίθενται οι 2 αριθμοί (ALUControl = x'0') το N παίρνει την τιμή 1 καθώς το αποτέλεσμα είναι αρνητικός αριθμός, αντίστοιχα όταν οι αριθμοί αφαιρούνται (ALUControl = x'1'), το V παίρνει την τιμή 1 ορθώς καθώς ο αριθμός που προκύπτει έχει μέτρο που ξεπερνάει την κωδικοποίηση των 32 bit. Όλα τα υπόλοιπα σήματα εξόδου φαίνεται να λειτουργούν σωστά.



#### 1.4 Περιγραφή της διαδρομής δεδομένων του επεξεργαστή (Data\_Path)

Στην ενότητα αυτή παρουσιάζεται η δομή του DataPath, μέσω της οποίας πραγματοποιούνται όλα τα βήματα εκτέλεσης της εντολής. Η δομή έχει υλοποιηθεί σε ανώτερο ιεραρχικό επίπεδο (structural). Το DataPath αποτελείται από τα εξής components :

- Instruction Memory (ROM)
- Register File (RAM)
- ALU
- Data Memory (RAM)
- Incrementer By 4
- Write Enable Register (με σύγχρονο RESET)
- 2x1 Multiplexer ( Sel[0:0] )
- 3x1 Multiplexer ( Sel[1:0] )
- Extend

Χρησιμοποιώντας τα παραπάνω εννέα components κάνουμε instantiate όλες τις απαραίτητες υπομονάδες για την ορθή λειτουργία της διαδρομής δεδομένων.

Συγκεκριμένα, ο WERegister χρησιμοποιείται για την υλοποίηση των αρχιτεκτονικών καταχωρητών (Program Counter, Status Register), καθώς και για την υλοποίηση των μη αρχιτεκτονικών καταχωρητών (Instruction Register, PC Register, A/B/I/S Registers, Memory Address Register, Write Data Memory Register και Read Data Memory Register) που τοποθετούνται ανάμεσα στα διακριτά βήματα εκτέλεσης της εντολής.

Ο 2x1 Multiplexer υλοποιεί έναν πολυπλέκτη για την ανάγνωση του αρχείου των καταχωρητών (A1/RD1, A2/RD2), υλοποιεί έναν πολυπλέκτη για την επιλογή σήματος Result μεταξύ των σημάτων του Read Data Memory Register και της ALU, υλοποιεί έναν πολυπλέκτη για την επιλογή του σήματος εγγραφής του αρχείου καταχωρητών (μεταξύ του PC+4 και του Result), υλοποιεί έναν πολυπλέκτη για την επιλογή του καταχωρητή στον οποίο θα γίνει η εγγραφή (μεταξύ του destination register Rd της εντολής - Instr[15:12] και του R14). Ο πολυπλέκτης αυτός είναι απαραίτητος προκειμένου να μπορεί να υλοποιηθεί η εντολή BL. Η οντότητα 2x1Mux υλοποιεί επίσης έναν πολυπλέκτη για την επιλογή του σήματος SrcB της δεύτερης εισόδου της ALU (Άμεση διευθυνσιοδότηση ή διευθυνσιοδότηση καταχωρητή) και η οντότητα 3x1Mux υλοποιεί τον πολυπλέκτη που επιλέγει την επόμενη εντολή που θα εκτελεστεί.

Η μονάδα INC4 (Incrementer by 4) χρησιμοποιείται για την αύξηση του PC σήματος (PC+4, διεύθυνση της επόμενης εντολής), καθώς και για την αύξηση του PC σήματος (PC+8) για την σωστή ανάγνωση του καταχωρητή R15.

Τέλος η μονάδα επέκτασης προσήμου/μηδενός (Extend) χρησιμοποιείται για την επέκταση του σήματος imm12 σε σήμα των 32 bit προκειμένου να χρησιμοποιηθεί απ' την ALU σε περίπτωση που η εντολή υποστηρίζει άμεση διευθυνσιοδότηση.

Ακολουθεί ο κώδικας VHDL της δομής του Data Path.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  --use IEEE.NUMERIC_STD.ALL;
4
5  entity Data_Path is
6  Port (
7      CLK      : in  std_logic;
8      RESET    : in  std_logic;
9      PCWrite   : in  std_logic;
10     IRWrite   : in  std_logic;
11     PCSrc     : in  std_logic_vector(1 downto 0);
12     RegSrc    : in  std_logic_vector(2 downto 0);
13     RegWrite   : in  std_logic;
14     MAWrite   : in  std_logic;
15     ImmSrc    : in  std_logic;
16     ALUSrc    : in  std_logic;
17     ALUControl : in  std_logic_vector(3 downto 0);
18     FlagsWrite : in  std_logic;
19     MemWrite   : in  std_logic;
20     MemtoReg   : in  std_logic;
21     Flags     : out std_logic_vector(3 downto 0);
22     PC        : out std_logic_vector(31 downto 0);
23     Instr     : out std_logic_vector(31 downto 0);
24     ALUResult  : out std_logic_vector(31 downto 0);
25     WriteData  : out std_logic_vector(31 downto 0);
26     Result    : out std_logic_vector(31 downto 0)
27 );
28 end Data_Path;
29
30 architecture Structural of Data_Path is
31
32     --Submodules Declaration
33     component ROM
34         Generic(N : positive := 6; -- address length
35                M : positive := 32); -- data word length
36     Port(
37         ADDR      : in  std_logic_vector(N-1 downto 0);
38         DATA_OUT  : out std_logic_vector(M-1 downto 0)
39     );
40 end component;
41
42     component REG_FILE
43         Generic(N : positive := 4; -- address length
44                M : positive := 32); -- data word length
45     Port(
46         CLK      : in  std_logic;
47         RESET    : in  std_logic;
48         WE3      : in  std_logic;
49         ADDR_A1  : in  std_logic_vector(N-1 downto 0);
50         ADDR_A2  : in  std_logic_vector(N-1 downto 0);
51         ADDR_A3  : in  std_logic_vector(N-1 downto 0);
52         WD3      : in  std_logic_vector(M-1 downto 0);
53         R15      : in  std_logic_vector(M-1 downto 0);
54         RD1      : out std_logic_vector(M-1 downto 0);
55         RD2      : out std_logic_vector(M-1 downto 0)
56     );
57 end component;
58
59     component ALU
60         Generic(Width : positive := 32);
61     Port(
62         CLK      : in  std_logic;
63         SrcA     : in  std_logic_vector(Width-1 downto 0);
64         SrcB     : in  std_logic_vector(Width-1 downto 0);
65         ALUControl : in  std_logic_vector(3 downto 0);
66         ALUResult : out std_logic_vector(Width-1 downto 0);
67         Flags    : out std_logic_vector(3 downto 0) -- N,Z,C,V
68     );
69 end component;
70
71     component Data_Memory
72         Generic(N : positive := 5; -- Address length
73                M : positive := 32); -- Data word length
74     Port(
75         CLK      : in  std_logic;
76         WE       : in  std_logic;
77         ADDR     : in  std_logic_vector(N-1 downto 0);
78         DATA_IN : in  std_logic_vector(M-1 downto 0);
79         DATA_OUT : out std_logic_vector(M-1 downto 0)
80     );
81 end component;
82
83     component Extend
84         Generic(Width_In : positive := 24;
85                Width_Out : positive := 32);
86     Port(
87         ImmSrc : in  std_logic;
88         S2_in  : in  std_logic_vector(Width_In-1 downto 0);
89         S2_out : out std_logic_vector(Width_Out-1 downto 0)
90     );
91 end component;
92
93     component MUX3x1
94         Generic(width : positive := 32);
95     Port (
96         A0      : in  std_logic_vector(width-1 downto 0);
97         A1      : in  std_logic_vector(width-1 downto 0);
98         A2      : in  std_logic_vector(width-1 downto 0);
99         SEL     : in  std_logic_vector(1 downto 0);
100        Y       : out std_logic_vector(width-1 downto 0)
101    );
102 end component;
103
104     component INC4
105         Generic(width : positive := 32);
106     Port (
107         INC_IN  : in  std_logic_vector(width-1 downto 0);
108         INC_OUT : out std_logic_vector(width-1 downto 0)
109     );
110 end component;
111
112     component WE_REGN
113         Generic (Width : positive := 32);
114     Port (
115         CLK      : in  std_logic;
116         Reset    : in  std_logic;
117         WE       : in  std_logic;
118         D        : in  std_logic_vector(Width-1 downto 0);
119         Q        : out std_logic_vector(Width-1 downto 0)
120     );
121 end component;
122
123     signal PC_IN_sig      : std_logic_vector(31 downto 0) := (others => '0');
124     signal PC_OUT_sig     : std_logic_vector(31 downto 0) := (others => '0');
125     signal IR_IN_sig      : std_logic_vector(31 downto 0) := (others => '0');
126     signal IR_OUT_sig     : std_logic_vector(31 downto 0) := (others => '0');
127     signal PCPlus4_sig    : std_logic_vector(31 downto 0) := (others => '0');
128     signal PCP4REG_sig    : std_logic_vector(31 downto 0) := (others => '0');
129     signal PCPlus8_sig    : std_logic_vector(31 downto 0) := (others => '0');
130     signal RA1_sig        : std_logic_vector(31 downto 0) := (others => '0');
131     signal RA2_sig        : std_logic_vector(31 downto 0) := (others => '0');
132     signal WA_sig         : std_logic_vector(31 downto 0) := (others => '0');
133     signal Result_sig     : std_logic_vector(31 downto 0) := (others => '0');
134     signal RD1_sig        : std_logic_vector(31 downto 0) := (others => '0');
135     signal RD2_sig        : std_logic_vector(31 downto 0) := (others => '0');
136     signal WriteData_sig  : std_logic_vector(31 downto 0) := (others => '0');
137     signal IREG_IN_sig    : std_logic_vector(31 downto 0) := (others => '0');
138     signal ExtImm_sig     : std_logic_vector(31 downto 0) := (others => '0');
139     signal SrcB_sig       : std_logic_vector(31 downto 0) := (others => '0');
140     signal SrcA_sig       : std_logic_vector(31 downto 0) := (others => '0');
141     signal Plags_sig      : std_logic_vector(31 downto 0) := (others => '0');
142     signal ALUResult_sig  : std_logic_vector(31 downto 0) := (others => '0');
143     signal MA_sig         : std_logic_vector(31 downto 0) := (others => '0');
144     signal WD_sig         : std_logic_vector(31 downto 0) := (others => '0');
145     signal Data_Memory_sig : std_logic_vector(31 downto 0) := (others => '0');
146     signal RD_sig         : std_logic_vector(31 downto 0) := (others => '0');
147     signal SREG_sig       : std_logic_vector(31 downto 0) := (others => '0');
148     signal WD3_sig        : std_logic_vector(31 downto 0) := (others => '0');
149
150     begin
151         --Program Counter instantiation
152         PC_inst: WE_REGN
153             Generic map(Width => 32)
154             Port map(
155                 CLK => CLK,
156                 Reset => RESET,
157                 WE => PCWrite,
158                 D => PC_IN_sig,
159                 Q => PC_OUT_sig
160             );
161
162         --Instruction Memory instantiation
163         INSTR_MEM_inst: ROM
164             Generic map(N => 6, M => 32)
165             Port map(
166                 ADDR => PC_OUT_sig(7 downto 2),
167                 DATA_OUT => IR_IN_sig
168             );
169
170         --Inorementer by 4 instantiation (pc+4)
171         Inc4_inst: INC4
172             Generic map(Width => 32)
173             Port map(
174                 INC_IN => PC_OUT_sig,
175                 INC_OUT => PCPlus4_sig
176             );
177
178         --Instruction Register instantiation
179         IR_REG_inst: WE_REGN
180             Generic map(Width => 32)
181             Port map(
182                 CLK => CLK,
183                 Reset => RESET,
184                 WE => IRWrite,
185                 D => IR_IN_sig,
186                 Q => IR_OUT_sig
187             );
188
189         --ALU instantiation
190         ALU_inst: ALU
191             Generic map(Width => 32)
192             Port map(
193                 CLK => CLK,
194                 SrcA => RA1_sig,
195                 SrcB => RA2_sig,
196                 ALUControl => ALUControl,
197                 ALUResult => Result_sig,
198                 Flags => Flags
199             );
200
201         --Data Memory instantiation
202         Data_Memory_inst: Data_Memory
203             Generic map(N => 5, M => 32)
204             Port map(
205                 CLK => CLK,
206                 WE => MemWrite,
207                 ADDR => ADDR_A1,
208                 DATA_IN => WriteData_sig,
209                 DATA_OUT => RD1_sig
210             );
211
212         --Register File instantiation
213         REG_inst: REG_FILE
214             Generic map(N => 4, M => 32)
215             Port map(
216                 CLK => CLK,
217                 RESET => RESET,
218                 WE3 => WE,
219                 ADDR_A1 => ADDR_A1,
220                 ADDR_A2 => ADDR_A2,
221                 ADDR_A3 => ADDR_A3,
222                 WD3 => WD3,
223                 R15 => R15,
224                 RD1 => RD1,
225                 RD2 => RD2
226             );
227
228         --Flags instantiation
229         Flags_inst: Extend
230             Generic map(Width_In => 24, Width_Out => 32)
231             Port map(
232                 ImmSrc => ImmSrc,
233                 S2_in => Result_sig(24 downto 1),
234                 S2_out => Flags
235             );
236
237         --PCPlus4 instantiation
238         PCPlus4_inst: MUX3x1
239             Generic map(width => 32)
240             Port map(
241                 A0 => PC_OUT_sig,
242                 A1 => PCPlus4_sig,
243                 A2 => PC_OUT_sig,
244                 SEL => PCSrc,
245                 Y => PC_IN_sig
246             );
247
248         --PCP4REG instantiation
249         PCP4REG_inst: MUX3x1
250             Generic map(width => 32)
251             Port map(
252                 A0 => PC_OUT_sig,
253                 A1 => PCPlus4_sig,
254                 A2 => PC_OUT_sig,
255                 SEL => PCSrc,
256                 Y => PCP4REG_sig
257             );
258
259         --PCPlus8 instantiation
260         PCPlus8_inst: MUX3x1
261             Generic map(width => 32)
262             Port map(
263                 A0 => PC_OUT_sig,
264                 A1 => PCPlus4_sig,
265                 A2 => PC_OUT_sig,
266                 SEL => PCSrc,
267                 Y => PCPlus8_sig
268             );
269
270         --MA instantiation
271         MA_inst: MUX3x1
272             Generic map(width => 32)
273             Port map(
274                 A0 => PC_OUT_sig,
275                 A1 => PCPlus4_sig,
276                 A2 => PC_OUT_sig,
277                 SEL => MAWrite,
278                 Y => MA_sig
279             );
280
281         --WD instantiation
282         WD_inst: MUX3x1
283             Generic map(width => 32)
284             Port map(
285                 A0 => PC_OUT_sig,
286                 A1 => PCPlus4_sig,
287                 A2 => PC_OUT_sig,
288                 SEL => MAWrite,
289                 Y => WD_sig
290             );
291
292         --Data_Memory instantiation
293         Data_Memory_inst: Data_Memory
294             Generic map(N => 5, M => 32)
295             Port map(
296                 CLK => CLK,
297                 WE => MemWrite,
298                 ADDR => ADDR_A1,
299                 DATA_IN => WriteData_sig,
300                 DATA_OUT => RD1_sig
301             );
302
303         --Register File instantiation
304         REG_inst: REG_FILE
305             Generic map(N => 4, M => 32)
306             Port map(
307                 CLK => CLK,
308                 RESET => RESET,
309                 WE3 => WE,
310                 ADDR_A1 => ADDR_A1,
311                 ADDR_A2 => ADDR_A2,
312                 ADDR_A3 => ADDR_A3,
313                 WD3 => WD3,
314                 R15 => R15,
315                 RD1 => RD1,
316                 RD2 => RD2
317             );
318
319         --Flags instantiation
320         Flags_inst: Extend
321             Generic map(Width_In => 24, Width_Out => 32)
322             Port map(
323                 ImmSrc => ImmSrc,
324                 S2_in => Result_sig(24 downto 1),
325                 S2_out => Flags
326             );
327
328         --PCPlus4 instantiation
329         PCPlus4_inst: MUX3x1
330             Generic map(width => 32)
331             Port map(
332                 A0 => PC_OUT_sig,
333                 A1 => PCPlus4_sig,
334                 A2 => PC_OUT_sig,
335                 SEL => PCSrc,
336                 Y => PC_IN_sig
337             );
338
339         --PCP4REG instantiation
340         PCP4REG_inst: MUX3x1
341             Generic map(width => 32)
342             Port map(
343                 A0 => PC_OUT_sig,
344                 A1 => PCPlus4_sig,
345                 A2 => PC_OUT_sig,
346                 SEL => PCSrc,
347                 Y => PCP4REG_sig
348             );
349
350         --PCPlus8 instantiation
351         PCPlus8_inst: MUX3x1
352             Generic map(width => 32)
353             Port map(
354                 A0 => PC_OUT_sig,
355                 A1 => PCPlus4_sig,
356                 A2 => PC_OUT_sig,
357                 SEL => PCSrc,
358                 Y => PCPlus8_sig
359             );
360
351         --MA instantiation
362         MA_inst: MUX3x1
363             Generic map(width => 32)
364             Port map(
365                 A0 => PC_OUT_sig,
366                 A1 => PCPlus4_sig,
367                 A2 => PC_OUT_sig,
368                 SEL => MAWrite,
369                 Y => MA_sig
370             );
371
372         --WD instantiation
373         WD_inst: MUX3x1
374             Generic map(width => 32)
375             Port map(
376                 A0 => PC_OUT_sig,
377                 A1 => PCPlus4_sig,
378                 A2 => PC_OUT_sig,
379                 SEL => MAWrite,
380                 Y => WD_sig
381             );
382
383         --Data_Memory instantiation
384         Data_Memory_inst: Data_Memory
385             Generic map(N => 5, M => 32)
386             Port map(
387                 CLK => CLK,
388                 WE => MemWrite,
389                 ADDR => ADDR_A1,
390                 DATA_IN => WriteData_sig,
391                 DATA_OUT => RD1_sig
392             );
393
394         --Register File instantiation
395         REG_inst: REG_FILE
396             Generic map(N => 4, M => 32)
397             Port map(
398                 CLK => CLK,
399                 RESET => RESET,
400                 WE3 => WE,
401                 ADDR_A1 => ADDR_A1,
402                 ADDR_A2 => ADDR_A2,
403                 ADDR_A3 => ADDR_A3,
404                 WD3 => WD3,
405                 R15 => R15,
406                 RD1 => RD1,
407                 RD2 => RD2
408             );
409
410         --Flags instantiation
411         Flags_inst: Extend
412             Generic map(Width_In => 24, Width_Out => 32)
413             Port map(
414                 ImmSrc => ImmSrc,
415                 S2_in => Result_sig(24 downto 1),
416                 S2_out => Flags
417             );
418
419         --PCPlus4 instantiation
420         PCPlus4_inst: MUX3x1
421             Generic map(width => 32)
422             Port map(
423                 A0 => PC_OUT_sig,
424                 A1 => PCPlus4_sig,
425                 A2 => PC_OUT_sig,
426                 SEL => PCSrc,
427                 Y => PC_IN_sig
428             );
429
430         --PCP4REG instantiation
431         PCP4REG_inst: MUX3x1
432             Generic map(width => 32)
433             Port map(
434                 A0 => PC_OUT_sig,
435                 A1 => PCPlus4_sig,
436                 A2 => PC_OUT_sig,
437                 SEL => PCSrc,
438                 Y => PCP4REG_sig
439             );
440
441         --PCPlus8 instantiation
442         PCPlus8_inst: MUX3x1
443             Generic map(width => 32)
444             Port map(
445                 A0 => PC_OUT_sig,
446                 A1 => PCPlus4_sig,
447                 A2 => PC_OUT_sig,
448                 SEL => PCSrc,
449                 Y => PCPlus8_sig
450             );
451
442         --MA instantiation
452         MA_inst: MUX3x1
453             Generic map(width => 32)
454             Port map(
455                 A0 => PC_OUT_sig,
456                 A1 => PCPlus4_sig,
457                 A2 => PC_OUT_sig,
458                 SEL => MAWrite,
459                 Y => MA_sig
460             );
461
453         --WD instantiation
462         WD_inst: MUX3x1
463             Generic map(width => 32)
464             Port map(
465                 A0 => PC_OUT_sig,
466                 A1 => PCPlus4_sig,
467                 A2 => PC_OUT_sig,
468                 SEL => MAWrite,
469                 Y => WD_sig
470             );
463         --Data_Memory instantiation
471         Data_Memory_inst: Data_Memory
472             Generic map(N => 5, M => 32)
473             Port map(
474                 CLK => CLK,
475                 WE => MemWrite,
476                 ADDR => ADDR_A1,
477                 DATA_IN => WriteData_sig,
478                 DATA_OUT => RD1_sig
479             );
474         --Register File instantiation
480         REG_inst: REG_FILE
481             Generic map(N => 4, M => 32)
482             Port map(
483                 CLK => CLK,
484                 RESET => RESET,
485                 WE3 => WE,
486                 ADDR_A1 => ADDR_A1,
487                 ADDR_A2 => ADDR_A2,
488                 ADDR_A3 => ADDR_A3,
489                 WD3 => WD3,
490                 R15 => R15,
491                 RD1 => RD1,
492                 RD2 => RD2
481             );
493         --Flags instantiation
494         Flags_inst: Extend
495             Generic map(Width_In => 24, Width_Out => 32)
496             Port map(
497                 ImmSrc => ImmSrc,
498                 S2_in => Result_sig(24 downto 1),
499                 S2_out => Flags
494             );
500         --PCPlus4 instantiation
501         PCPlus4_inst: MUX3x1
502             Generic map(width => 32)
503             Port map(
504                 A0 => PC_OUT_sig,
505                 A1 => PCPlus4_sig,
506                 A2 => PC_OUT_sig,
507                 SEL => PCSrc,
508                 Y => PC_IN_sig
501             );
509         --PCP4REG instantiation
510         PCP4REG_inst: MUX3x1
511             Generic map(width => 32)
512             Port map(
513                 A0 => PC_OUT_sig,
514                 A1 => PCPlus4_sig,
515                 A2 => PC_OUT_sig,
516                 SEL => PCSrc,
517                 Y => PCP4REG_sig
510             );
518         --PCPlus8 instantiation
519         PCPlus8_inst: MUX3x1
520             Generic map(width => 32)
521             Port map(
522                 A0 => PC_OUT_sig,
523                 A1 => PCPlus4_sig,
524                 A2 => PC_OUT_sig,
525                 SEL => PCSrc,
526                 Y => PCPlus8_sig
515             );
527         --MA instantiation
528         MA_inst: MUX3x1
529             Generic map(width => 32)
530             Port map(
531                 A0 => PC_OUT_sig,
532                 A1 => PCPlus4_sig,
533                 A2 => PC_OUT_sig,
534                 SEL => MAWrite,
535                 Y => MA_sig
522             );
536         --WD instantiation
537         WD_inst: MUX3x1
538             Generic map(width => 32)
539             Port map(
540                 A0 => PC_OUT_sig,
541                 A1 => PCPlus4_sig,
542                 A2 => PC_OUT_sig,
543                 SEL => MAWrite,
544                 Y => WD_sig
527             );
545         --Data_Memory instantiation
546         Data_Memory_inst: Data_Memory
547             Generic map(N => 5, M => 32)
548             Port map(
549                 CLK => CLK,
550                 WE => MemWrite,
551                 ADDR => ADDR_A1,
552                 DATA_IN => WriteData_sig,
553                 DATA_OUT => RD1_sig
542             );
554         --Register File instantiation
555         REG_inst: REG_FILE
556             Generic map(N => 4, M => 32)
557             Port map(
558                 CLK => CLK,
559                 RESET => RESET,
560                 WE3 => WE,
561                 ADDR_A1 => ADDR_A1,
562                 ADDR_A2 => ADDR_A2,
563                 ADDR_A3 => ADDR_A3,
564                 WD3 => WD3,
565                 R15 => R15,
566                 RD1 => RD1,
567                 RD2 => RD2
552             );
568         --Flags instantiation
569         Flags_inst: Extend
570             Generic map(Width_In => 24, Width_Out => 32)
571             Port map(
572                 ImmSrc => ImmSrc,
573                 S2_in => Result_sig(24 downto 1),
574                 S2_out => Flags
563             );
575         --PCPlus4 instantiation
576         PCPlus4_inst: MUX3x1
577             Generic map(width => 32)
578             Port map(
579                 A0 => PC_OUT_sig,
580                 A1 => PCPlus4_sig,
581                 A2 => PC_OUT_sig,
582                 SEL => PCSrc,
583                 Y => PC_IN_sig
572             );
584         --PCP4REG instantiation
585         PCP4REG_inst: MUX3x1
586             Generic map(width => 32)
587             Port map(
588                 A0 => PC_OUT_sig,
589                 A1 => PCPlus4_sig,
590                 A2 => PC_OUT_sig,
584             );
591         --PCPlus8 instantiation
592         PCPlus8_inst: MUX3x1
593             Generic map(width => 32)
594             Port map(
595                 A0 => PC_OUT_sig,
596                 A1 => PCPlus4_sig,
597                 A2 => PC_OUT_sig,
598                 SEL => PCSrc,
599                 Y => PCPlus8_sig
589             );
600         --MA instantiation
601         MA_inst: MUX3x1
602             Generic map(width => 32)
603             Port map(
604                 A0 => PC_OUT_sig,
605                 A1 => PCPlus4_sig,
606                 A2 => PC_OUT_sig,
607                 SEL => MAWrite,
608                 Y => MA_sig
602             );
609         --WD instantiation
610         WD_inst: MUX3x1
611             Generic map(width => 32)
612             Port map(
613                 A0 => PC_OUT_sig,
614                 A1 => PCPlus4_sig,
615                 A2 => PC_OUT_sig,
616                 SEL => MAWrite,
617                 Y => WD_sig
607             );
618         --Data_Memory instantiation
619         Data_Memory_inst: Data_Memory
620             Generic map(N => 5, M => 32)
621             Port map(
622                 CLK => CLK,
623                 WE => MemWrite,
624                 ADDR => ADDR_A1,
625                 DATA_IN => WriteData_sig,
626                 DATA_OUT => RD1_sig
612             );
627         --Register File instantiation
628         REG_inst: REG_FILE
629             Generic map(N => 4, M => 32)
630             Port map(
631                 CLK => CLK,
632                 RESET => RESET,
633                 WE3 => WE,
634                 ADDR_A1 => ADDR_A1,
635                 ADDR_A2 => ADDR_A2,
636                 ADDR_A3 => ADDR_A3,
637                 WD3 => WD3,
638                 R15 => R15,
639                 RD1 => RD1,
640                 RD2 => RD2
623             );
641         --Flags instantiation
642         Flags_inst: Extend
643             Generic map(Width_In => 24, Width_Out => 32)
644             Port map(
645                 ImmSrc => ImmSrc,
646                 S2_in => Result_sig(24 downto 1),
647                 S2_out => Flags
632             );
648         --PCPlus4 instantiation
649         PCPlus4_inst: MUX3x1
650             Generic map(width => 32)
651             Port map(
652                 A0 => PC_OUT_sig,
653                 A1 => PCPlus4_sig,
654                 A2 => PC_OUT_sig,
655                 SEL => PCSrc,
656                 Y => PC_IN_sig
642             );
657         --PCP4REG instantiation
658         PCP4REG_inst: MUX3x1
659             Generic map(width => 32)
660             Port map(
661                 A0 => PC_OUT_sig,
662                 A1 => PCPlus4_sig,
663                 A2 => PC_OUT_sig,
664                 SEL => PCSrc,
665                 Y => PCP4REG_sig
647             );
666         --PCPlus8 instantiation
667         PCPlus8_inst: MUX3x1
668             Generic map(width => 32)
669             Port map(
670                 A0 => PC_OUT_sig,
671                 A1 => PCPlus4_sig,
672                 A2 => PC_OUT_sig,
673                 SEL => PCSrc,
674                 Y => PCPlus8_sig
652             );
675         --MA instantiation
676         MA_inst: MUX3x1
677             Generic map(width => 32)
678             Port map(
679                 A0 => PC_OUT_sig,
680                 A1 => PCPlus4_sig,
681                 A2 => PC_OUT_sig,
682                 SEL => MAWrite,
683                 Y => MA_sig
667             );
684         --WD instantiation
685         WD_inst: MUX3x1
686             Generic map(width => 32)
687             Port map(
688                 A0 => PC_OUT_sig,
689                 A1 => PCPlus4_sig,
690                 A2 => PC_OUT_sig,
684             );
691         --Data_Memory instantiation
692         Data_Memory_inst: Data_Memory
693             Generic map(N => 5, M => 32)
694             Port map(
695                 CLK => CLK,
696                 WE => MemWrite,
697                 ADDR => ADDR_A1,
698                 DATA_IN => WriteData_sig,
699                 DATA_OUT => RD1_sig
687             );
700         --Register File instantiation
701         REG_inst: REG_FILE
702             Generic map(N => 4, M => 32)
703             Port map(
704                 CLK => CLK,
705                 RESET => RESET,
706                 WE3 => WE,
707                 ADDR_A1 => ADDR_A1,
708                 ADDR_A2 => ADDR_A2,
709                 ADDR_A3 => ADDR_A3,
710                 WD3 => WD3,
711                 R15 => R15,
712                 RD1 => RD1,
713                 RD2 => RD2
704             );
714         --Flags instantiation
715         Flags_inst: Extend
716             Generic map(Width_In => 24, Width_Out => 32)
717             Port map(
718                 ImmSrc => ImmSrc,
719                 S2_in => Result_sig(24 downto 1),
720                 S2_out => Flags
707             );
721         --PCPlus4 instantiation
722         PCPlus4_inst: MUX3x1
723             Generic map(width => 32)
724             Port map(
725                 A0 => PC_OUT_sig,
726                 A1 => PCPlus4_sig,
727                 A2 => PC_OUT_sig,
728                 SEL => PCSrc,
729                 Y => PC_IN_sig
714             );
730         --PCP4REG instantiation
731         PCP4REG_inst: MUX3x1
732             Generic map(width => 32)
733             Port map(
734                 A0 => PC_OUT_sig,
735                 A1 => PCPlus4_sig,
736                 A2 => PC_OUT_sig,
737                 SEL => PCSrc,
738                 Y => PCP4REG_sig
719             );
739         --PCPlus8 instantiation
740         PCPlus8_inst: MUX3x1
741             Generic map(width => 32)
742             Port map(
743                 A0 => PC_OUT_sig,
744                 A1 => PCPlus4_sig,
745                 A2 => PC_OUT_sig,
746                 SEL => PCSrc,
747                 Y => PCPlus8_sig
724             );
748         --MA instantiation
749         MA_inst: MUX3x1
750             Generic map(width => 32)
751             Port map(
752                 A0 => PC_OUT_sig,
753                 A1 => PCPlus4_sig,
754                 A2 => PC_OUT_sig,
755                 SEL => MAWrite,
756                 Y => MA_sig
739             );
757         --WD instantiation
758         WD_inst: MUX3x1
759             Generic map(width => 32)
760             Port map(
761                 A0 => PC_OUT_sig,
762                 A1 => PCPlus4_sig,
763                 A2 => PC_OUT_sig,
764                 SEL => MAWrite,
765                 Y => WD_sig
744             );
766         --Data_Memory instantiation
767         Data_Memory_inst: Data_Memory
768             Generic map(N => 5, M => 32)
769             Port map(
770                 CLK => CLK,
771                 WE => MemWrite,
772                 ADDR => ADDR_A1,
773                 DATA_IN => WriteData_sig,
774                 DATA_OUT => RD1_sig
751             );
775         --Register File instantiation
776         REG_inst: REG_FILE
777             Generic map(N => 4, M => 32)
778             Port map(
779                 CLK => CLK,
780                 RESET => RESET,
781                 WE3 => WE,
782                 ADDR_A1 => ADDR_A1,
783                 ADDR_A2 => ADDR_A2,
784                 ADDR_A3 => ADDR_A3,
785                 WD3 => WD3,
786                 R15 => R15,
787                 RD1 => RD1,
788                 RD2 => RD2
754             );
789         --Flags instantiation
790         Flags_inst: Extend
791             Generic map(Width_In => 24, Width_Out => 32)
792             Port map(
793                 ImmSrc => ImmSrc,
794                 S2_in => Result_sig(24 downto 1),
795                 S2_out => Flags
767             );
796         --PCPlus4 instantiation
797         PCPlus4_inst: MUX3x1
798             Generic map(width => 32)
799             Port map(
800                 A0 => PC_OUT_sig,
801                 A1 => PCPlus4_sig,
802                 A2 => PC_OUT_sig,
803                 SEL => PCSrc,
804                 Y => PC_IN_sig
790             );
805         --PCP4REG instantiation
806         PCP4REG_inst: MUX3x1
807             Generic map(width => 32)
808             Port map(
809                 A0 => PC_OUT_sig,
810                 A1 => PCPlus4_sig,
811                 A2 => PC_OUT_sig,
812                 SEL => PCSrc,
813                 Y => PCP4REG_sig
795             );
814         --PCPlus8 instantiation
815         PCPlus8_inst: MUX3x1
816             Generic map(width => 32)
817             Port map(
818                 A0 => PC_OUT_sig,
819                 A1 => PCPlus4_sig,
820                 A2 => PC_OUT_sig,
821                 SEL => PCSrc,
822                 Y => PCPlus8_sig
800             );
823         --MA instantiation
824         MA_inst: MUX3x1
825             Generic map(width => 32)
826             Port map(
827                 A0 => PC_OUT_sig,
828                 A1 => PCPlus4_sig,
829                 A2 => PC_OUT_sig,
830                 SEL => MAWrite,
831                 Y => MA_sig
814             );
832         --WD instantiation
833         WD_inst: MUX3x1
834             Generic map(width => 32)
835             Port map(
836                 A0 => PC_OUT_sig,
837                 A1 => PCPlus4_sig,
838                 A2 => PC_OUT_sig,
839                 SEL => MAWrite,
840                 Y => WD_sig
819             );
841         --Data_Memory instantiation
842         Data_Memory_inst: Data_Memory
843             Generic map(N => 5, M => 32)
844             Port map(
845                 CLK => CLK,
846                 WE => MemWrite,
847                 ADDR => ADDR_A1,
848                 DATA_IN => WriteData_sig,
849                 DATA_OUT => RD1_sig
826             );
850         --Register File instantiation
851         REG_inst: REG_FILE
852             Generic map(N => 4, M => 32)
853             Port map(
854                 CLK => CLK,
855                 RESET => RESET,
856                 WE3 => WE,
857                 ADDR_A1 => ADDR_A1,
858                 ADDR_A2 => ADDR_A2,
859                 ADDR_A3 => ADDR_A3,
860                 WD3 => WD3,
861                 R15 => R15,
862                 RD1 => RD1,
863                 RD2 => RD2
829             );
864         --Flags instantiation
865         Flags_inst: Extend
866             Generic map(Width_In => 24, Width_Out => 32)
867             Port map(
868                 ImmSrc => ImmSrc,
869                 S2_in => Result_sig(24 downto 1),
870                 S2_out => Flags
842             );
871         --PCPlus4 instantiation
872         PCPlus4_inst: MUX3x1
873             Generic map(width => 32)
874             Port map(
875                 A0 => PC_OUT_sig,
876                 A1 => PCPlus4_sig,
877                 A2 => PC_OUT_sig,
878                 SEL => PCSrc,
879                 Y => PC_IN_sig
850             );
880         --PCP4REG instantiation
881         PCP4REG_inst: MUX3x1
882             Generic map(width => 32)
883             Port map(
884                 A0 => PC_OUT_sig,
885                 A1 => PCPlus4_sig,
886                 A2 => PC_OUT_sig,
887                 SEL => PCSrc,
888                 Y => PCP4REG_sig
855             );
889         --PCPlus8 instantiation
890         PCPlus8_inst: MUX3x1
891             Generic map(width => 32)
892             Port map(
893                 A0 => PC_OUT_sig,
894                 A1 => PCPlus4_sig,
895                 A2 => PC_OUT_sig,
896                 SEL => PCSrc,
897                 Y => PCPlus8_sig
871             );
898         --MA instantiation
899         MA_inst: MUX3x1
900             Generic map(width => 32)
901             Port map(
902                 A0 => PC_OUT_sig,
903                 A1 => PCPlus4_sig,
904                 A2 => PC_OUT_sig,
905                 SEL => MAWrite,
906                 Y => MA_sig
884             );
907         --WD instantiation
908         WD_inst: MUX3x1
909             Generic map(width => 32)
910             Port map(
911                 A0 => PC_OUT_sig,
912                 A1 => PCPlus4_sig,
913                 A2 => PC_OUT_sig,
914                 SEL => MAWrite,
915                 Y => WD_sig
889             );
916         --Data_Memory instantiation
917         Data_Memory_inst: Data_Memory
918             Generic map(N => 5, M => 32)
919             Port map(
920                 CLK => CLK,
921                 WE => MemWrite,
922                 ADDR => ADDR_A1,
923                 DATA_IN => WriteData_sig,
924                 DATA_OUT => RD1_sig
901             );
925         --Register File instantiation
926         REG_inst: REG_FILE
927             Generic map(N => 4, M => 32)
928             Port map(
929                 CLK => CLK,
930                 RESET => RESET,
931                 WE3 => WE,
932                 ADDR_A1 => ADDR_A1,
933                 ADDR_A2 => ADDR_A2,
934                 ADDR_A3 => ADDR_A3,
935                 WD3 => WD3,
936                 R15 => R15,
937                 RD1 => RD1,
938                 RD2 => RD2
904             );
939         --Flags instantiation
940         Flags_inst: Extend
941
```

```

199      --PCPlus4 Register instantiation
200      PCP4_REG_inst: WE_REGN
201      Generic map(Width => 32)
202      Port map(
203          CLK    => CLK,
204          Reset  => RESET,
205          WE     => '1',
206          D      => PCPlus4_sig,
207          Q      => PCP4REG_sig
208      );
209
210      --Incrementer by 8 instantiation (pc+8)
211      Inc8_inst: INC4
212      Generic map(Width => 32)
213      Port map(
214          INC_IN  => PCP4REG_sig,
215          INC_OUT => PCPlus8_sig
216      );
217
218      --MuxRnx15 instantiation
219      MUX_Rnx15_inst: MUX2x1
220      Generic map(Width => 4)
221      Port map(
222          A0 => IR_OUT_sig(19 downto 16),
223          A1 => "1111",
224          SEL => RegSrc(0),
225          Y  => RA1_sig
226      );
227
228      --MuxRmxRd instantiation
229      MUX_RmxRd_inst: MUX2x1
230      Generic map(Width => 4)
231      Port map(
232          A0 => IR_OUT_sig(3  downto 0),
233          A1 => IR_OUT_sig(15 downto 12),
234          SEL => RegSrc(1),
235          Y  => RA2_sig
236      );
237
238      --MuxRdx14 instantiation
239      MUX_Rdx14_inst: MUX2x1
240      Generic map(Width => 4)
241      Port map(
242          A0 => IR_OUT_sig(15 downto 12),
243          A1 => "1110",
244          SEL => RegSrc(2),
245          Y  => WA_sig
246      );
247
248      --Register File instantiation
249      REG_FILE_inst: REG_FILE
250      Generic map(N => 4, M => 32)
251      Port map(
252          CLK    => CLK,
253          RESET  => RESET,
254          WE3    => RegWrite,
255          ADDR_A1 => RA1_sig,
256          ADDR_A2 => RA2_sig,
257          ADDR_A3 => WA_sig,
258          WD3    => WD3_sig,
259          R15    => PCPlus8_sig,
260          RD1    => RD1_sig,
261          RD2    => RD2_sig
262      );
263
264      --Extend module instantiation
265      Extend_inst: Extend
266      Generic map(Width_In  => 24, Width_Out => 32)
267      Port map(
268          ImmSrc => ImmSrc,
269          SZ_in  => IR_OUT_sig(23 downto 0),
270          SZ_out => IREG_IN_sig
271      );
272
273      --Register A instantiation
274      RegA_inst: WE_REGN
275      Generic map(Width => 32)
276      Port map(
277          CLK    => CLK,
278          Reset  => RESET,
279          WE     => '1',
280          D      => RD1_sig,
281          Q      => SrcA_sig
282      );
283
284      --Register B instantiation
285      RegB_inst: WE_REGN
286      Generic map(Width => 32)
287      Port map(
288          CLK    => CLK,
289          Reset  => RESET,
290          WE     => '1',
291          D      => RD2_sig,
292          Q      => WriteData_sig
293      );
294
295      --Register I instantiation
296      RegI_inst: WE_REGN
297      Generic map(Width => 32)
298      Port map(
299          CLK    => CLK,
300          Reset  => RESET,
301          WE     => '1',
302          D      => IREG_IN_sig,
303          Q      => ExtImm_sig
304      );
305
306      --MuxRD2xExtImm instantiation
307      MuxRD2xExtImm_inst: MUX2x1
308      Generic map(Width => 32)
309      Port map(
310          A0 => WriteData_sig,
311          A1 => ExtImm_sig,
312          SEL => ALUSrc,
313          Y  => SrcB_sig
314      );
315
316      --Arithmetic Logic Unit instantiation
317      ALU_inst: ALU
318      Generic map(Width => 32)
319      Port map(
320          CLK    => CLK,
321          SrcA    => SrcA_sig,
322          SrcB    => SrcB_sig,
323          ALUControl => ALUControl,
324          ALUResult => ALUResult_sig,
325          Flags   => Flags_sig
326      );
327
328      --Status Register instantiation
329      SR_REG_inst: WE_REGN
330      Generic map(Width => 4)
331      Port map(
332          CLK    => CLK,
333          Reset  => RESET,
334          WE     => FlagsWrite,
335          D      => Flags_sig,
336          Q      => Flags
337      );

```

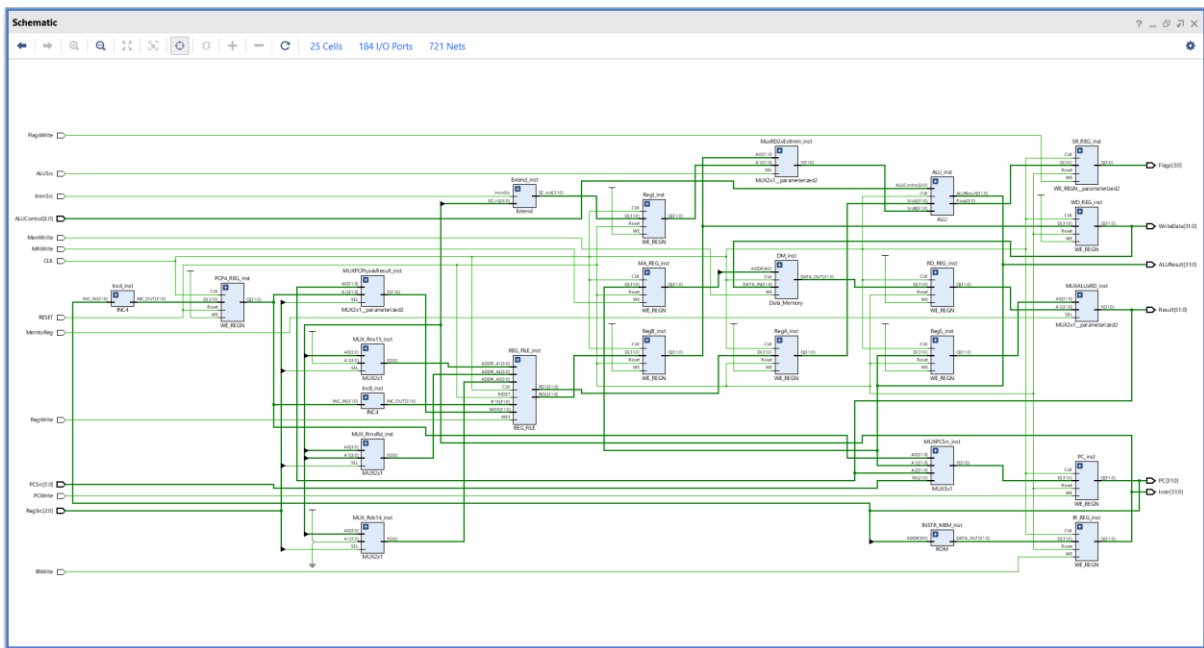
```

339 --Memory Address Register (MA) instantiation
340 MA_REG_inst: WE_REGN
341 Generic map (Width => 32)
342 Port map(
343     CLK => CLK,
344     Reset => RESET,
345     WE => MAWrite,
346     D => ALUResult_sig,
347     Q => MA_sig
348 );
349
350 --Memory Write Data Register (WD) instantiation
351 WD_REG_inst: WE_REGN
352 Generic map (Width => 32)
353 Port map(
354     CLK => CLK,
355     Reset => RESET,
356     WE => '1',
357     D => WriteData_sig,
358     Q => WD_sig
359 );
360
361 --Data Memory (DM) instantiation (Distributed RAM)
362 DM_inst: Data_Memory
363 Generic map (N => 5, M => 32)
364 Port map(
365     CLK => CLK,
366     WE => MemWrite,
367     ADDR => MA_sig(6 downto 2),
368     DATA_IN => WD_sig,
369     DATA_OUT => Data_Memory_sig
370 );
371
372 --Memory Read Data Register (RD) instantiation
373 RD_REG_inst: WE_REGN
374 Generic map (Width => 32)
375 Port map(
376     CLK => CLK,
377     Reset => RESET,
378     WE => '1',
379     D => Data_Memory_sig,
380     Q => RD_sig
381 );
382
383 --Register S instantiation
384 RegS_inst: WE_REGN
385 Generic map (Width => 32)
386 Port map(
387     CLK => CLK,
388     Reset => RESET,
389     WE => '1',
390     D => ALUResult_sig,
391     Q => SREG_sig
392 );
393
394 --MUXAluResultxReadDataMem instantiation
395 MUXALUxRD_inst: MUX2x1
396 Generic map (Width => 32)
397 Port map(
398     A0 => SREG_sig,
399     A1 => RD_sig,
400     SEL => MemtoReg,
401     Y => Result_sig
402 );
403
404 --MUX(PC+4)xResult to write WD3 Register File
405 MUXPCPlus4xResult_inst: MUX2x1
406 Generic map (Width => 32)
407 Port map(
408     A0 => Result_sig,
409     A1 => PCP4REG_sig,
410     SEL => RegSrc(2),
411     Y => WD3_sig
412 );
413
414 --Mux To choose next instruction's address
415 MUXPCSrc_inst: MUX3x1
416 Generic map (Width => 32)
417 Port map(
418     A0 => PCP4REG_sig,
419     A1 => ALUResult_sig,
420     A2 => RESULT_sig,
421     SEL => PCSrc,
422     Y => PC_IN_sig
423 );
424
425 --DataPath Output Signals assertion
426 PC <= PC_OUT_sig;
427 Instr <= IR_OUT_sig;
428 ALUResult <= ALUResult_sig;
429 WriteData <= WD_sig;
430 Result <= Result_sig;
431
432 end Structural;

```

VHDL υλοποίηση της Διαδρομής Δεδομένων (DataPath)





Data Path RTL

Στο RTL διάγραμμα διακρίνονται καθαρά όλες οι υπομονάδες της δομής της διαδρομής δεδομένων του επεξεργαστή, τα σήματα εισόδου/εξόδου καθώς και η επικοινωνία των μονάδων με τη βοήθεια των εσωτερικών σημάτων που ορίστηκαν στην υλοποίηση.

### 1.5 Περιγραφή των υπομονάδων της συνδυαστικής λογικής της μονάδας ελέγχου

Η συνδυαστική λογική της μονάδας ελέγχου αποτελείται από :

- Τον αποκωδικοποιητή εντολών (Instruction Decoder) , ο οποίος παράγει τα κατάλληλα σήματα με βάση τα οποία η διαδρομή δεδομένων υλοποιεί την ορθή εκτέλεση της εντολής.
- Τη λογική μονάδα ελέγχου συνθήκης (CONDLogic), η οποία ελέγχει εάν ικανοποιείται η συνθήκη που ορίζεται στο πεδίο cond της εντολής με βάση τις τρέχουσες τιμές των σημαιών N, Z, C, V (flags). Το σήμα εξόδου CondEx\_in εγκρίνει την εκτέλεση της εντολής (υπό συνθήκη), όταν παίρνει την τιμή 1.

#### Instruction Decoder

Εντολή	Instr <sub>27:26</sub> op	Instr <sub>25:20</sub> funct	Τύπος	RegSrc	ALUSrc	ImmSrc	ALUControl	MemtoReg	NoWrite_in
ADD	00	10100X	DP Imm	0X0	1	0	0000	0	0
ADD	00	00100X	DP Reg	000	0	X	0000	0	0
SUB	00	10010X	DP Imm	0X0	1	0	0001	0	0
SUB	00	00010X	DP Reg	000	0	X	0001	0	0
CMP	00	110101	DP Imm	0X0	1	0	0001	X	1
CMP	00	010101	DP Reg	000	0	X	0001	X	1
AND	00	10000X	DP Imm	0X0	1	0	1010	0	0
AND	00	00000X	DP Reg	000	0	X	1010	0	0
ORR	00	11100X	DP Imm	0X0	1	0	1011	0	0
ORR	00	01100X	DP Reg	000	0	X	1011	0	0
XOR	00	10001X	DP Imm	0X0	1	0	1110	0	0
XOR	00	00001X	DP Reg	000	0	X	1110	0	0
MOV	00	111010	DP Imm	0X0	1	0	0111	0	0
MOV	00	011010	DP Reg	000	0	X	0111	0	0
MOVN	00	111110	DP Imm	0X0	1	0	1001	0	0
MOVN	00	011110	DP Reg	000	0	X	1001	0	0
LSL	00	011010	DP Reg	0X0	0	0	0010	0	0
ASR	00	011010	DP Reg	0X0	0	0	0100	0	0
LDR	01	011001	M imm+	0X0	1	0	0000	1	0
LDR	01	010001	M imm-	0X0	1	0	0001	1	0
STR	01	011000	M imm+	010	1	0	0000	X	0
STR	01	010000	M imm-	010	1	1	0001	X	0
B	10	10XXXX	B imm+	0X1	1	1	0000	0	0
BL	10	11XXXX	B imm+	1X1	1	1	0000	0	0

Πίνακας2:Πίνακας αλήθειας του Instruction Decoder



```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity InstrDec is
5  Port (
6      op          : in std_logic_vector(1 downto 0);
7      funct       : in std_logic_vector(5 downto 0);
8      sh          : in std_logic_vector(1 downto 0);
9      RegSrc      : out std_logic_vector(2 downto 0);
10     ALUSrc       : out std_logic;
11     ImmSrc       : out std_logic;
12     ALUControl   : out std_logic_vector(3 downto 0);
13     MemtoReg     : out std_logic;
14     NoWrite_in   : out std_logic
15 );
16 end InstrDec;
17
18 architecture Behavioral of InstrDec is
19 begin
20     process (op, funct, sh)
21     begin
22         if(op = "00") then -- Data Processing Instructions
23             if(funct(5) = '1') then
24                 ALUSrc <= '1';
25             else
26                 ALUSrc <= '0';
27             end if;
28             if(funct(0) = '1') then
29                 NoWrite_in <= '1';
30             else
31                 NoWrite_in <= '0';
32             end if;
33             case funct(4 downto 1) is
34                 when "0100" =>
35                     ALUControl <= "0000"; -- ADD
36                 when "0010" =>
37                     ALUControl <= "0001"; -- SUB
38                 when "1010" =>
39                     ALUControl <= "0001"; -- CMP
40
41                 when "0000" =>
42                     ALUControl <= "1010"; -- AND
43                 when "1100" =>
44                     ALUControl <= "1011"; -- ORR
45                 when "0001" =>
46                     ALUControl <= "1110"; -- XOR
47                 when "1101" =>
48                     if(sh = "00") then
49                         ALUControl <= "0111"; -- MOV
50                     elsif(sh = "01") then
51                         ALUControl <= "0011"; -- LSR
52                     elsif(sh = "10") then
53                         ALUControl <= "0100"; -- ASR
54                     else
55                         ALUControl <= "0000";
56                     end if;
57                 when "1111" =>
58                     ALUControl <= "1001"; -- MONV
59                 when others =>
60                     ALUControl <= "0000";
61             end case;
62             RegSrc <= "000";
63             ImmSrc <= '0';
64             MemtoReg <= '0';
65
66             elsif(op = "01")then --Memory Instructions
67                 if(funct(3) = '1')then
68                     ALUControl <= "0000"; --LDR/STR(+)
69                 else
70                     ALUControl <= "0001"; --LDR/STR(-)
71                 end if;
72                 RegSrc <= "010";
73                 ALUSrc <= '1';
74                 ImmSrc <= '0';
75                 MemtoReg <= '1';
76                 NoWrite_in <= '0';
77             elsif(op = "10") then -- Branch Instructions
78                 if(funct(4) = '1')then
79                     RegSrc <= "101";
80                 else
81                     RegSrc <= "001";
82                 end if;
83                 ALUSrc <= '1';
84                 ImmSrc <= '1';
85                 ALUControl <= "0000";
86                 MemtoReg <= '0';
87                 NoWrite_in <= '0';
88             else
89                 ALUControl <= "0000";
90                 ImmSrc <= '0';
91                 RegSrc <= "000";
92                 NoWrite_in <= '0';
93                 MemtoReg <= '0';
94                 ALUSrc <= '0';
95             end if;
96         end process;
97     end Behavioral;

```

VHDL υλοποίηση του Instruction Decoder.

## CONDLogic

Στον παρακάτω πίνακα φαίνονται τα μνημονικά συνθήκης με τις εξισώσεις Boole των σημαιών που τις ικανοποιούν. Η σχεδίαση της λογικής ελέγχου συνθήκης σε VHDL πραγματοποιείται βάσει των μνημονικών συνθήκης.

cond <sub>3:0</sub>	Μνημονικό	Όνομα	CondEx
0000	EQ	Equal	$Z$
0001	NE	Not equal	$\bar{Z}$
0010	CS/HS	Carry set / unsigned higher or same	$C$
0011	CC/LO	Carry clear / unsigned lower	$\bar{C}$
0100	MI	Minus / negative	$N$
0101	PL	Plus / positive or zero	$\bar{N}$
0110	VS	Overflow / overflow set	$V$
0111	VC	No overflow / overflow clear	$\bar{V}$

cond <sub>3:0</sub>	Μνημονικό	Όνομα	CondEx
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z+\bar{C}$
1010	GE	Signed greater or equal	$\bar{N}\oplus\bar{V}$
1011	LT	Signed less	$N\oplus V$
1100	GT	Signed greater	$\bar{Z}N\oplus\bar{V}$
1101	LE	Signed less or equal	$Z+(N\oplus V)$
1110	AL (ή none)	Always / unconditional	1
1111	none	For unconditional instructions	1

Πίνακας 3: Μνημονικά συνθήκης με τις εξισώσεις boole των σημαιών που τις ικανοποιούν.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity CONDLogic is
5  Port (
6      cond      : in std_logic_vector(3 downto 0);
7      flags      : in std_logic_vector(3 downto 0);
8      CondEx_in : out std_logic
9  );
10 end CONDLogic;
11
12 architecture Behavioral of CONDLogic is
13     -- N,Z,C,V
14     begin
15         CondExec_proc: process(cond, flags)
16         begin
17             case cond is
18                 when "0000" => CondEx_in <= flags(1); -- (Z)
19                 when "0001" => CondEx_in <= not(flags(1)); -- (notZ)
20                 when "0010" => CondEx_in <= flags(2); -- (C)
21                 when "0011" => CondEx_in <= not(flags(2)); -- (notC)
22                 when "0100" => CondEx_in <= flags(0); -- (N)
23                 when "0101" => CondEx_in <= not(flags(0)); -- (notN)
24                 when "0110" => CondEx_in <= flags(3); -- (V)
25                 when "0111" => CondEx_in <= not(flags(3)); -- (notV)
26                 when "1000" => CondEx_in <= not(flags(1)) and flags(2); -- (notZ) AND (C)
27                 when "1001" => CondEx_in <= flags(1) or not(flags(2)); -- (Z) OR (notC)
28                 when "1010" => CondEx_in <= not(flags(0) xor flags(3)); -- (not(N) xor (V))
29                 when "1011" => CondEx_in <= flags(0) xor flags(3); -- (N) xor (V)
30                 when "1100" => CondEx_in <= not(flags(1)) and not(flags(0) xor flags(3)); -- (not(Z)) AND (not((N) XOR (V)))
31                 when "1101" => CondEx_in <= flags(1) or (flags(0) xor flags(3)); -- (Z) OR ((N) XOR (V))
32                 when "1110" => CondEx_in <= '1';
33                 when "1111" => CondEx_in <= '1';
34                 when others => CondEx_in <= '1';
35             end case;
36         end process;
37     end Behavioral;
```

## 1.6 Περιγραφή της μηχανής πεπερασμένων καταστάσεων (FSM) της μονάδας ελέγχου.

Η μηχανή πεπερασμένων καταστάσεων της μονάδας ελέγχου αποτελεί τον «ηλεκτρονικό εγκέφαλο» του επεξεργαστή. Συσχετίζει τα βήματα εκτέλεσης των εντολών με τους απαιτούμενους κύκλους του ρολογιού ενεργοποιώντας κατάλληλα τα σήματα έγκρισης εγγραφής των διαφόρων υπομονάδων της διαδρομής δεδομένων, καθώς και το σήμα επιλογής διεύθυνσης επόμενης εντολής PCSrc[1:0]. Σε περίπτωση που το σήμα CondEx\_in = 0 η εκτέλεση της εντολής τερματίζεται πρόωρα με την εκτέλεση του τελευταίου βήματος (βήμα 5) όπου ενεργοποιείται το PCWrite έτσι ώστε στον επόμενο κύκλο να εκτελεσθεί το βήμα 1 της αμέσως επόμενης εντολής (διεύθυνση PC+4).

Το FSM αποτελεί ένα σύγχρονο ακολουθιακό κύκλωμα με σύγχρονο RESET και η μετάβαση στην αμέσως επόμενη κατάσταση πραγματοποιείται σε κάθε κατερχόμενη ακμή του ρολογιού. Η συνδυαστική λογική της επιλογής της επόμενης κατάστασης υλοποιήθηκε σύμφωνα με τον οδηγό της εργασίας. Παρακάτω παρουσιάζεται ο πίνακας αλήθειας της FSM δηλαδή οι τιμές των σημάτων εξόδου που ανταποκρίνονται σε κάθε δυνατό state.

Current state	IRWrite	RegWrite	MAWrite	MemWrite	FlagsWrite	PCSrc	PCWrite
S0	1	0	0	0	0	00	0
S1	0	0	0	0	0	00	0
S2a	0	0	1	0	0	00	0
S2b	0	0	0	0	0	00	0
S3	0	0	0	0	0	00	0
S4a	0	1	0	0	0	00	1
S4b	0	0	0	0	0	01	1
S4c	0	0	0	0	0	00	1
S4d	0	0	0	1	0	00	1
S4e	0	1	0	0	1	00	1
S4f	0	0	0	0	1	01	1
S4g	0	0	0	0	1	00	1
S4h	0	0	0	0	0	11	1
S4i	0	1	0	0	0	11	1

Πίνακας 4: Πίνακας αλήθειας του FSM. Το state S4i προστέθηκε ώστε να υλοποιείται η εντολή BL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity FSM is
5  Port (
6      CLK      : in  std_logic;
7      RESET    : in  std_logic;
8      op       : in  std_logic_vector(1 downto 0);
9      S        : in  std_logic;
10     L        : in  std_logic;
11     Rd       : in  std_logic_vector(3 downto 0);
12     NoWrite_in : in  std_logic;
13     CondEx_in : in  std_logic;
14     PCWrite   : out std_logic;
15     IRWrite   : out std_logic;
16     RegWrite  : out std_logic;
17     FlagsWrite : out std_logic;
18     MAWrite   : out std_logic;
19     MemWrite  : out std_logic;
20     PCSrc     : out std_logic_vector(1 downto 0)
21 );
22 end FSM;
23
24 architecture Behavioral of FSM is
25     type FSM_states is (S0,S1,S2a,S2b,S3,S4a,S4b,S4c,S4d,S4e,
26                         S4f,S4g,S4h,S4i);
27     signal current_state, next_state : FSM_states;
28     signal op_sig : std_logic_vector(1 downto 0);
29     signal Rd_sig : std_logic_vector(3 downto 0);
30     signal S_sig : std_logic;
31     signal L_sig : std_logic;
32     signal NoW : std_logic;
33     signal cEx : std_logic;
34 begin
```



```

161 when S4c =>
162     IRWrite    <= '0';
163     RegWrite   <= '0';
164     MAWrite    <= '0';
165     MemWrite   <= '0';
166     FlagsWrite <= '0';
167     PCSrc      <= "00";
168     PCWrite    <= '1';
169     next_state <= S0;
170 when S4d =>
171     IRWrite    <= '0';
172     RegWrite   <= '0';
173     MAWrite    <= '0';
174     MemWrite   <= '1';
175     FlagsWrite <= '0';
176     PCSrc      <= "00";
177     PCWrite    <= '1';
178     next_state <= S0;
179 when S4e =>
180     IRWrite    <= '0';
181     RegWrite   <= '1';
182     MAWrite    <= '0';
183     MemWrite   <= '0';
184     FlagsWrite <= '1';
185     PCSrc      <= "00";
186     PCWrite    <= '1';
187     next_state <= S0;
188 when S4f =>
189     IRWrite    <= '0';
190     RegWrite   <= '0';
191     MAWrite    <= '0';
192     MemWrite   <= '0';
193     FlagsWrite <= '1';
194     PCSrc      <= "10";
195     PCWrite    <= '1';
196     next_state <= S0;

197 when S4g =>
198     IRWrite    <= '0';
199     RegWrite   <= '0';
200     MAWrite    <= '0';
201     MemWrite   <= '0';
202     FlagsWrite <= '1';
203     PCSrc      <= "00";
204     PCWrite    <= '1';
205     next_state <= S0;
206 when S4h =>
207     IRWrite    <= '0';
208     RegWrite   <= '0';
209     MAWrite    <= '0';
210     MemWrite   <= '0';
211     FlagsWrite <= '0';
212     PCSrc      <= "11";
213     PCWrite    <= '1';
214     next_state <= S0;
215 when S4i =>
216     IRWrite    <= '0';
217     RegWrite   <= '1';
218     MAWrite    <= '0';
219     MemWrite   <= '0';
220     FlagsWrite <= '0';
221     PCSrc      <= "11";
222     PCWrite    <= '1';
223     next_state <= S0;
224 when others =>
225     next_state <= S0;
226 end case;
227 end process;
228 end Behavioral;

```

## VHDL υλοποίηση του FSM του Control Unit

### 1.7 Περιγραφή της δομής της μονάδας ελέγχου (Control) του επεξεργαστή.

Στην ενότητα αυτή παρουσιάζεται η VHDL υλοποίηση της μονάδας ελέγχου δηλαδή του ενός από τα δύο βασικά modules του επεξεργαστή. Η σχεδίαση έχει γίνει σε ανώτερο ιεραρχικό επίπεδο (structural) και περιλαμβάνει τα submodules :

- Instruction Decoder
- Conditional Logic
- FSM

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Control is
5  Port (
6      CLK       : in std_logic;
7      RESET     : in std_logic;
8      IR        : in std_logic_vector(31 downto 0);
9      SR        : in std_logic_vector(3  downto 0);
10     RegSrc     : out std_logic_vector(2 downto 0);
11     ALUSrc     : out std_logic;
12     MemtoReg   : out std_logic;
13     ALUControl : out std_logic_vector(3 downto 0);
14     ImmSrc     : out std_logic;
15     IRWrite    : out std_logic;
16     RegWrite   : out std_logic;
17     MAWrite    : out std_logic;
18     MemWrite   : out std_logic;
19     FlagsWrite : out std_logic;
20     PCSrc      : out std_logic_vector(1 downto 0);
21     PCWrite    : out std_logic;
22 );
23 end Control;

```

```

25 architecture Structural of Control is
26
27 --Submodules declaration
28 component InstrDec
29 Port(
30     op          : in std_logic_vector(1 downto 0);
31     funct       : in std_logic_vector(5 downto 0);
32     sh          : in std_logic_vector(1 downto 0);
33     RegSrc      : out std_logic_vector(2 downto 0);
34     ALUSrc      : out std_logic;
35     ImmSrc      : out std_logic;
36     AluControl  : out std_logic_vector(3 downto 0);
37     MemtoReg    : out std_logic;
38     NoWrite_in  : out std_logic
39 );
40 end component;
41
42 component CONDLogic
43 Port(
44     cond        : in std_logic_vector(3 downto 0);
45     flags       : in std_logic_vector(3 downto 0);
46     CondEx_in   : out std_logic
47 );
48 end component;
49
50 component FSM
51 Port(
52     CLK         : in std_logic;
53     RESET       : in std_logic;
54     op          : in std_logic_vector(1 downto 0);
55     S           : in std_logic;
56     L           : in std_logic;
57     Rd          : in std_logic_vector(3 downto 0);
58     NoWrite_in  : in std_logic;
59     CondEx_in   : in std_logic;
60     PCWrite     : out std_logic;
61     IRWrite     : out std_logic;
62     RegWrite    : out std_logic;
63     FlagsWrite  : out std_logic;
64     MAWrite     : out std_logic;
65     MemWrite    : out std_logic;
66     PCSrc       : out std_logic_vector(1 downto 0)
67 );
68 end component;
69
70 signal NoWrite_in_sig : std_logic := '0';
71 signal CondEx_in_sig  : std_logic := '0';
72 begin
73
74     InstructionDecoder_inst: InstrDec
75     port map(
76         op      => IR(27 downto 26),
77         funct   => IR(25 downto 20),
78         sh      => IR(6  downto 5),
79         RegSrc  => RegSrc,
80         ALUSrc  => ALUSrc,
81         MemtoReg => MemtoReg,
82         ALUControl => ALUControl,
83         ImmSrc  => ImmSrc,
84         NoWrite_in => NoWrite_in_sig
85     );
86

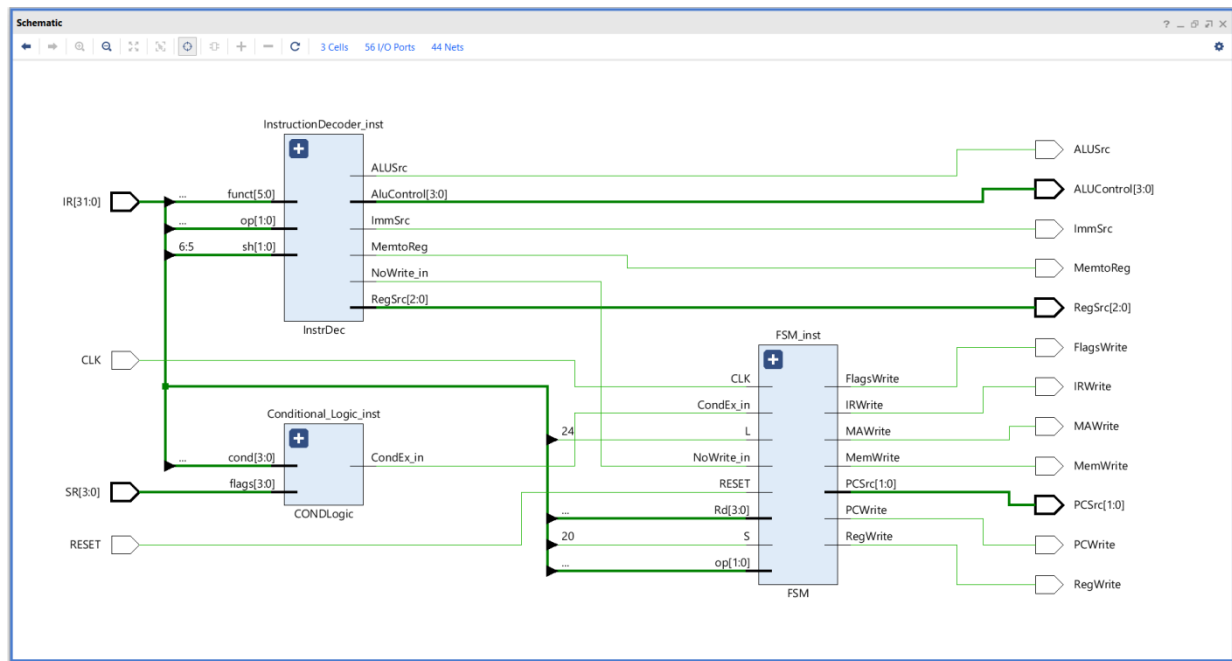
```

```

87     FSM_inst: FSM
88     port map(
89         CLK      => CLK,
90         RESET    => RESET,
91         op       => IR(27 downto 26),
92         S        => IR(20),
93         L        => IR(24),
94         Rd       => IR(15 downto 12),
95         NoWrite_in => NoWrite_in_sig,
96         CondEx_in => CondEx_in_sig,
97         IRWrite  => IRWrite,
98         RegWrite => RegWrite,
99         MAWrite  => MAWrite,
100        MemWrite => MemWrite,
101        FlagsWrite => FlagsWrite,
102        PCSrc    => PCSrc,
103        PCWrite  => PCWrite
104    );
105
106    Conditional_Logic_inst: CONDLogic
107    port map(
108        cond      => IR(31 downto 28),
109        flags     => SR,
110        CondEx_in => CondEx_in_sig
111    );
112
113 end Structural;

```

*VHDL υλοποίηση του Control Unit*



### Control Unit RTL

```

88 -- Stimulus process definition
89 Stimulus_process: process
90 begin
91 -- Synchronous RESET is deasserted on CLK falling edge
92 -- after GSR signal disable (it remains enabled for 100 ns)
93 RESET <= '1';
94 wait for 100 ns;
95 wait until (CLK = '0' and CLK'event);
96 RESET <= '0'; -- current state S0, nxState S1
97 wait for 1*CLK_period;
98 IR(27 downto 26) <= "01"; --current state S1, next state S2a
99 IR(31 downto 28) <= "1111";
100 wait for 1*CLK_period;
101 IR(24) <= '1'; -- current state S2a , nxState S3
102 wait for 1*CLK_period;
103 IR(15 downto 12) <= "0000"; --ourState S3, nxState S4a
104 wait for 1*CLK_period;
105 --ourState S4a, nxState S0
106 wait for 1*CLK_period;
107 --ourState S0, nxState S1
108 wait for 1*CLK_period;
109 IR(27 downto 26) <= "01"; --current state S1, next state S2a
110 IR(31 downto 28) <= "1111";
111 wait for 1*CLK_period;
112 IR(24) <= '1'; -- current state S2a , nxState S3
113 wait for 1*CLK_period;
114 IR(15 downto 12) <= "1111"; --ourState S3, nxState S4b
115 wait for 1*CLK_period;
116 --ourState S4a, nxState S0
117 wait for 1*CLK_period;
118 --ourState S0, nxState S1
119 wait for 1*CLK_period;
120 IR(27 downto 26) <= "00"; --currStateS1, nxState S2b
121 IR(31 downto 28) <= "1111";
122 IR(20) <= '0';
123 wait for 1*CLK_period;
124 IR(20) <= '0';
125 IR(15 downto 12) <= "0000"; --ourState S2b, nxState S4a
126 wait for 1*CLK_period;
127
128 wait for 1*CLK_period;
129 --ourState S4a, nxState S0
130 wait for 1*CLK_period;
131 --ourState S0, nxState S1
132 wait for 1*CLK_period;
133 IR(27 downto 26) <= "00"; --currStateS1, nxState S2b
134 IR(31 downto 28) <= "1111";
135 IR(20) <= '0';
136 wait for 1*CLK_period;
137 IR(20) <= '0';
138 IR(15 downto 12) <= "1111"; --ourState S2b, nxState S4b
139 wait for 1*CLK_period;
140 --ourState S4b, nxState S0
141 wait for 1*CLK_period;
142 --ourState S0, nxState S1
143 wait for 1*CLK_period;
144 IR(27 downto 26) <= "00"; --currStateS1, nxState S2b
145 IR(31 downto 28) <= "1111";
146 IR(20) <= '0';
147 wait for 1*CLK_period;
148 IR(20) <= '1';
149 IR(15 downto 12) <= "0000"; --ourState S2b, nxState S4e
150 wait for 1*CLK_period;
151 --ourState S4e, nxState S0
152 wait for 1*CLK_period;
153 --ourState S0, nxState S1
154 wait for 1*CLK_period;
155 IR(27 downto 26) <= "00"; --currStateS1, nxState S2b
156 IR(31 downto 28) <= "1111";
157 IR(20) <= '0';
158 wait for 1*CLK_period;
159 IR(20) <= '1';
160 IR(15 downto 12) <= "1111"; --ourState S2b, nxState S4f
161 wait for 1*CLK_period;
162 --ourState S4f, nxState S0
163 wait for 1*CLK_period;
164 --ourState S0, nxState S1
165 wait for 1*CLK_period;
166 IR(27 downto 26) <= "10"; --ourState S1, nxState S4h
167 IR(31 downto 28) <= "1111";

```

### Control Unit Testbench (Stimulus Process).

Εφαρμόζουμε κατάλληλα σήματα εισόδου ώστε το FSM module να περάσει από όλες τις δυνατές καταστάσεις του. Στο τέλος επαναφέρουμε το σήμα RESET στο 1 και επιβεβαιώνουμε ότι το επόμενο state είναι το S0 ανεξάρτητα απ'τα σήματα του sensitivity list στο συνδυαστικό process επιλογής επόμενης κατάστασης.



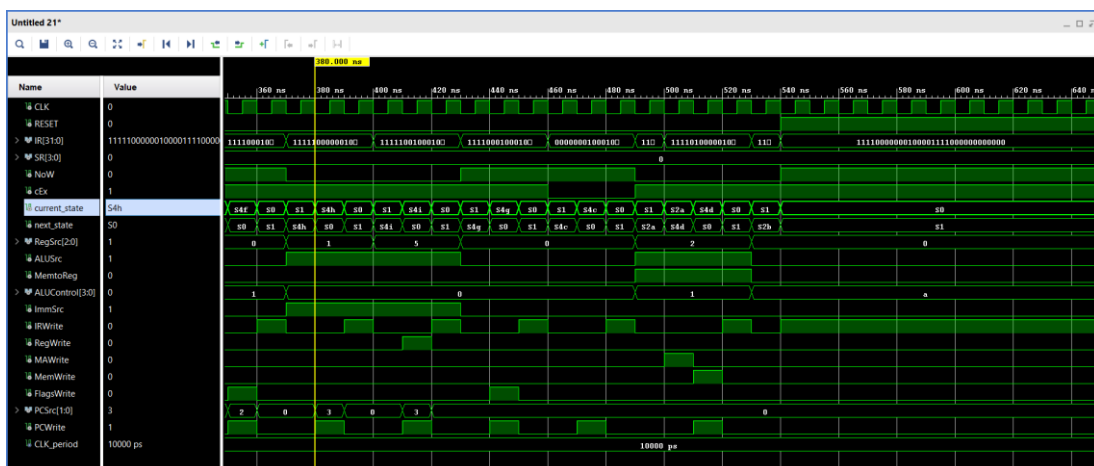
```

165 IR(31 downto 28) <= "1111";
166 IR(24) <= '0';
167 wait for 1*CLK_period;
168 --ourState S4h, nxState S0
169 wait for 1*CLK_period;
170 --ourState S0, nxState S1
171 wait for 1*CLK_period;
172 IR(27 downto 26) <= "10"; --ourState S1, nxState S4i
173 IR(31 downto 28) <= "1111";
174 IR(24) <= '1';
175 wait for 1*CLK_period;
176 --ourState S4i, nxState S0
177 wait for 1*CLK_period;
178 --ourState S0, nxState S1
179 wait for 1*CLK_period;
180 IR(27 downto 26) <= "00"; --ourState S1, nxState S4g
181 IR(31 downto 28) <= "1111";
182 IR(20) <= '1';
183 wait for 1*CLK_period;
184 --ourState S4g, nxState S0
185 wait for 1*CLK_period;
186 --ourState S0, nxState S1
187 wait for 1*CLK_period;
188 IR(31 downto 28) <= "0000"; --ourState S1, nxState S4o
189 wait for 1*CLK_period;
190 --ourState S4o, nxState S0
191 wait for 1*CLK_period;
192 --ourState S0, nxState S1
193 wait for 1*CLK_period;
194 IR(27 downto 26) <= "01"; --ourState S1, nxState S2a
195 IR(31 downto 28) <= "1111";
196 wait for 1*CLK_period;
197 IR(24) <= '0'; --ourState S2a, nxState S4d
198 wait for 1*CLK_period;
199 --ourState S4d, nxState S0
200 wait for 1*CLK_period;
201 --ourState S0, nxState S1
202 wait for 1*CLK_period;
203 IR(27 downto 26) <= "00"; --ourState S1, nxState S2b
204 IR(31 downto 28) <= "1111";
205 IR(20) <= '0';
206 wait for 1*CLK_period; --ourState S2b, nxState S0
207 IR(20) <= '1';
208 IR(15 downto 12) <= "1111";
209 RESET <= '1';
210 wait for 1*CLK_period;
211 wait for 100 ns;
212
213 -- Message and simulation end
214 report "TESTS COMPLETED";
215 stop(2);
216 end process;
217
218 end Behavioral;

```



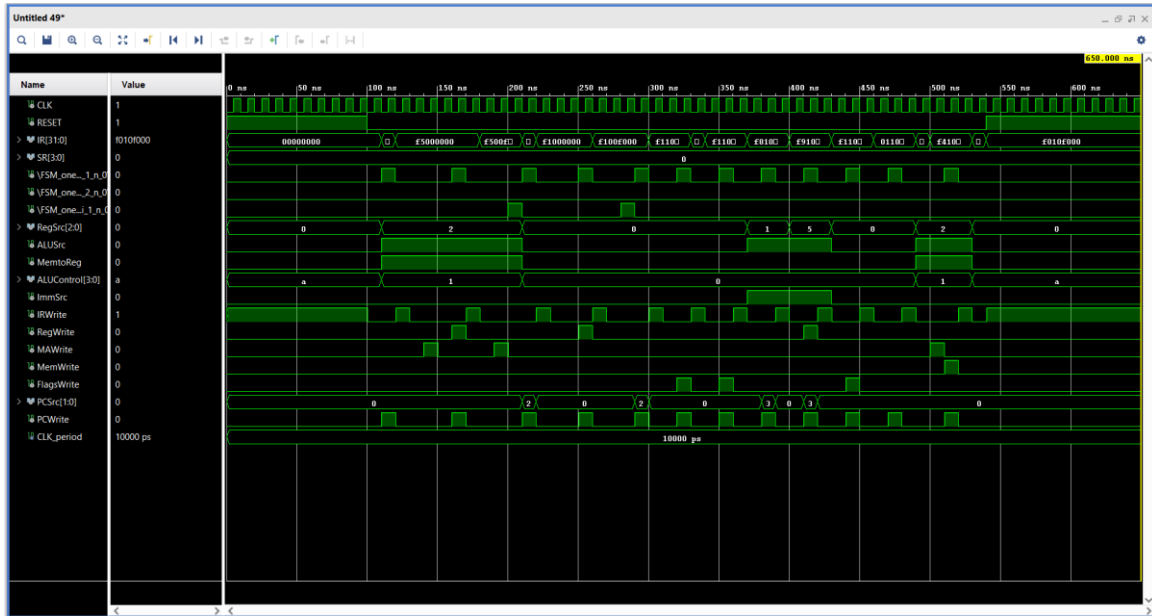
Control Unit Behavioral Simulation (FSM –based) (Part 1)



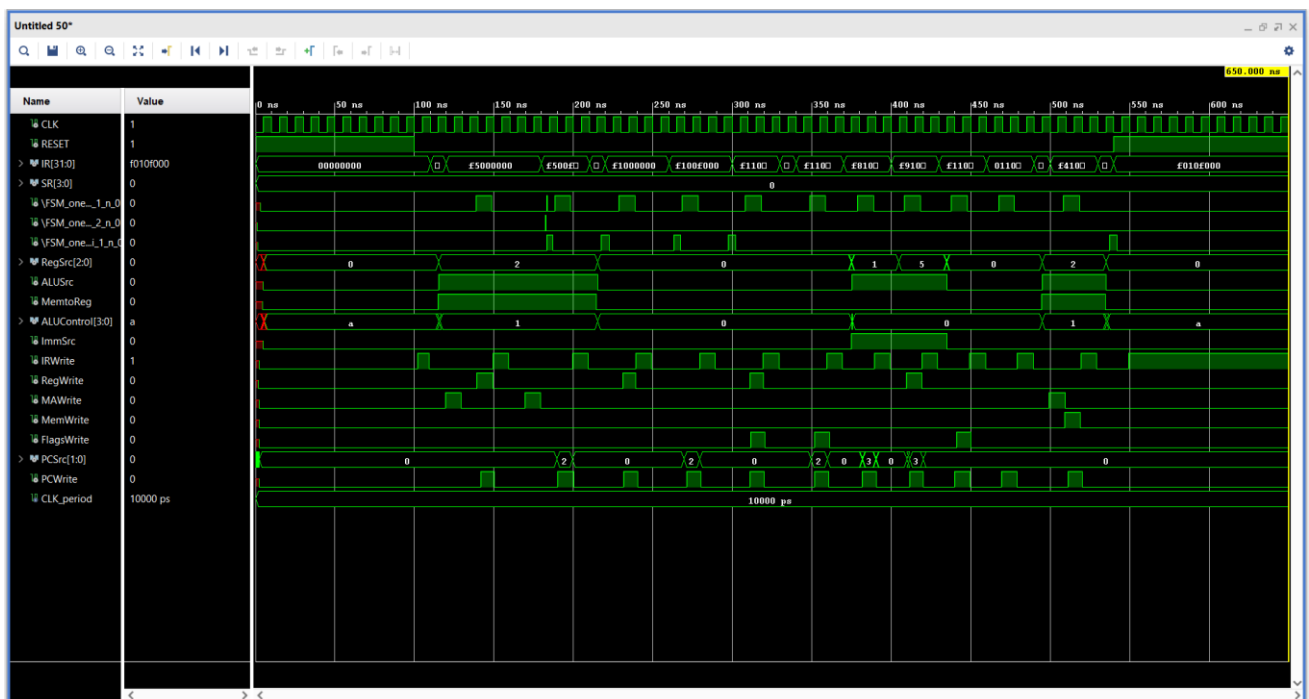
Control Unit Behavioral Simulation (FSM –based) (Part 2)



Το behavioural simulation ανταποκρίνεται ακριβώς στο testbench ενώ τα σήματα εξόδου είναι τα προβλεπόμενα, επομένως επιβεβαιώνεται ότι η λογική μονάδα λειτουργεί σύμφωνα με τις οδηγίες.



*Control Unit Post Synthesis Functional Simulation*



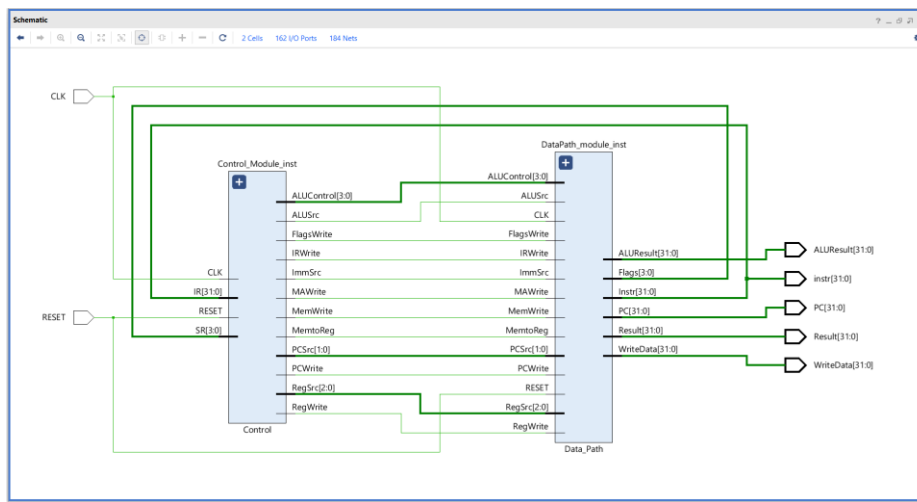
*Control Unit Post Implementation Timing Simulation*

## 1.8 Περιγραφή της δομής του επεξεργαστή (Processor).

Η δομή του επεξεργαστή σχεδιάστηκε στο ανώτερο ιεραρχικά επίπεδο και περιλαμβάνει ως στοιχεία τη διαδρομή δεδομένων (Data Path) και τη μονάδα ελέγχου (Control Unit). Οι μόνες εισοδοι του επεξεργαστή είναι το σήμα του ρολογιού (CLK) για τον χρονισμό ολόκληρης της μονάδας καθώς και το σήμα RESET. Οι έξοδοι είναι κάποια από τα σήματα που παράγει η μονάδα διαδρομής δεδομένων (Instruction, PC, ALUResult, Result, WriteData).

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Processor is
5  Port (
6      CLK      : in  std_logic;
7      RESET    : in  std_logic;
8      PC       : out std_logic_vector(31 downto 0);
9      instr    : out std_logic_vector(31 downto 0);
10     ALUResult : out std_logic_vector(31 downto 0);
11     WriteData : out std_logic_vector(31 downto 0);
12     Result    : out std_logic_vector(31 downto 0)
13 );
14 end Processor;
15
16 architecture Structural of Processor is
17
18     component Data_Path
19     Port (
20         CLK      : in  std_logic;
21         RESET    : in  std_logic;
22         PCWrite  : in  std_logic;
23         IRWrite  : in  std_logic;
24         PCSrc    : in  std_logic_vector(1 downto 0);
25         RegSrc   : in  std_logic_vector(2 downto 0);
26         RegWrite : in  std_logic;
27         MAWrite  : in  std_logic;
28         ImmSrc   : in  std_logic;
29         ALUSrc   : in  std_logic;
30         ALUControl : in  std_logic_vector(3 downto 0);
31         FlagsWrite : in  std_logic;
32         MemWrite : in  std_logic;
33         MemtoReg : in  std_logic;
34         Flags    : out std_logic_vector(3 downto 0);
35         PC       : out std_logic_vector(31 downto 0);
36         Instr    : out std_logic_vector(31 downto 0);
37         ALUResult : out std_logic_vector(31 downto 0);
38         WriteData : out std_logic_vector(31 downto 0);
39         Result    : out std_logic_vector(31 downto 0)
40     );
41 end component;
42
43 component Control
44 Port (
45     CLK      : in  std_logic;
46     RESET    : in  std_logic;
47     IR       : in  std_logic_vector(31 downto 0);
48     SR       : in  std_logic_vector(31 downto 0);
49     RegSrc   : out std_logic_vector(2 downto 0);
50     ALUSrc   : out std_logic;
51     MemtoReg : out std_logic;
52     ALUControl : out std_logic_vector(3 downto 0);
53     ImmSrc   : out std_logic;
54     IRWrite  : out std_logic;
55     RegWrite : out std_logic;
56     MAWrite  : out std_logic;
57     MemWrite : out std_logic;
58     FlagsWrite : out std_logic;
59     PCSrc    : out std_logic_vector(1 downto 0);
60     PCWrite  : out std_logic
61 );
62 end component;
63
64 signal PCWrite_sig      : std_logic := '0';
65 signal IRWrite_sig      : std_logic := '0';
66 signal PCSrc_sig        : std_logic_vector(1 downto 0) := (others => '0');
67 signal RegSrc_sig       : std_logic_vector(2 downto 0) := (others => '0');
68 signal RegWrite_sig     : std_logic := '0';
69 signal MAWrite_sig      : std_logic := '0';
70 signal ImmSrc_sig       : std_logic := '0';
71 signal ALUSrc_sig       : std_logic := '0';
72 signal ALUControl_sig   : std_logic_vector(3 downto 0) := (others => '0');
73 signal FlagsWrite_sig   : std_logic := '0';
74 signal MemWrite_sig     : std_logic := '0';
75 signal MemtoReg_sig     : std_logic := '0';
76 signal Flags_sig        : std_logic_vector(3 downto 0) := (others => '0');
77 signal PC_sig           : std_logic_vector(31 downto 0) := (others => '0');
78 signal Instr_sig        : std_logic_vector(31 downto 0) := (others => '0');
79 signal ALUResult_sig    : std_logic_vector(31 downto 0) := (others => '0');
80 signal WriteData_sig    : std_logic_vector(31 downto 0) := (others => '0');
81 signal Result_sig       : std_logic_vector(31 downto 0) := (others => '0');
82
83 begin
84     DataPath_module_inst: Data_Path
85     port map(
86         CLK      => CLK,
87         RESET    => RESET,
88         PCWrite  => PCWrite_sig,
89         IRWrite  => IRWrite_sig,
90         PCSrc    => PCSrc_sig,
91         RegSrc   => RegSrc_sig,
92         RegWrite => RegWrite_sig,
93         MAWrite  => MAWrite_sig,
94         ImmSrc   => ImmSrc_sig,
95         ALUSrc   => ALUSrc_sig,
96         ALUControl => ALUControl_sig,
97         FlagsWrite => FlagsWrite_sig,
98         MemWrite => MemWrite_sig,
99         MemtoReg => MemtoReg_sig,
100        Flags    => Flags_sig,
101        PC       => PC_sig,
102        Instr    => Instr_sig,
103        ALUResult => ALUResult_sig,
104        WriteData => WriteData_sig,
105        Result    => Result_sig
106    );
107
108     Control_Module_inst: Control
109     port map(
110         CLK      => CLK,
111         RESET    => RESET,
112         IR       => Instr_sig,
113         SR       => Flags_sig,
114         RegSrc   => RegSrc_sig,
115         ALUSrc   => ALUSrc_sig,
116         MemtoReg => MemtoReg_sig,
117         ALUControl => ALUControl_sig,
118         ImmSrc   => ImmSrc_sig,
119         IRWrite  => IRWrite_sig,
120         RegWrite => RegWrite_sig,
121         MAWrite  => MAWrite_sig,
122         MemWrite => MemWrite_sig,
123         FlagsWrite => FlagsWrite_sig,
124         PCSrc    => PCSrc_sig,
125         PCWrite  => PCWrite_sig
126    );
127
128     --Processor Output Signals Assertion
129     PC      <= PC_sig;
130     instr   <= Instr_sig;
131     ALUResult <= ALUResult_sig;
132     WriteData <= WriteData_sig;
133     Result  <= Result_sig;
134 end Structural;
```

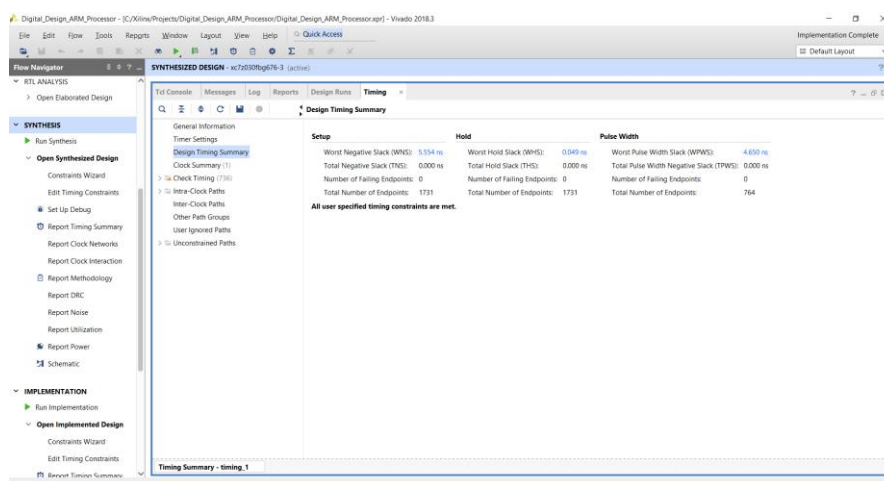
VHDL υλοποίηση του Processor



Processor RTL, διακρίνονται καθαρά οι βασικές μονάδες της δομής, τα σήματα εισόδου/εξόδου καθώς και τα σήματα επικοινωνίας μεταξύ των μονάδων

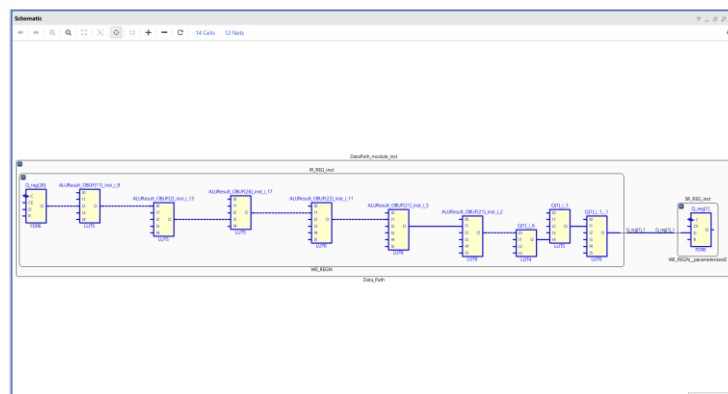
## 1.9 Εύρεση της μέγιστης συχνότητας λειτουργίας του επεξεργαστή.

Η πιο ακριβής ανάλυση χρονισμού που έχουμε διαθέσιμη είναι εκείνη που πραγματοποιεί το Vivado IDE στο implemented design model, επομένως πατάμε run implementation και στη συνέχεια ανοίγουμε το timing report.



Implementation Timing Report

Από το timing summary βλέπουμε ότι το critical path του κυκλώματος παίρνει χρόνο 5,54 ns. Δεδομένου ότι το critical path ορίζει και τη μεγαλύτερη δυνατή καθυστέρηση καταλαβαίνουμε ότι η μέγιστη συχνότητα λειτουργίας δε μπορεί να ξεπερνάει τα  $1/5,54\text{ns} = 180\text{ MHz}$ .



Critical Path (5,54 ns)

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

**2.1** Στο σημείο αυτό παρουσιάζεται το πρόγραμμα σε συμβολική γλώσσα αρχιτεκτονικής ARM που χρησιμοποιήθηκε για την επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή.

```
MAIN_PROGRAM:
MOV R0, #5; R0 = 5
MOV R1, #8; R1 = 8
ADD R2, R1, R0; R2 = 5+8 = 13
SUB R3, R1, R0; R3 = 8-5 = 3
MOV R3, R2; R3 = 13
STR R3, [R0]; DataMemory[5] = 13
LDR R4, [R0]; R4 = DataMemory[5] , R4 = 13
MOV R5, #15; R5 = 15
LSR_R0, R5, #2;
ASR_R0, R5, #2;
MOV R5, #0; R5 = 0
MOV R9, #0x000000FF; R9 = 255
MOV R0, R0
B MAIN_PROGRAM
```

Το πρόγραμμα αυτό ανέδειξε κάποια προβλήματα που υπήρχαν στη σχεδίαση. Αρχικά δε δούλευαν οι εντολές μνήμης STR και LDR. Μου πήρε αρκετή ώρα να αντιληφθώ ότι το πρόβλημα ήταν ότι κατά την εκτέλεση της STR η FSM ύστερα απ' την κατάσταση S2a δεν προχωρούσε στην S4d (η οποία ενεργοποιεί και το σήμα έγκρισης εγγραφής της μνήμης δεδομένων) αλλά περνούσε εσφαλμένα στην S3. Αυτό συνέβαινε διότι το σήμα L που καθορίζει και τη σωστή μετάβαση έπαιρνε την τιμή IR[24] αντί για την IR[20]. Το πρόβλημα διορθώθηκε πρόχειρα δεδομένης της έλλειψης χρόνου. Στη συνέχεια όμως αναδείχθηκαν κι άλλα προβλήματα σχετικά με τις εντολές LSR και ASR οι οποίες δε δίνουν τα αναμενόμενα αποτελέσματα, δυστυχώς δεν προλαβαίνω να το διορθώσω.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity ROM is
6   Generic( N : positive := 6; -- address length
7           M : positive := 32); -- data word length
8   Port (
9     ADDR      : in  std_logic_vector(N-1 downto 0);
10    DATA_OUT  : out std_logic_vector(M-1 downto 0));
11 end ROM;
12
13 architecture Behavioral of ROM is
14   type ROM_array is array (0 to 2**N-1)
15     of std_logic_vector(M-1 downto 0);
16   constant ROM : ROM_array := (
17     X"E3A00005", X"E3A01008", X"E0812000", X"E0413000",
18     X"E1A03002", X"E5803000", X"E5904000", X"E3A0500F",
19     X"E1A00125", X"E1A00145", X"E3A05000", X"E3A090FF",
20     X"E1A00000", X"EAF00000", X"00000000", X"00000000",
21     X"00000000", X"00000000", X"00000000", X"00000000",
22     X"00000000", X"00000000", X"00000000", X"00000000",
23     X"00000000", X"00000000", X"00000000", X"00000000",
24     X"00000000", X"00000000", X"00000000", X"00000000",
25     X"00000000", X"00000000", X"00000000", X"00000000",
26     X"00000000", X"00000000", X"00000000", X"00000000",
27     X"00000000", X"00000000", X"00000000", X"00000000",
28     X"00000000", X"00000000", X"00000000", X"00000000",
29     X"00000000", X"00000000", X"00000000", X"00000000",
30     X"00000000", X"00000000", X"00000000", X"00000000",
31     X"00000000", X"00000000", X"00000000", X"00000000",
32     X"00000000", X"00000000", X"00000000", X"00000000");
33 begin
34   DATA_OUT <= ROM(to_integer(unsigned(ADDR)));
35 end Behavioral;
```

Αριστερά φαίνεται η μνήμη ROM (instruction memory) που υλοποιεί το παραπάνω πρόγραμμα assembly.

Για το compile του προγράμματος χρησιμοποιήθηκε ο flat assembler for ARM version 1.43 και για την κωδικοποίηση σε δεκαεξαδικό σύστημα ο HxD editor.

Για την επαλήθευση της εκτέλεσης των παραπάνω εντολών θα παρατηρήσουμε τις τιμές που αποθηκεύονται στο εσωτερικό σήμα του Register File (RF array) , δηλαδή τις τιμές που αποθηκεύονται οι καταχωρητές του αρχείου καταχωρητών, καθώς και το εσωτερικό σήμα current state της FSM του Control.

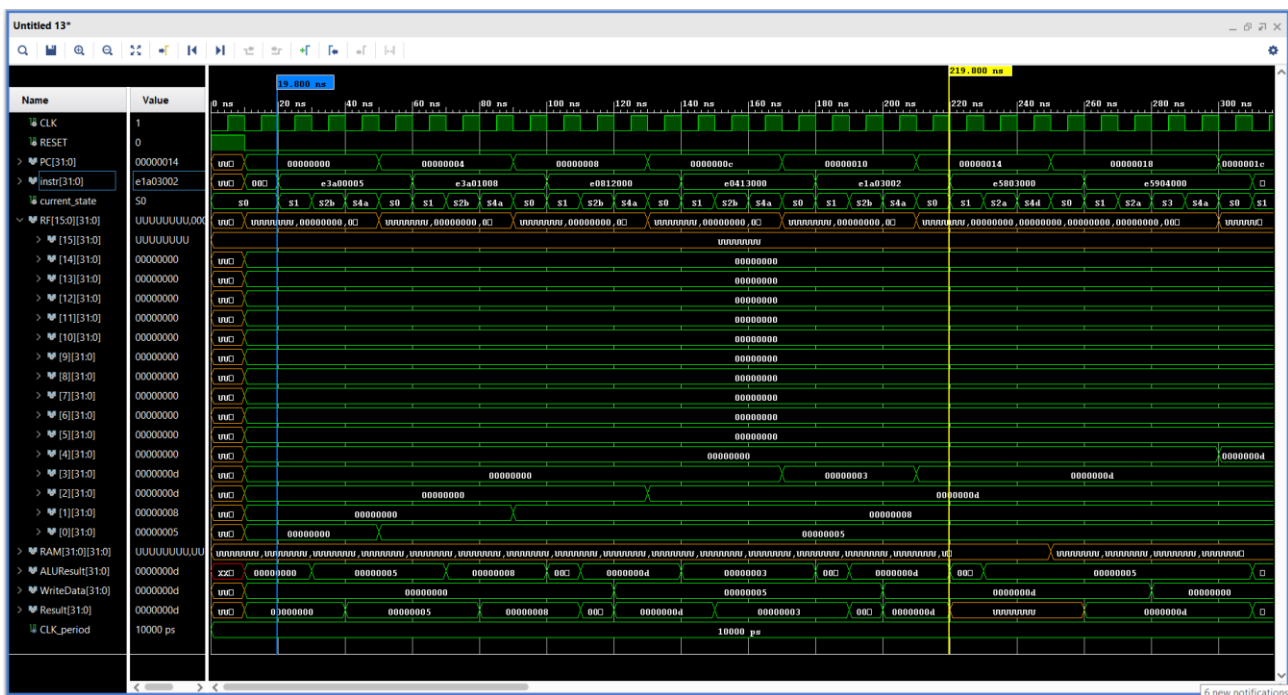
MOV R0, #5                    x"E3A00005"

MOV R1, #8                    x"E3A01008"

ADD R2, R1, R0                x"E0812000"

SUB R3, R1, R0                x"E0413000"

MOV R3, R2                    x"E1A03002"

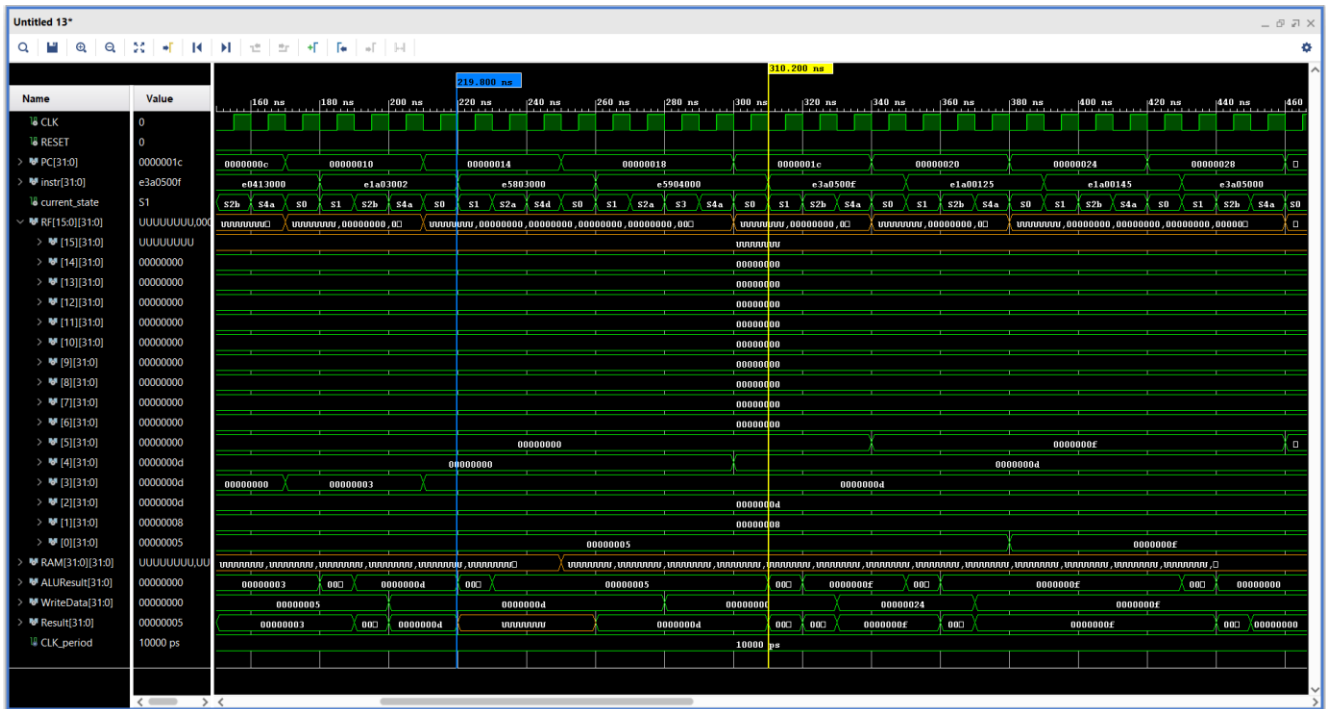


Τα όρια έχουν μπει για να ξεχωρίσουμε τις 5 παραπάνω εντολές επεξεργασίας δεδομένων. Φαίνεται ότι το current state σήμα του FSM παίρνει με τη σειρά τιμές οι οποίες είναι αποδεκτές σύμφωνα με τον πίνακα της ενότητας 4.3.3 του φυλλαδίου. Συγκεκριμένα το state γίνεται με τη σειρά S0 -> S1 -> S2b -> S4a -> S0 σε όλες τις παραπάνω εντολές.

ΤΟ RF array επίσης παίρνει τις σωστές τιμές. Δηλαδή ο register R0 παίρνει την τιμή '5' μόλις ολοκληρώνεται το 3ο clock cycle απ' τη στιγμή που διαβάζεται η εντολή απ' τον Instruction Register του datapath. Η πράξεις ADD / SUBB υλοποιούνται επίσης σωστά καθώς οι registers R2, R3 αποκτούν τις τιμές 13 και 3 αντίστοιχα.

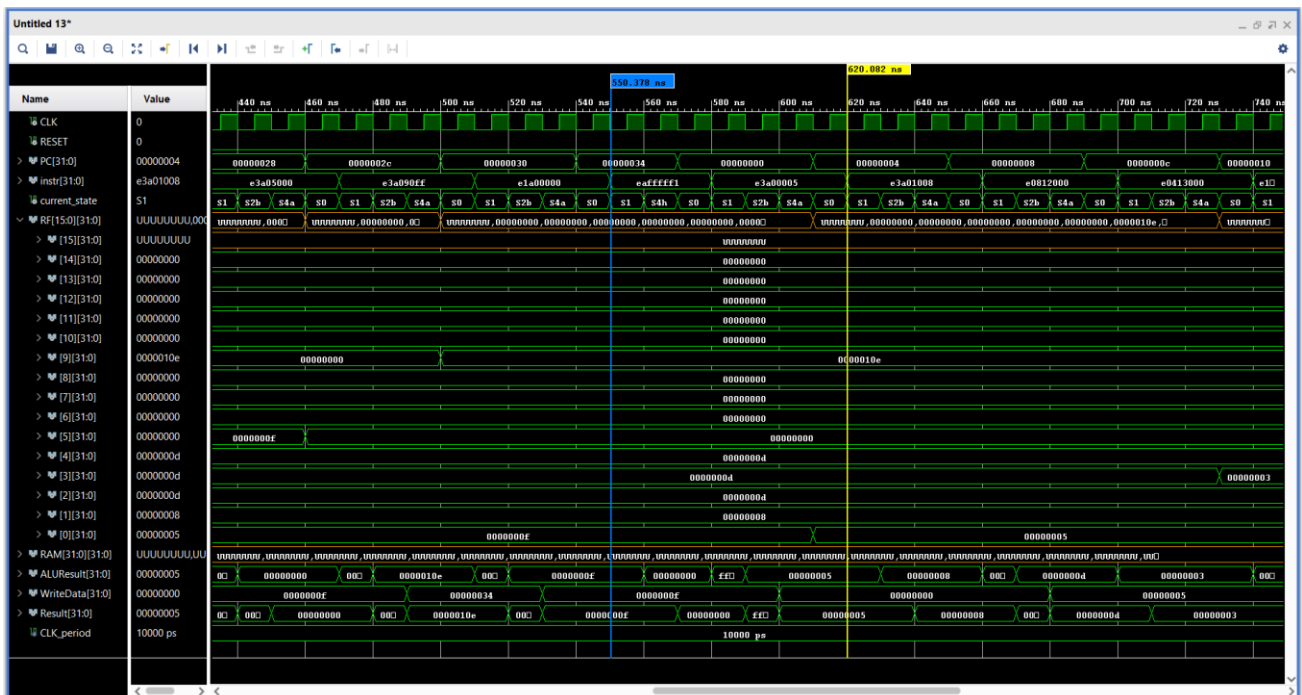
STR R3, [R0]                    x"E5803000"

LDR R4, [R0]                    x"E5904000"



Το διάγραμμα επικεντρώνει στις 2 εντολές μνήμης. Βλέπουμε ότι όταν εκτελείται η εντολή LDR R4, R[0] ο καταχωρητής R4 παίρνει σωστά την τιμή 13, πράγμα που σημαίνει ότι η μνήμη δεδομένων έχει εγγραφεί σωστά στην προηγούμενη εντολή.

Επίσης το current state signal πέρνει διαδοχικά τις τιμές S0->S1->S4d->S0 για την STR και S0 -> S1 -> S2a -> S3 -> S4a -> S0 για την LDR.



Τέλος, παρατηρούμε ότι κατά την εκτέλεση της τελευταίας εντολής (B MAIN\_PROGRAM) επιστρέφουμε ακριβώς στην αρχή του προγράμματος κι οι εντολές ξεκινούν να εκτελούνται πάλι με τη σειρά. Αυτό επιβεβαιώνει ότι η εντολή διακλάδωσης λειτουργεί σωστά κάτι που φαίνεται κι απ' τον μετρητή προγράμματος ο οποίος μηδενίζεται και ξεκινά απ' την αρχή.