ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

PROJECT 1

Μάθημα: Σχεδίαση Ψηφιακών Συστημάτων

Υπεύθυνος καθηγητής: Πασχάλης Αντώνης

2023

1.Περιγραφή στοιχείων δομής του επεξεργαστή

1.1 Σύνολο εντολών

Στην συγκεκριμένη εργασία υλοποιήθηκαν όλες οι ζητούμενες εντολές από την εκφώνηση. Παρακάτω βρίσκεται η λίστα και η περιγραφή των εντολών.

- LDR Rd, [Rn, #imm12]; Τοποθετεί τα περιεχόμενα της θέσης μνήμης Rn+#imm12 στον καταχωρητή Rd.
- LDR Rd, [Rn, #-imm12]; Τοποθετεί τα περιεχόμενα της θέσης μνήμης Rn-#imm12 στον καταχωρητή Rd.
- STR Rd, [Rn, #imm12]; Τοποθετεί τα περιεχόμενα του καταχωρητή Rd στη θέσης μνήμης Rn+#imm12.
- STR Rd, [Rn, #-imm12]; Τοποθετεί τα περιεχόμενα του καταχωρητή Rd στη θέσης μνήμης Rn-#imm12.
- ADD Rd,Rn,#imm8; Εκτελεί την πράξη (+) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd.
- SUB Rd,Rn,#imm8; Εκτελεί την πράξη (-) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd.
- AND Rd,Rn,#imm8; Εκτελεί την πράξη (and) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd.
- ORR Rd,Rn,#imm8; Εκτελεί την πράξη (or) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd.
- EOR Rd,Rn,#imm8; Εκτελεί την πράξη (xor) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd.

- ADDS Rd,Rn,#imm8; Εκτελεί την πράξη (+) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνει τις σημαίες.
- SUBS Rd,Rn,#imm8; Εκτελεί την πράξη (-) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνει τις σημαίες.
- ANDS Rd,Rn,#imm8; Εκτελεί την πράξη (and) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνει τις σημαίες.
- ORRS Rd,Rn,#imm8; Εκτελεί την πράξη (or) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνει τις σημαίες.
- EORS Rd,Rn,#imm8; Εκτελεί την πράξη (xor) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και του #imm8 και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνει τις σημαίες.
- ADD Rd,Rn,Rm; Εκτελεί την πράξη (+) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd.
- SUB Rd,Rn,Rm; Εκτελεί την πράξη (-) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd.
- AND Rd,Rn,Rm; Εκτελεί την πράξη (and) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd.
- ORR Rd,Rn,Rm; Εκτελεί την πράξη (or) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd.
- EOR Rd,Rn,Rm; Εκτελεί την πράξη (xor) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd.
- ADD Rd,Rn,Rm; Εκτελεί την πράξη (+) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνονται οι σημαίες.
- SUB Rd,Rn,Rm; Εκτελεί την πράξη (-) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνονται οι σημαίες.
- AND Rd,Rn,Rm; Εκτελεί την πράξη (and) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνονται οι σημαίες.
- ORR Rd,Rn,Rm; Εκτελεί την πράξη (or) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνονται οι σημαίες.
- EOR Rd,Rn,Rm; Εκτελεί την πράξη (xor) ανάμεσα στα περιεχόμενα του καταχωρητή Rn και των περιεχομένων του καταχωρητή Rm και τα σώζει στον καταχωρητή Rd. Επίσης ενημερώνονται οι σημαίες.
- CMP Rn,#imm8; Ενημερώνει τις σημαίες για την πράξη Rn-#imm8.
- CMP Rn,Rm; Ενημερώνει τις σημαίες για την πράξη Rn-Rm.
- MOV Rd,#imm8; Τοποθετεί στον καταχωρητή Rd το #imm8.
- MOV Rd,Rm; Τοποθετεί στον καταχωρητή Rd το Rm.
- Η NOP είναι ουσιαστικά η MOV R0,R0 και αξίζει να σημειωθεί ότι στον FASMARM δεν μεταφράζεται έτσι η εντολή NOP οπότε ίσως χρειαστεί να γραφεί ως MOV R0,R0 όπως και στον ενδεικτικό παράδειγμα αργότερα.
- MOV Rd,#imm8; Τοποθετεί στον καταχωρητή Rd το αντίθετο του #imm8.

- MOV Rd,Rm; Τοποθετεί στον καταχωρητή Rd το αντίθετο του Rm.
- LSL Rd, Rm, #shamt5; Κάνει αριστερή ολίσθηση κατά #shamt5 θέσεις τον Rm και τον τοποθετεί στον Rm.
- LSR Rd, Rm, #shamt5; Κάνει δεξιά ολίσθηση κατά #shamt5 θέσεις τον Rm και τον τοποθετεί στον Rm.
- ASR Rd, Rm, #shamt5; Κάνει δεξιά αριθμητική ολίσθηση κατά #shamt5 θέσεις τον Rm και τον τοποθετεί στον Rm.
- ROR Rd, Rm, #shamt5; Κάνει περιστροφή κατά #shamt5 θέσεις τον Rm και τον τοποθετεί στον Rm.
- B Label; Μετακινεί τον PC κατα label θέσεις.
- BL Label; Μετακινεί τον PC κατα label θέσεις και τοποθετεί την προηγούμενη θέση στον καταχωρητή 14.

1.2 Λίστα στοιχείων datapath.

- Instruction Memory
- PC Register
- Next Instruction Adder
- Multiplexer Register 1
- Multiplexer Register 2
- Multiplexer Write Register
- Multiplexer Write Data
- Adder for Register 15
- Extend Unit
- Register File
- Multiplexer ALU Source 1
- Multiplexer ALU Source 2
- ALU
- Flags Register
- RAM
- Multiplexer Memory to Register
- Multiplexer PC Source

1.3 Περιγραφή Datapath

Το Datapath έχει παρόμοια δομή με εκείνη που δόθηκε στην εκφώνηση. Υπήρξε όμως ανάγκη για προσθήκη ενός ακόμα πολυπλέκτη. Παρακάτω δίνεται ο κώδικας και το RTL διάγραμμα του ανώτερου επιπέδου:

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

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

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

entity datapath is
generic (
WORDLENGTH : positive := 32; -- word size

ROM_SIZE : positive := 8; -- instr rom size +2
REGISTER_SIZE: positive:=4; --reg size
RAM_SIZE:positive :=7 --ram size+2
);
port(
CLK: in STD_LOGIC;
RESET: in STD_LOGIC;
PC_WRITE: in STD_LOGIC;
IMM_SRC: in STD_LOGIC;
PC_SRC: in STD_LOGIC;
REG_WRITE: in STD_LOGIC;
ALU_SRC_0: in STD_LOGIC;
ALU_SRC: in STD_LOGIC;
MEM_TO_REG: in STD_LOGIC;
MEM_WRITE: in STD_LOGIC;
FLAGS_WRITE: in STD_LOGIC;
REG_SRC: IN std_logic_vector(2 downto 0);
ALU_CONTROL: IN std_logic_vector(3 downto 0);


INSTR: out std_logic_vector(WORDLENGTH-1 downto 0);
PC: out std_logic_vector(WORDLENGTH-1 downto 0);
ALU_RESULT: out std_logic_vector(WORDLENGTH-1 downto 0);
WRITE_DATA: out std_logic_vector(WORDLENGTH-1 downto 0);
RESULT: out std_logic_vector(WORDLENGTH-1 downto 0);
FLAGS: out std_logic_vector(3 downto 0)



);
end datapath;

architecture datapath_beh of datapath is
signal addr_reg_intr_memory: std_logic_vector(WORDLENGTH-1 downto 0);
signal addr_adder_m2to1: std_logic_vector(WORDLENGTH-1 downto 0);
signal addr_m2to1_register: std_logic_vector(WORDLENGTH-1 downto 0);
signal RD1:  std_logic_vector(WORDLENGTH-1 downto 0);
```

```
signal RD2:  std_logic_vector(WORDLENGTH-1 downto 0);
signal ALU_SRC_A: std_logic_vector(WORDLENGTH-1 downto 0);

signal ALU_SRC_B: std_logic_vector(WORDLENGTH-1 downto 0);
signal ALU_RESULT_S: std_logic_vector(WORDLENGTH-1 downto 0);
signal Memory_Result: std_logic_vector(WORDLENGTH-1 downto 0);
signal REXTEND:  std_logic_vector(WORDLENGTH-1 downto 0);
signal s_instr: std_logic_vector(WORDLENGTH-1 downto 0);
signal RESULT_S:  std_logic_vector(WORDLENGTH-1 downto 0);
signal RA1: std_logic_vector(REGISTER_SIZE-1 downto 0);
signal RA2: std_logic_vector(REGISTER_SIZE-1 downto 0);
signal WA: std_logic_vector(REGISTER_SIZE-1 downto 0);
signal RR5: std_logic_vector(WORDLENGTH-1 downto 0);
signal RWD: std_logic_vector(WORDLENGTH-1 downto 0);
signal FLAGS_I:  std_logic_vector(3 downto 0);

component register_s is
generic (
N : positive := 32 -- address length
);
port (
CLK,RESET,WE: in STD_LOGIC;
D: in STD_LOGIC_VECTOR (N-1 downto 0);
Q: out STD_LOGIC_VECTOR (N-1 downto 0));
end component;

component instruction_memory is
generic (
N : positive := 4; -- address length
M : positive := 32); -- data word lengthport (
port(
ADDR: in STD_LOGIC_VECTOR (N-1 downto 0);
DATA_OUT: out STD_LOGIC_VECTOR (M-1 downto 0));
end component;

component adder_const is
generic (
N : positive := 32; -- address length
A : positive := 4 -- address length
);
port(
DATA_IN: in STD_LOGIC_VECTOR (N-1 downto 0);
DATA_OUT: out STD_LOGIC_VECTOR (N-1 downto 0));
end component;

component multiplexer2to1 is
```

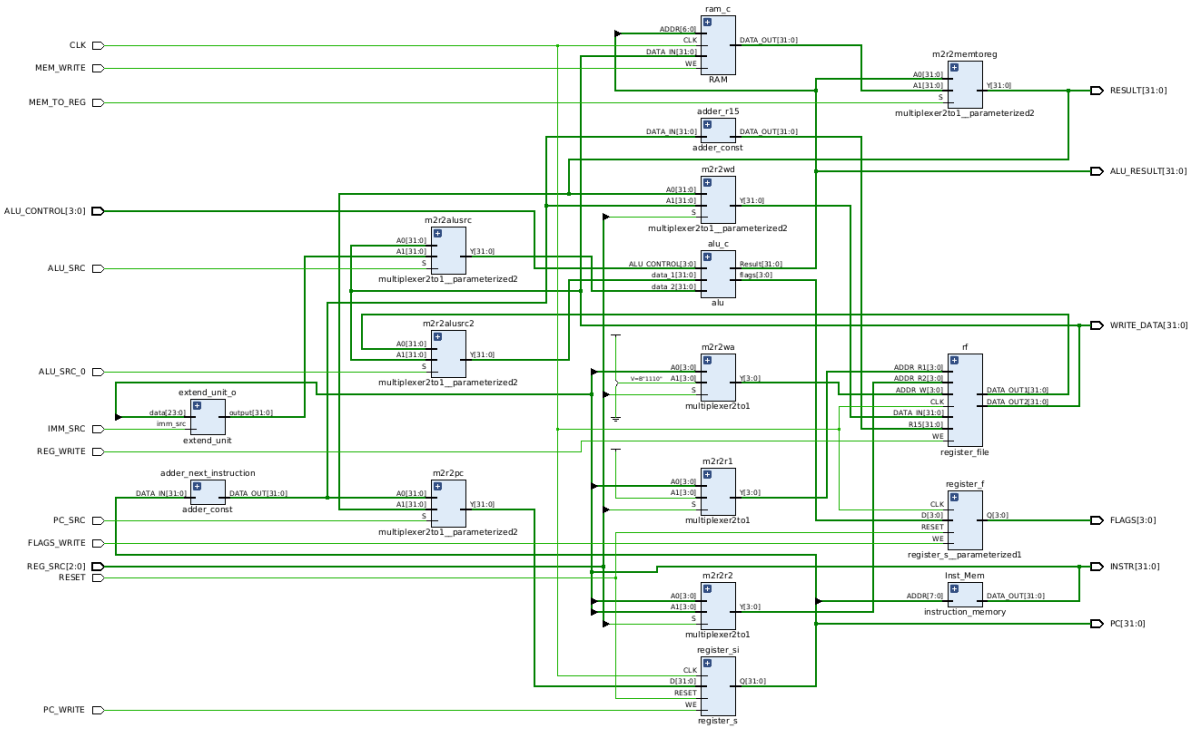
```
generic (WIDTH : positive := 32); -- προεπιλεγμένη τιμή
port (
S: in STD_LOGIC;
A0: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
A1: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0)
);
end component;

component extend_unit is
generic(
N: positive:=32;
S: positive:=24;
U: positive:=12
);
port(
data: in std_logic_vector(S-1 downto 0);
output: out std_logic_vector(N-1 downto 0);
imm_src: in std_logic
);
end component;

component register_file is
generic (
N : positive := 4; -- address length
M : positive := 32); -- data word lengthport (
port(
CLK: in STD_LOGIC;
WE: in STD_LOGIC;
ADDR_W: in STD_LOGIC_VECTOR (N-1 downto 0);
ADDR_R1: in STD_LOGIC_VECTOR (N-1 downto 0);
ADDR_R2: in STD_LOGIC_VECTOR (N-1 downto 0);
DATA_IN: in STD_LOGIC_VECTOR (M-1 downto 0);
DATA_OUT1: out STD_LOGIC_VECTOR (M-1 downto 0);
DATA_OUT2: out STD_LOGIC_VECTOR (M-1 downto 0);
R15: in STD_LOGIC_VECTOR (M-1 downto 0));
end component;

component alu is
generic(
N: integer:=32
);
port(
data_1:in std_logic_vector(N-1 downto 0);
data_2:in std_logic_vector(N-1 downto 0);
ALU_CONTROL:in std_logic_vector(3 downto 0);
```

```
Result: out std_logic_vector(N-1 downto 0);
flags: out std_logic_vector(3 downto 0)
);
end component;

component RAM is
generic (
N : positive := 10; -- address length
M : positive := 32); -- data word length
port (
CLK: in STD_LOGIC;
WE: in STD_LOGIC;
ADDR: in STD_LOGIC_VECTOR (N-1 downto 0);
DATA_IN: in STD_LOGIC_VECTOR (M-1 downto 0);
DATA_OUT: out STD_LOGIC_VECTOR (M-1 downto 0)
);
end component;

begin
Inst_Mem:instruction_memory generic map ( N=>ROM_SIZE,M=>WORDLENGTH)
port map(
ADDR=>addr_reg_intr_memory(ROM_SIZE-1 downto 0),
DATA_OUT=>s_instr
);

register_si:register_s generic map ( N=>WORDLENGTH)
port map(
CLK=>CLK,
RESET=>RESET,
WE=>PC_WRITE,
D=>addr_m2to1_register,
Q=>addr_reg_intr_memory
);

adder_next_instruction:adder_const generic map (N=>WORDLENGTH,A=>4)
port map(
DATA_IN=>addr_reg_intr_memory,
DATA_OUT=>addr_adder_m2to1
);


m2r2r1: multiplexer2to1 generic map (WIDTH=>REGISTER_SIZE)
port map(
S=>REG_SRC(0),
A0=>s_instr(19 downto 16),
A1=>x"f",
```

```
Y=>RA1
);

m2r2r2: multiplexer2to1 generic map (WIDTH=>REGISTER_SIZE)
port map(
S=>REG_SRC(1),
A0=>s_instr(3 downto 0),
A1=>s_instr(15 downto 12),
Y=>RA2
);

m2r2wa: multiplexer2to1 generic map (WIDTH=>REGISTER_SIZE)
port map(
S=>REG_SRC(2),
A0=>s_instr(15 downto 12),
A1=>x"e",
Y=>WA
);

m2r2wd: multiplexer2to1 generic map (WIDTH=>WORDLENGTH)
port map(
S=>REG_SRC(2),
A0=>RESULT_S,
A1=>addr_adder_m2to1,
Y=>RWD
);

m2r2pc: multiplexer2to1 generic map (WIDTH=>WORDLENGTH)
port map(
S=>PC_SRC,
A0=>addr_adder_m2to1,
A1=>RESULT_S,
Y=>addr_m2to1_register
);

adder_r15:adder_const generic map (N=>WORDLENGTH,A=>4)
port map(
DATA_IN=>addr_adder_m2to1,
DATA_OUT=>RR5
);

extend_unit_o:extend_unit generic map(N=>WORDLENGTH,S=>24,U=>12)
port map(
data=>s_instr(23 downto 0),
output=>REXTEND,
imm_src=>IMM_SRC
```

```
);

rf: register_file generic map(N=>REGISTER_SIZE,M =>WORDLENGTH) -- data
word lengthport (
port map(
CLK=>CLK,
WE=>REG_WRITE,
ADDR_W=>WA,
ADDR_R1=>RA1,
ADDR_R2=>RA2,
DATA_IN=>RWD,
DATA_OUT1=>RD1,
DATA_OUT2=>RD2,
R15=>RR5
);

m2r2alusrc: multiplexer2to1 generic map (WIDTH=>WORDLENGTH)
port map(
S=>ALU_SRC,
A0=>RD2,
A1=>REXTEND,
Y=>ALU_SRC_B
);

m2r2alusrc2: multiplexer2to1 generic map (WIDTH=>WORDLENGTH)
port map(
S=>ALU_SRC_0,
A0=>RD1,
A1=>RD2,
Y=>ALU_SRC_A
);

alu_c:alu generic map(N=>WORDLENGTH)
port map(
data_1=>ALU_SRC_A,
data_2=>ALU_SRC_B,
ALU_CONTROL=>ALU_CONTROL,
Result=>ALU_RESULT_S,
flags=>FLAGS_I
);

register_f:register_s generic map ( N=>4)
port map(
CLK=>CLK,
RESET=>RESET,
WE=>FLAGS_WRITE,
```

```
D=>FLAGS_I,
Q=>FLAGS
);

ram_c: RAM generic map(N=>RAM_SIZE,M=>WORDLENGTH)
port map(
CLK=>CLK,
WE=>MEM_WRITE,
ADDR=>ALU_RESULT_S(RAM_SIZE-1 downto 0),
DATA_IN=>RD2,
DATA_OUT=>Memory_Result
);

m2r2memtoreg: multiplexer2to1 generic map (WIDTH=>WORDLENGTH)
port map(
S=>MEM_TO_REG,
A0=>ALU_RESULT_S,
A1=>Memory_Result,
Y=>RESULT_S
);
INSTR<=s_instr;
PC<=addr_reg_intr_memory;
ALU_RESULT<=ALU_RESULT_S;
WRITE_DATA<=RD2;
RESULT<=RESULT_S;
end datapath_beh;
```



Για να περιγράψουμε αυτό το γράφημα χρειάζεται να περιγράψουμε τη χρήση κάθε component.

1.3.1 Περιγραφή οντοτήτων

- Instruction Memory: Σε αυτό το component ουσιαστικά αποθηκεύουμε τις εντολές μας. Δέχεται μια διεύθυνση και μας δίνει μια εντολή.
- PC Register: Αποθηκέυει τη διεύθυνση που είμαστε αυτή τη στιγμή. Παίρνει ως είσοδο το ρολόι το reset καθώς και αν γράφουμε σε αυτό ή όχι (πάντα γράφουμε). Επιπλέον δέχεται την διεύθυνση που θα αποθηκεύσει από έναν πολυπλέκτη (περιγράφεται στη συνέχεια) και δίνει τη διεύθυνση του στη μνήμη εντολών και σε έναν αθροιστή.
- Next Instruction Adder: Αθροίζει την διεύθυνση που παίρνει από τον καταχωρητή διεύθυνσης με το 4 και τοποθετεί το αποτέλεσμα σε έναν πολυπλέκτη για χρήση αν εκτελείται η επόμενη εντολή.
- Multiplexer Register 1: Επιλέγει αν η διεύθυνση του καταχωρητή 1 θα δίνεται από την εντολή [19:16] ή θα είναι σταθερά η 15η θέση. Η επιλογή γίνεται χρησιμοποιώντας το πρώτο bit του RegSrc.
- Multiplexer Register 2:Επιλέγει αν η διεύθυνση του καταχωρητή 2 θα δίνεται από την εντολή στις θέσεις [3:0] ή από τις θέσεις [15:12]. Η επιλογή γίνεται χρησιμοποιώντας το δεύτερο bit του RegSrc.
- Multiplexer Write Register:Επιλέγει αν η διεύθυνση του καταχωρητή εγγραφής θα δίνεται από την εντολή στις θέσεις [15:12] ή από τη σταθερή τιμή 14. Η επιλογή γίνεται χρησιμοποιώντας το τρίτο bit του RegSrc.
- Multiplexer Write Data: Επιλέγουμε ανάμεσα στο αποτέλεσμα που έχουμε βρεί (έχει επιλεχθεί με τη χρήση πολυπλέκτη) και στην τιμή του PC+4. Η επιλογή γίνεται

χρησιμοποιώντας το τρίτο bit του RegSrc. Το αποτέλεσμα είναι η τιμή που θα αποθηκευτεί στον καταχωρητή εγγραφής.

- Adder for Register 15: Προσθέτουμε ακόμα 4 στο αποτέλεσμα του PC + 4 και το βάζουμε ανεξαρτήτως της τιμής RegWrite στον καταχωρητή 15.
- Extend Unit: Κάνουμε επέκταση προσήμου. Όταν το ImmSrc είναι 0 τότε κάνουμε επέκταση μηδενός από τα 12 bit στα 32 bit. Όταν το ImmSrc είναι 1 τότε κάνουμε επέκταση προσήμου από τα 24 bit στα 32 bit.
- Register File: Αρχικά χρησιμοποιεί το ρολόι. Επίσης αποθηκεύει 15 καταχωρητές και πάντα δείχνει τις τιμές των καταχωρητών με διευθύνσεις RA1, RA2 στις 2 εξόδους. Πάντα αποθηκεύει μια τιμή που δίνουμε στον καταχωρητή 15. Όταν το RegWrite είναι ένα τοποθετούμε τα δεδομένα του Data_In στον καταχωρητή με διεύθυνση Addr_W.
- Multiplexer ALU Source 1: Ο συγκεκριμένος πολυπλέκτης δεν βρίσκεται στο διάγραμμα. Ουσιαστικά δίνει την δυνατότητα επιλογής από το πρώτο αποτέλεσμα του καταχωρητή του register file και από το δεύτερο αποτέλεσμα του καταχωρητή του register file. Η επιλογή γίνεται με βάση ενός νέου control σήματος του ALU_SRC_0. Το αποτέλεσμα τοποθετείται στην ALU. Ο λόγος ύπαρξης του καταχωρητή αυτού αναφέρεται στην ALU.
- Multiplexer ALU Source 2: Δίνει τη δυνατότητα επιλογής ανάμεσα στο δεύτερο αποτέλεσμα του register file και στο αποτέλεσμα της μονάδας επέκτασης. Η επιλογή γίνεται μέσω του σήματος εισόδου ALU_SRC_A.
- ALU: Η ALU δέχεται δύο αριθμούς και ανάλογα με την τιμή του ALUControl κάνει μία πράξη. από τις εξής:
    - Πρόσθεση
    - Αφαίρεση
    - Λογικό και
    - Λογικό ή
    - Λογικό xor
    - Μεταφορά της δεύτερης εισόδου στο αποτέλεσμα
    - Μεταφορά της δεύτερης εισόδου στο αποτέλεσμα αντεστραμμένης
    - Λογική ολίσθηση στα αριστερά της πρώτης εισόδου κατά όσο υποδεικνύει η δεύτερη είσοδος (σε αυτό το σημείο είναι αναγκαία η χρήση δυο πολυπλεκτών κα΄θως χρειάζεται και η τιμή του δεύτερου αποτελέσματος του register file και η τιμή του extend).
    - Λογική ολίσθηση στα δεξιά της πρώτης εισόδου κατά όσο υποδεικνύει η δεύτερη είσοδος (σε αυτό το σημείο είναι αναγκαία η χρήση δυο πολυπλεκτών κα΄θως χρειάζεται και η τιμή του δεύτερου αποτελέσματος του register file και η τιμή του extend).
    - Αριθμητική ολίσθηση στα δεξιά της πρώτης εισόδου κατά όσο υποδεικνύει η δεύτερη είσοδος (σε αυτό το σημείο είναι αναγκαία η χρήση δυο πολυπλεκτών κα΄θως χρειάζεται και η τιμή του δεύτερου αποτελέσματος του register file και η τιμή του extend).
    - Περιστροφή της πρώτης εισόδου κατά όσο υποδεικνύει η δεύτερη είσοδος (σε αυτό το σημείο είναι αναγκαία η χρήση δυο πολυπλεκτών κα΄θως χρειάζεται και η τιμή του δεύτερου αποτελέσματος του register file και η τιμή του extend).
- Flags Register: Αποθηκεύει τις σημαίες. Χρησιμοποιεί επίσης τα σήματα CLK και reset.
- RAM: Διαβάζει στη μνήμη από μια διεύθυνση (με τη χρήση του ρολογιού). Όταν το MemWrite είναι 1 τότε γράφει τη μνήμη την τιμή writedata τη διεύθυνση που δίνεται

- Multiplexer Memory to Register: Επιλέγει αν το αποτέλεσμά μας θα προκύπτει από την μνήμη ή από το αποτέλεσμα της ALU.
- Multiplexer PC Source: Επιλέγει αν θα χρησιμοποιηθεί η επόμενη διεύθυνση ή το αποτέλεσμα ώς διεύθυνση επόμενης εντολής.

1.4 Υπομονάδες Μονάδας Ελέγχου

Instruction Decode

Φροντίζει για την πλειοψηφία των σημάτων ελέγχου και επίσης για το εσωτερικό σήμα NoWrite.

OP	FUNCT	SH	RegSrc	ALUSrc0	ALUSrc	ImmSrc	ALU Control	Mem To Reg	No Write
01	010001	X	0X0	X	1	0	0001	1	0
01	011001	X	0X0	X	1	0	0000	1	0
01	010000	X	010	X	1	0	0001	0	0
01	011000	X	010	X	1	0	0000	0	0
00	10100X	X	0X0	X	1	0	0000	0	0
00	10010X	X	0X0	X	1	0	0001	0	0
00	10000X	X	0X0	X	1	0	0010	0	0
00	11100X	X	0X0	X	1	0	0011	0	0
00	10001X	X	0X0	X	1	0	0100	0	0
00	00100X	X	000	X	0	X	0000	0	0
00	00010X	X	000	X	0	X	0001	0	0
00	00000X	X	000	X	0	X	0010	0	0
00	01100X	X	000	X	0	X	0011	0	0
00	00001X	X	000	X	0	X	0100	0	0
00	110101	X	0X0	X	1	0	0001	0	1
00	010101	X	000	X	0	X	0001	0	1
00	111010	00	0X0	X	1	0	0101	0	0
00	111110	X	0X0	X	1	0	0110	0	0
00	011010	00	000	1	1	0	0111	0	0
00	011110	X	000	X	0	X	0110	0	0
00	011010	00	0X0	1	1	0	0111	0	0

00	011010	01	0X0	1	1	0	1000	0	0
00	011010	10	0X0	1	1	0	1001	0	0
00	011010	11	0X0	1	1	0	1010	0	0
10	10XXX X	X	1X1	X	1	1	0000	0	0
10	11XXX X	X	1X1	X	1	1	0000	0	0

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

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

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

entity InstrDec is
port(
op:in std_logic_vector(1 downto 0);
sh:in std_logic_vector(1 downto 0);
funct: in std_logic_vector(5 downto 0);

RegSrc: out std_logic_vector(2 downto 0);
ALUControl: out std_logic_vector(3 downto 0);
ALUSrc_0:out std_logic;
ALUSrc:out std_logic;
MemToReg:out std_logic;
ImmSrc:out std_logic;
NoWrite:out std_logic
);
end InstrDec;

architecture InstrDec_beh of InstrDec is

begin

process(op,funct,sh)
begin
if(op="10")then
    ALUSrc_0<='0';
```

```
    RegSrc(0)<='1';
    RegSrc(2)<='1';
    ALUSrc<='1';
    ImmSRc<='1';
    ALUControl<="0000";
    MemToReg<='0';
    NoWrite<='0';
elsif(op="01") then
ALUSrc_0<='0';
    ALUSrc<='1';
    ImmSRC<='0';
    NoWrite<='0';
    if(funct(0)='1')then
        MemToReg<='1';
        RegSrc<="000";
    else
        MemToReg<='0';
        RegSrc<="010";
    end if;
    if(funct(3)='1')then
        ALUControl<="0000";
    else
         ALUControl<="0001";
    end if;
elsif(op="00") then
    RegSrc<="000";
    ImmSRC<='0';
    MemToReg<='0';
    if(funct="011010")then
        ALUSrc<='1';
        ALUSrc_0<='1';
        if(sh="00")then
            ALUControl<="0111";
        elsif(sh="01")then
        ALUControl<="1000";
        elsif(sh="10")then
        ALUControl<="1001";
        elsif(sh="11")then
        ALUControl<="1010";
        end if;
    else
        ALUSrc<=funct(5);
        ALUSrc_0<='0';
        if(funct(4 downto 1)="0100")then
        NoWrite<='0';
            ALUControl<="0000";
```

```
        elsif(funct(4 downto 1)="0010")then
        NoWrite<='0';
        ALUControl<="0001";
        elsif(funct(4 downto 1)="0000")then
        NoWrite<='0';
        ALUControl<="0010";
        elsif(funct(4 downto 1)="1100")then
        NoWrite<='0';
        ALUControl<="0011";
        elsif(funct(4 downto 1)="0001")then
        NoWrite<='0';
        ALUControl<="0100";
        elsif(funct(4 downto 1)="1010")then
        NoWrite<='0';
        NoWrite<='1';
        ALUControl<="0001";
        elsif(funct(4 downto 1)="1101")then
        NoWrite<='0';
        ALUControl<="0101";
        elsif(funct(4 downto 1)="1111")then
        NoWrite<='0';
        ALUControl<="0110";
        end if;
    end if;

end if;
end process;


end InstrDec_beh;
```

Write Enable Logic

Φροντίζει για όλα τα σήματα ελέγχου για οποιαδήποτε εγγραφή.

OP	S/L	No Write	Reg Write	Mem Write	Flags Write
01	1	0	1	0	0
01	0	0	0	1	1
00	0	0	1	0	0
00	1	1	0	0	1
00	1	0	1	0	1

10	X	X	0	0	0

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

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

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

entity WELogic is
port(
op:in std_logic_vector(1 downto 0);
S:in std_logic;
NoWrite:in std_logic;
MemWrite:out std_logic;
FlagsWrite:out std_logic;
RegWrite:out std_logic
);
end WELogic;

architecture WELogic_beh of WELogic is

begin
process(op,s,NoWrite)
begin
if(op="10")then
    RegWrite<='0';
    MemWrite<='0';
    FlagsWrite<='0';
    elsif(op="01")then
        RegWrite<=S;
        MemWrite<=not S;
        FlagsWrite<='0';
    elsif(op="00")then
    if(NoWrite='1')then
        RegWrite<='0';
        MemWrite<='0';
        FlagsWrite<='1';
    else
        RegWrite<='1';
```

```
        MemWrite<='0';
        FlagsWrite<=S;
    end if;
    end if;
end process;


end WELogic_beh;
```

PC Logic

Φροντίζει για την πηγή της επόμενης πηγής του Program Counter

OP	Rd	Register Write	PC Source
0	0-14	1	0
0	15	1	1
0	0-14	1	0
0	15	1	1
0	X	0	0
0	X	0	0
1	X	0	1

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

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

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

entity PCLogic is
port(
Rd:in std_logic_vector(3 downto 0);
```

```
op:in std_logic;
RegWrite:in std_logic;
PCSrc:out std_logic
);
end PCLogic;

architecture PCLogic_beh of PCLogic is

begin
process(op,Rd,RegWrite)
begin
if(op='1')then
PCSrc<='1';
else
if(RegWrite='0')then
PCSrc<='0';
else
if(unsigned(Rd)=15)then
PCSrc<='1';
else
PCSrc<='0';
end if;
end if;
end if;
end process;


end PCLogic_beh;
```

Conditional Logic

Φροντίζει για την conditional λογική και την εκτέλεση της εντολής βασισμένο σε αυτές. Ο πίνακας αληθείας βασίζεται στον πίνακα της εκφώνησης.

$cond_{3:0}$	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	$\bar{V}$

$cond_{3:0}$	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	$\overline{N \oplus V}$
1011	LT	Signed less	$N \oplus V$
1100	GT	Signed greater	$\bar{Z}\,\overline{N \oplus V}$
1101	LE	Signed less or equal	$Z+(N \oplus V)$
1110	AL (ή none)	Always / unconditional	1
1111	none	For unconditional instructions	1

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

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

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

entity CONDLogic is
port(
cond:in std_logic_vector(3 downto 0);
flags:in std_logic_vector(3 downto 0);

CondEx:out std_logic

);
end CONDLogic;

architecture CONDLogic_beh of CONDLogic is

begin
process(cond,flags)
begin
if(cond="0000")then
    CondEx<=flags(2);
elsif(cond="0001")then
    CondEx<=not flags(2);
elsif(cond="0010")then
    CondEx<= flags(1);
elsif(cond="0011")then
    CondEx<=not flags(1);
elsif(cond="0100")then
```

```
    CondEx<= flags(3);
elsif(cond="0101")then
    CondEx<=not flags(3);
elsif(cond="0110")then
    CondEx<=flags(0);
elsif(cond="0111")then
    CondEx<=not flags(0);
elsif(cond="1000")then
    CondEx<=(not flags(2)) and flags(1);
elsif(cond="1001")then
    CondEx<=flags(2) or (not flags(1));
elsif(cond="1010")then
    CondEx<=not (flags(3) xor (flags(0)));
elsif(cond="1011")then
    CondEx<=flags(3) xor (flags(0));
elsif(cond="1100")then
    CondEx<=not flags(2) and (not (flags(3) xor (flags(0))));
elsif(cond="1101")then
    CondEx<=flags(2) or (flags(3) xor flags(0));
elsif(cond="1110")then
    CondEx<='1';
elsif(cond="1111")then
    CondEx<='1';
end if;
end process;

end CONDLogic_beh;
```

1.5 Ανώτερο Ιεραρχικό Επίπεδο Μονάδας Ελέγχου

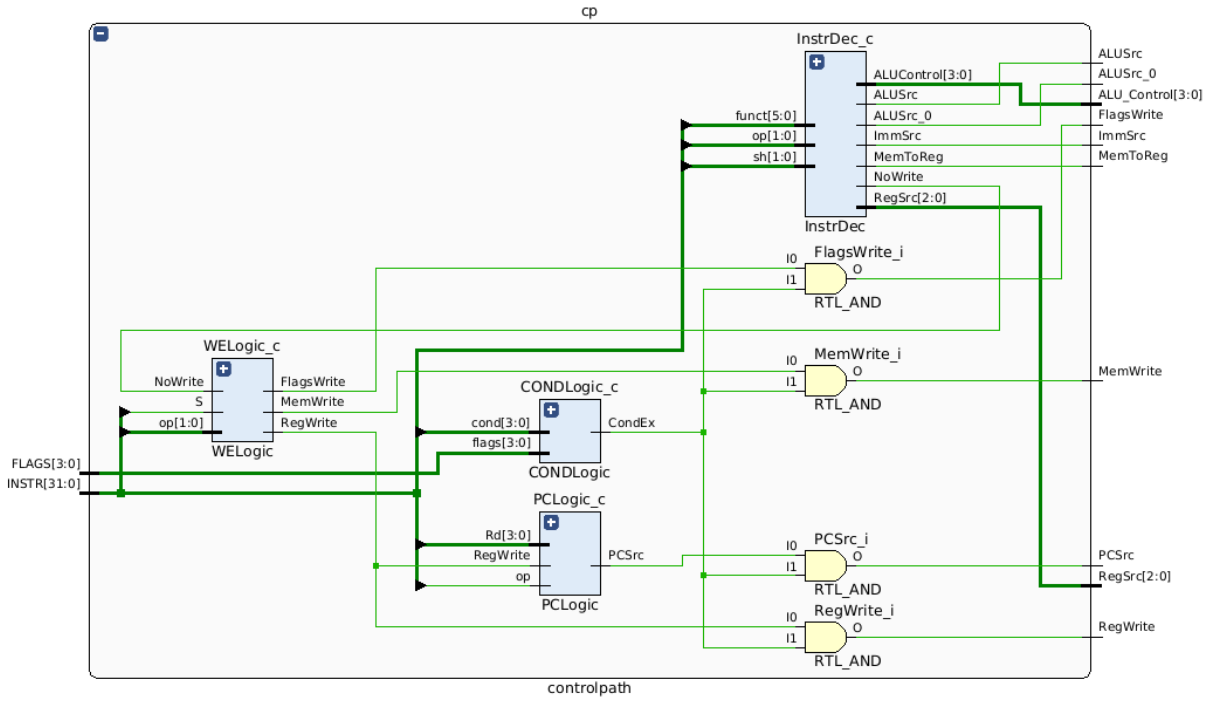
Παρατηρούμε ότι οι έξοδοι της μονάδας Instruction Decode πηγαίνουν κατευθείαν στην έξοδο. Οι έξοδοι από τις μονάδες WE Logic και PC Logic μπαίνουν σε πύλες AND με το αποτέλεσμα της μονάδας COND Logic. Αυτό συμβαίνει γιατί αν μια εντολή κριθεί ότι δεν εκτελείται λόγω conditional τότε αυτές οι συγκεκριμένες έξοδοι πρέπει να είναι 0.

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

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

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
```

```
--library UNISIM;
--use UNISIM.VComponents.all;

entity controlpath is
generic(
    WORDLENGTH : positive := 32 -- word size

);
port(
    INSTR:in std_logic_vector(WORDLENGTH-1 downto 0);
    FLAGS:in std_logic_vector(3 downto 0);

    RegSrc:out std_logic_vector(2 downto 0);
    ALU_Control:out std_logic_vector(3 downto 0);
    ALUSrc_0:out std_logic;
    ALUSrc:out std_logic;
    MemToReg:out std_logic;
    ImmSrc:out std_logic;
    MemWrite:out std_logic;
    FlagsWrite:out std_logic;
    RegWrite:out std_logic;
    PCSrc:out std_logic
);
end controlpath;

architecture controlpath_beh of controlpath is
signal NoWrite_s: std_logic;
signal MemWrite_s: std_logic;
signal FlagsWrite_s: std_logic;
signal RegWrite_s: std_logic;
signal PCSrc_s: std_logic;
signal CondEx_s: std_logic;

component InstrDec is
port(
op:in std_logic_vector(1 downto 0);
sh:in std_logic_vector(1 downto 0);
funct: in std_logic_vector(5 downto 0);

RegSrc: out std_logic_vector(2 downto 0);
ALUControl: out std_logic_vector(3 downto 0);
ALUSrc_0:out std_logic;
ALUSrc:out std_logic;
MemToReg:out std_logic;
ImmSrc:out std_logic;
NoWrite:out std_logic
```

```
);
end component;

component WELogic is
port(
op:in std_logic_vector(1 downto 0);
S:in std_logic;
NoWrite:in std_logic;
MemWrite:out std_logic;
FlagsWrite:out std_logic;
RegWrite:out std_logic
);
end component;

component PCLogic is
port(
Rd:in std_logic_vector(3 downto 0);
op:in std_logic;
RegWrite:in std_logic;
PCSrc:out std_logic
);
end component;

component CONDLogic is
port(
cond:in std_logic_vector(3 downto 0);
flags:in std_logic_vector(3 downto 0);

CondEx:out std_logic

);
end component;
begin

InstrDec_c:InstrDec port map(
op=>INSTR(27 downto 26),
sh=>INSTR(6 downto 5),
funct=>INSTR(25 downto 20),

RegSrc=>RegSrc,
ALUControl=>ALU_Control,
ALUSrc_0=>ALUSrc_0,
ALUSrc=>ALUSrc,
MemToReg=>MemToReg,
ImmSrc=>ImmSrc,
NoWrite=>NoWrite_s
```

```
);

WELogic_c:WELogic port map(
op=>INSTR(27 downto 26),
S=>INSTR(20),
NoWrite=>NoWrite_s,
MemWrite=>MemWrite_s,
FlagsWrite=>FlagsWrite_s,
RegWrite=>RegWrite_s
);

PCLogic_c:PCLogic port map(
Rd=>INSTR(15 downto 12),
op=>INSTR(27),
RegWrite=>RegWrite_s,
PCSrc=>PCSrc_s
);

CONDLogic_c:CONDLogic port map(
cond=>INSTR(31 downto 28),
flags=>FLAGS,

CondEx=>CondEx_s
);

MemWrite<=MemWrite_s and CondEx_s;
FlagsWrite<=FlagsWrite_s and CondEx_s;
RegWrite<=RegWrite_s and CondEx_s;
PCSrc<=PCSrc_s and CondEx_s;


end controlpath_beh;
```



1.6 Ανώτερο Ιεραρχικό Επίπεδο Επεξεργαστή

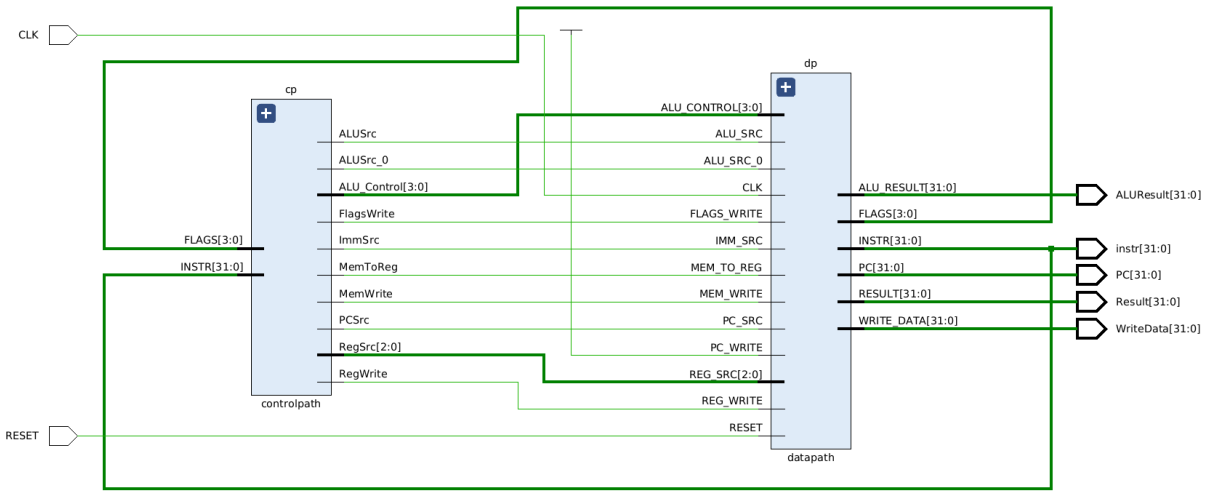
Στον επεξεργαστή ουσιαστικά ενώνουμε το controlpath με το datapath. Οι μόνες είσοδοι είναι το ρολόι και το RESET και οι μόνες έξοδοι είναι κυρίως για να μπορούμε να ελέγξουμε την ορθότητα της υλοποίησης (φαίνονται στον κώδικα).

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

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

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

entity processor is
generic(
WORDLENGTH : positive := 32; -- word size

ROM_SIZE : positive := 8; -- instr rom size +2
REGISTER_SIZE: positive:=4; --reg size
RAM_SIZE:positive :=7 --ram size+2
);
```

```
port(
CLK:in std_logic;
RESET:in std_logic;

PC: out std_logic_vector(WORDLENGTH-1 downto 0);
instr: out std_logic_vector(WORDLENGTH-1 downto 0);
ALUResult: out std_logic_vector(WORDLENGTH-1 downto 0);
WriteData: out std_logic_vector(WORDLENGTH-1 downto 0);
Result: out std_logic_vector(WORDLENGTH-1 downto 0)

);
end processor;

architecture processor_beh of processor is
signal INSTR_S: std_logic_vector(WORDLENGTH-1 downto 0);
signal FLAGS_S: std_logic_vector(3 downto 0);

signal RegSrc_S: std_logic_vector(2 downto 0);
signal ALU_Control_S: std_logic_vector(3 downto 0);
signal ALUSrc_0_S: std_logic;
signal ALUSrc_S: std_logic;
signal MemToReg_S: std_logic;
signal ImmSrc_S: std_logic;
signal MemWrite_S: std_logic;
signal FlagsWrite_S: std_logic;
signal RegWrite_S: std_logic;
signal PCSrc_S: std_logic;
component controlpath is
generic(
    WORDLENGTH : positive := 32 -- word size

);
port(
    INSTR:in std_logic_vector(WORDLENGTH-1 downto 0);
    FLAGS:in std_logic_vector(3 downto 0);

    RegSrc:out std_logic_vector(2 downto 0);
    ALU_Control:out std_logic_vector(3 downto 0);
    ALUSrc_0:out std_logic;
    ALUSrc:out std_logic;
    MemToReg:out std_logic;
    ImmSrc:out std_logic;
    MemWrite:out std_logic;
    FlagsWrite:out std_logic;
    RegWrite:out std_logic;
    PCSrc:out std_logic
```

```
);
end component;

component datapath is
generic (
WORDLENGTH : positive := 32; -- word size

ROM_SIZE : positive := 8; -- instr rom size +2
REGISTER_SIZE: positive:=4; --reg size
RAM_SIZE:positive :=7 --ram size+2
);
port(
CLK: in STD_LOGIC;
RESET: in STD_LOGIC;
PC_WRITE: in STD_LOGIC;
IMM_SRC: in STD_LOGIC;
PC_SRC: in STD_LOGIC;
REG_WRITE: in STD_LOGIC;
ALU_SRC_0: in STD_LOGIC;
ALU_SRC: in STD_LOGIC;
MEM_TO_REG: in STD_LOGIC;
MEM_WRITE: in STD_LOGIC;
FLAGS_WRITE: in STD_LOGIC;
REG_SRC: IN std_logic_vector(2 downto 0);
ALU_CONTROL: IN std_logic_vector(3 downto 0);

INSTR: out std_logic_vector(WORDLENGTH-1 downto 0);
PC: out std_logic_vector(WORDLENGTH-1 downto 0);
ALU_RESULT: out std_logic_vector(WORDLENGTH-1 downto 0);
WRITE_DATA: out std_logic_vector(WORDLENGTH-1 downto 0);
RESULT: out std_logic_vector(WORDLENGTH-1 downto 0);
FLAGS: out std_logic_vector(3 downto 0)
);
end component;
begin

dp:datapath generic
map(WORDLENGTH=>WORDLENGTH,ROM_SIZE=>ROM_SIZE,REGISTER_SIZE=>REGISTER_SI
ZE,RAM_SIZE=>RAM_SIZE)
port map(
CLK=>CLK,
RESET=>RESET,
PC_WRITE=>'1',
IMM_SRC=>ImmSrc_S,
PC_SRC=>PCSrc_S,
REG_WRITE=>RegWrite_S,
```

```
ALU_SRC_0=>ALUSrc_0_S,
ALU_SRC=>ALUSrc_S,
MEM_TO_REG=>MemToReg_S,
MEM_WRITE=>MemWrite_S,
FLAGS_WRITE=>FlagsWrite_S,
REG_SRC=>RegSrc_S,
ALU_CONTROL=>ALU_Control_S,

INSTR=>INSTR_S,
PC=>PC,
ALU_RESULT=>ALUResult,
WRITE_DATA=>WriteData,
RESULT=>Result,
FLAGS=>FLAGS_S
);
cp:controlpath generic map(WORDLENGTH=>WORDLENGTH)
port map(
    INSTR=>INSTR_S,
    FLAGS=>FLAGS_S,

    RegSrc=>RegSrc_S,
    ALU_Control=>ALU_Control_S,
    ALUSrc_0=>ALUSrc_0_S,
    ALUSrc=>ALUSrc_S,
    MemToReg=>MemToReg_S,
    ImmSrc=>ImmSrc_S,
    MemWrite=>MemWrite_S,
    FlagsWrite=>FlagsWrite_S,
    RegWrite=>RegWrite_S,
    PCSrc=>PCSrc_S
);
instr<=instr_s;
end processor_beh;
```



1.6 Ανώτερο Ιεραρχικό Επίπεδο Επεξεργαστή

Παρατηρούμε ότι η μέγιστη συχνότητα λειτουργίας είναι τα 75.18 MHz ή 13.300 ns περίοδος ρολογιού.
Αυτό ορίζεται από το μεγαλύτερο μονοπάτι το οποίο βρίσκεται ανάμεσα στον καταχωρητή διευθύνσεων και στη RAM. Καθώς η RAM κάνει εγγραφή στην πτώση του ρολογιού ο χρόνος του ρολογιού πρέπει να είναι επαρκής για να φτάσουν τα δεδομένα και να γίνει η εγγραφή σωστα. Παρακάτω φαίνεται το μονοπάτι με το λιγότερο setup slack.

Summary

Name	Path 1
Slack	0.049ns
Source	dp/register_si/Q_reg[4]/C (rising edge-triggered cell FDRE clocked by CLK {rise@0.000ns fall@6.650ns period=13.300ns})
Destination	dp/ram_c/RAM_reg_0_31_28_28/SP/I (falling edge-triggered cell RAMS32 clocked by CLK {rise@0.000ns fall@6.650ns period=13.300ns})
Path Group	CLK
Path Type	Setup (Max at Slow Process Corner)
Requirement	6.650ns (CLK fall@6.650ns - CLK rise@0.000ns)
Data Path Delay	6.218ns (logic 3.431ns (55.179%) route 2.787ns (44.821%))
Logic Levels	9 (CARRY4=8 LUT3=1)
Clock Path Skew	-0.145ns
Clock Un...rtainty	0.035ns

Η χειρότερη σύντομη διαδρομή είναι προφανώς από τον καταχωρητή στον καταχωρητη διευθύνσεων με το μικρότερο old slack. Παρακάτω η περίληψη του μονοπατιού.

Summary

Name	Path 11
Slack (Hold)	0.206ns
Source	dp/register_si/Q_reg[11]/C (rising edge-triggered cell FDRE clocked by CLK {rise@0.000ns fall@6.650ns period=13.300ns})
Destination	dp/register_si/Q_reg[11]/D (rising edge-triggered cell FDRE clocked by CLK {rise@0.000ns fall@6.650ns period=13.300ns})
Path Group	CLK
Path Type	Hold (Min at Fast Process Corner)
Requirement	0.000ns (CLK rise@0.000ns - CLK rise@0.000ns)
Data P...Delay	0.464ns (logic 0.257ns (55.342%) route 0.207ns (44.658%))
Logic Levels	2 (CARRY4=1 LUT5=1)
Clock ... Skew	0.145ns

2.Επαλήθευση της σωστής σχεδίασης και λειτουργίας του επεξεργαστή.

2.1 Πρόγραμμα Προσομοίωσης

Παρακάτω παρουσιάζεται το πρόγραμμα προσομοίωσης.

```
MOV R4,#0x8;
MAIN:MOV R0, #0x5;
MOV R1, R0;
MOV R0,R0;
MVN R2,#0x5;
MVN R3,R2;
MAIN1:ADD R1,R0,#0x5;
SUB R1,R0,#0x5;
AND R1,R2,#0x1;
ORR R1,R2,#0x8;
EOR R1,R2,#0x10;
MOV R0,0x1;
MOV R1,0x2;
ADD R2,R0,R1;
SUB R2,R0,R1;
AND R2,R0,R1;
ORR R2,R0,R1;
EOR R2,R0,R1;
MAIN2:CMP R0,#0x1;
CMP R0,R1;
MAIN3:MVN R1,0x1;
MOV R0,0x1;
LSL R2,R0,#5
LSR R2,R1,#5
ASR R2,R1,#5
ROR R2,R0,#5
MAIN4:MOV R0,#0x89;
MOV R1,#0x0;
STR R0,[R1,0x1];
ADD R1,R1,0x3;
LDR R2,[R1,0x-1];
MAIN5:CMP R4,#0x9;
IT EQ;
BEQ MAIN;
MOV R4,#0x9;
BL MAIN;
```

Παρατηρούμε ότι υπάρχουν 6 κομμάτια στον κώδικα που φαίνονται και από τα main labels. Θα εξηγήσουμε τη χρησιμότητα του κάθε κομματιού.

- MAIN: Έλεγχος εντολών MOV και MVN. Έλεγχο κάποιον σημαιών και σωστής υλοποίησης του register file.
- MAIN1: Έλεγχος αριθμητικών εντολών στην αρχή με τη χρήση immediate και μετά με τη χρήση δεύτερου καταχωρητή. Είναι ένα σημαντικό τεστ για την ALU.
- MAIN2: Έλεγχος των CMP εντολών (μόνο των σημάτων τα flags θα ελεγχθούν αργότερα)
- MAIN3: Έλεγχος εντολών ολίσθησης και περιστροφής (έλεγχος των ειδικών ALU operations και του επιπλέον από το σχήμα πολυπλέκτη)
- MAIN4: Έλεγχος του STR και LDR με θετικό και αρνητικό immediate.
- MAIN5: Έλεγχος των σημαιών με ένα μικρό κομμάτι κώδικα το οποίο ξαναεκτελέι το πρόγραμμα την πρώτη φορά ενώ τη δεύτερη χρησιμοποιείται ένα Branch λίγο πιο νωρίς

Αξίζει να σημειωθεί ότι το τελικό πρόγραμμα δεν είναι η μόνη επαλήθευση του κώδικα. Αποφάσισα να δημιουργήσω μια πιο σταδιακή προσομοίωση. Στα αρχεία που θα σας παραδώσω περιλαμβάνονται και αρχεία προσομοίωσης κάποιων επιμέρους component. Αν βεβαιωθούμε ότι κάποιες λειτουργίες είναι σωστές ή σωστά γενικοποιημένες, η πλήρης επαλήθευση του κώδικα σε μεγαλύτερο επίπεδο δεν είναι αναγκαία. Σας προτείνω να δείτε και τις επιμέρους προσομοιώσεις.

Παρακάτω είναι δύο προγράμματα που χρησιμοποιήθηκαν σε επιμέρους προσομοιώσεις (το δεύτερο είναι εκείνο που δοκιμάστηκε το datapath ξεχωριστά)

```
MAIN:
MOV R1, #0x1;
MOV R2, #0x2;
MOV R3, #0x3;
MOV R4, #0x4;
ADD R3,R1,R2;
SUB R4,R1,R3;
AND R5,R1,R4;
ORR R6,R2,R3;
EOR R7,R2,R4;
ADD R8,R1,#0x2;
SUB R9,R1,#0x3;
AND R10,R1,#0x4;
ORR R11,R2,#0x3;
EOR R12,R2,#0x4;
```

```
MOV R1, #0x5;
MAIN:
ADD R2,R1,#0x1;
ADD R2,R2,R1;
STR R2,[R1,#0x0];
BL MAIN;
```

```
NOP;
```

2.2 Κώδικες Προσομοίωσης

Αξίζει να σημειωθεί ότι δημιουργήθηκε ένα επιπλέον c++ αρχείο για τη μετατροπή από binary στο format του instruction memory όπως είναι στην vhdl. Η χρήση του αποδείχθηκε αναγκαία και ίσως βοηθήσει και εσάς. Είναι το test.cc αρχείο που περιλαμβάνεται στον κώδικα.

Μεταγλωττίζεται με
g++ -o test test.cc

Εκτελείται με:
./test <διαδρομή .bin αρχείου>

Κώδικας Instruction Memory:

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

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

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

entity instruction_memory is
generic (
N : positive := 4; -- address length
M : positive := 32); -- data word lengthport (
port(
ADDR: in STD_LOGIC_VECTOR (N-1 downto 0);
DATA_OUT: out STD_LOGIC_VECTOR (M-1 downto 0));
end instruction_memory;

architecture instruction_memory_beh of instruction_memory is

type InstructionMemory is array (0 to (2**(N-2))-1)
of STD_LOGIC_VECTOR (M-1 downto 0);
constant ROM : InstructionMemory :=(X"E3A04008",
X"E3A00005",
```

```
X"E1A01000",
X"E1A00000",
X"E3E02005",
X"E1E03002",
X"E2801005",
X"E2401005",
X"E2021001",
X"E3821008",
X"E2221010",
X"E3A00001",
X"E3A01002",
X"E0802001",
X"E0402001",
X"E0002001",
X"E1802001",
X"E0202001",
X"E3500001",
X"E1500001",
X"E3E01001",
X"E3A00001",
X"E1A02280",
X"E1A022A1",
X"E1A022C1",
X"E1A022E0",
X"E3A00089",
X"E3A01000",
X"E5810001",
X"E2811003",
X"E5112001",
X"E3540009",
X"0AFFFFDF",
X"E3A04009",
X"EBFFFFDD",
others=>x"E1A00000");
begin
DATA_OUT <= ROM(to_integer(unsigned(ADDR)/4));


end instruction_memory_beh;
```

Κώδικας Προσομοίωσης:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

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

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

entity proccessor_sim is
generic(
WORDLENGTH : positive := 32; -- word size

ROM_SIZE : positive := 8; -- instr rom size +2
REGISTER_SIZE: positive:=4; --reg size
RAM_SIZE:positive :=7 --ram size+2
);
end proccessor_sim;

architecture proccessor_sim_beh of proccessor_sim is
signal CLK: std_logic;
signal RESET: std_logic;

signal PC:  std_logic_vector(WORDLENGTH-1 downto 0);
signal instr:  std_logic_vector(WORDLENGTH-1 downto 0);
signal ALUResult:  std_logic_vector(WORDLENGTH-1 downto 0);
signal WriteData:  std_logic_vector(WORDLENGTH-1 downto 0);
signal Result:  std_logic_vector(WORDLENGTH-1 downto 0);

component processor is
generic(
WORDLENGTH : positive := 32; -- word size

ROM_SIZE : positive := 8; -- instr rom size +2
REGISTER_SIZE: positive:=4; --reg size
RAM_SIZE:positive :=7 --ram size+2
);
port(
CLK:in std_logic;
RESET:in std_logic;

PC: out std_logic_vector(WORDLENGTH-1 downto 0);
instr: out std_logic_vector(WORDLENGTH-1 downto 0);
ALUResult: out std_logic_vector(WORDLENGTH-1 downto 0);
```

```
WriteData: out std_logic_vector(WORDLENGTH-1 downto 0);
Result: out std_logic_vector(WORDLENGTH-1 downto 0)

);
end component;
begin

pr:processor generic
map(WORDLENGTH=>WORDLENGTH,ROM_SIZE=>ROM_SIZE,REGISTER_SIZE=>REGISTER_SI
ZE,RAM_SIZE=>RAM_SIZE)
port map(
CLK=>CLK,
RESET=>RESET,

PC=>PC,
instr=>instr,
ALUResult=>ALUResult,
WriteData=>WriteData,
Result=>Result
);

CLK_process : process
begin
    CLK<='1';
    wait for 6.650ns;
    CLK<='0';
    wait for 6.650ns;
   end process;

stimulus_process: process
begin

    RESET <='1';
    wait for 13.300ns;
    wait until (CLK = '1' and CLK'event);
    RESET<='0';
    wait for 917.700ns;

end process;

end proccessor_sim_beh;
```

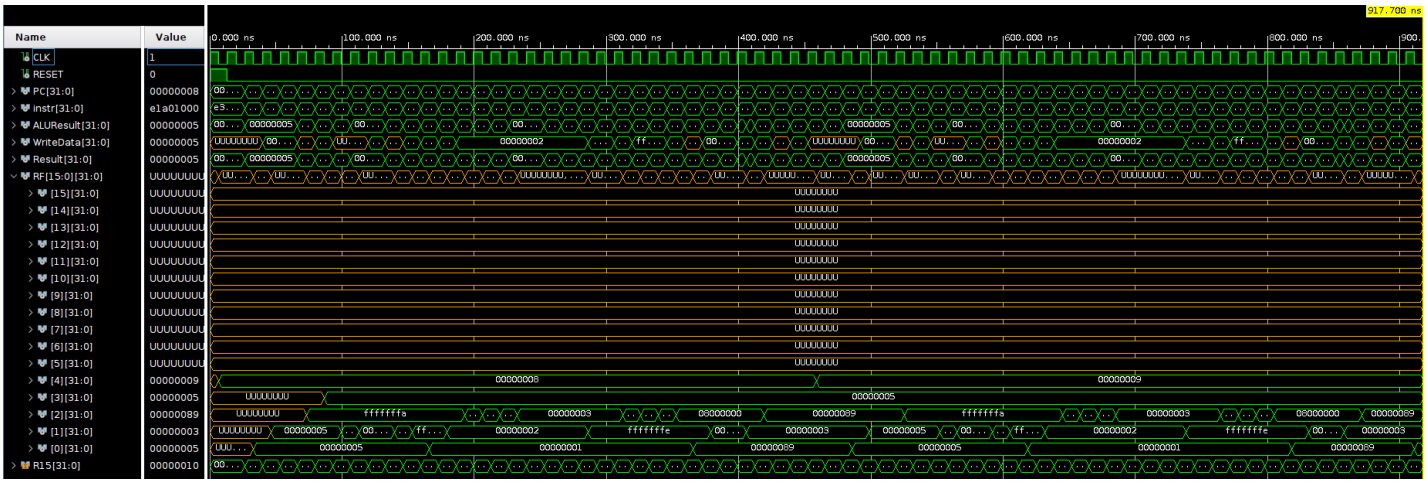
2.3 Κώδικες Προσομοίωσης

Σχεδιαγράμματα προσομοιώσεων. Επειδή η ορατότητα είναι μειωμένη σας παρουσιάζω τα συμπεράσματα. Ουσιαστικά τρέχουμε το πρόγραμμα 2 φορές όπως αναφέραμε στη

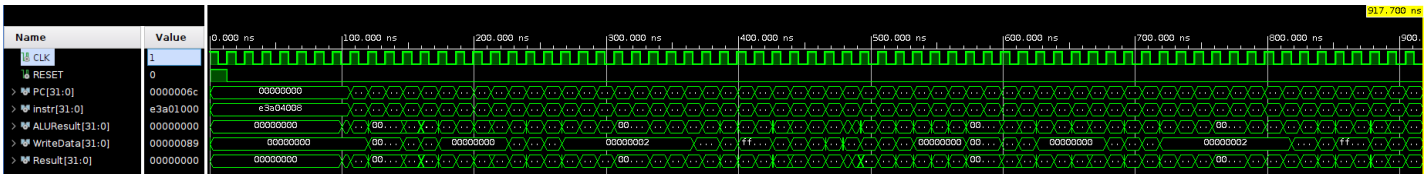
περιγραφή του κώδικα για τον έλεγχο των σημαιών. Σε κάθε κύκλο κοιτάμε προσεκτικά τα αποτελέσματα. Ένα σημαντικό εργαλείο είναι η ενεργοποίηση της καταγραφής της προσομοίωσης όλων των σημάτων (αλλάζει από τις ρυθμίσεις). Με αυτό τον τρόπο έχουμε πρόσβαση σε οποιαδήποτε τιμή σε όλο τον επεξεργαστή. Σύσταση μου είναι το άνοιγμα του αρχείου καταχωρητών καθώς βλέπετε με ευκολία όλα τα περιεχόμενα των καταχωρητών για τον έλεγχο της ορθότητας της εκτέλεσης (Έτσι ακριβώς φαίνεται και στην εικόνα). Φροντίζουμε λοιπόν η αλληλουχία των αποτελεσμάτων να ταυτίζεται με την αναμενόμενη.

Δυστυχώς το post implementation simulation και το post-synthesis simulation δεν δουλεύουν σωστά ακριβώς όπως έχουμε ορισμένο τον processor. Για αυτό το λόγο δημιούργησα ένα επιπλέον αρχείο προσομοίωσης το processor_sim_2. Αν λοιπόν από τον επεξεργαστή βγάλουμε το instr output (και στο entity και στο behavioral, έχω τοποθετήσει όλη τη λειτουργικότητα του σε μια γραμμή) τότε δουλεύουν τα πάντα. (Το processor_sim_2 έχει σωστά σε όλες τις προσομοιώσεις. Μόνο για να γίνει το implementation πρέπει να βγάλουμε το instr λόγω μη αρκετών I/O pins στην πλακέτα.)

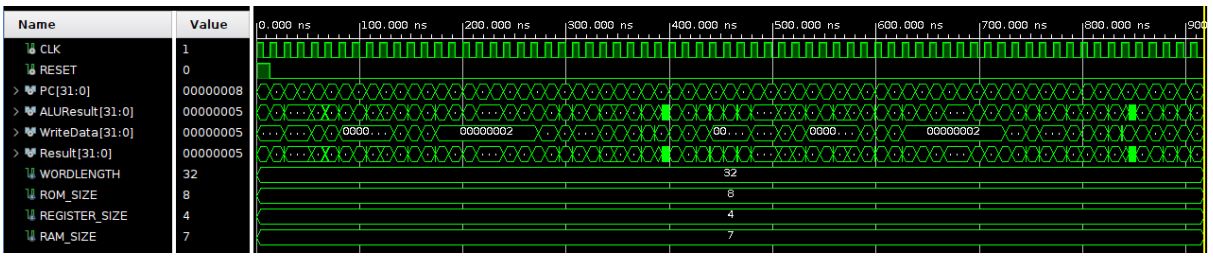
Behavioral Simulation:



Post-Synthesis Simulation:



Post-Implementation Simulation:



2.4 Synthesis και Implementation

Name	Slice LUTs (53200)	Slice Registers (106400)	Bonded IOB (200)	BUFGCTRL (32)
processor	653	79	162	2
cp (controlpath)	25	13	0	0
dp (datapath)	628	66	0	0

Synthesis

Status:	Complete
Messages:	33 warnings
Active run:	synth_1
Part:	xc7z020clg484-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	Automatically selected checkpoint

Παρατηρούμε ότι οι πόροι που χρησιμοποιήσαμε είναι αυτοί που περιμέναμε να χρησιμοποιήσουμε.