Πανεπιστήμιο Δυτικής Αττικής

Σχολή Μηχανικών

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Σχεδίαση Ψηφιακών Συστημάτων

Θεωρία Τμήμα 2

Εξαμηνιαία Εργασία Μαθήματος - Υλοποίηση Επεξεργαστή MIPS

Αναστασία Αλμπάνη

Ice19390009

Εισαγωγή

Το κύκλωμα επεξεργαστή MIPS αποτελείται από τις μονάδες:

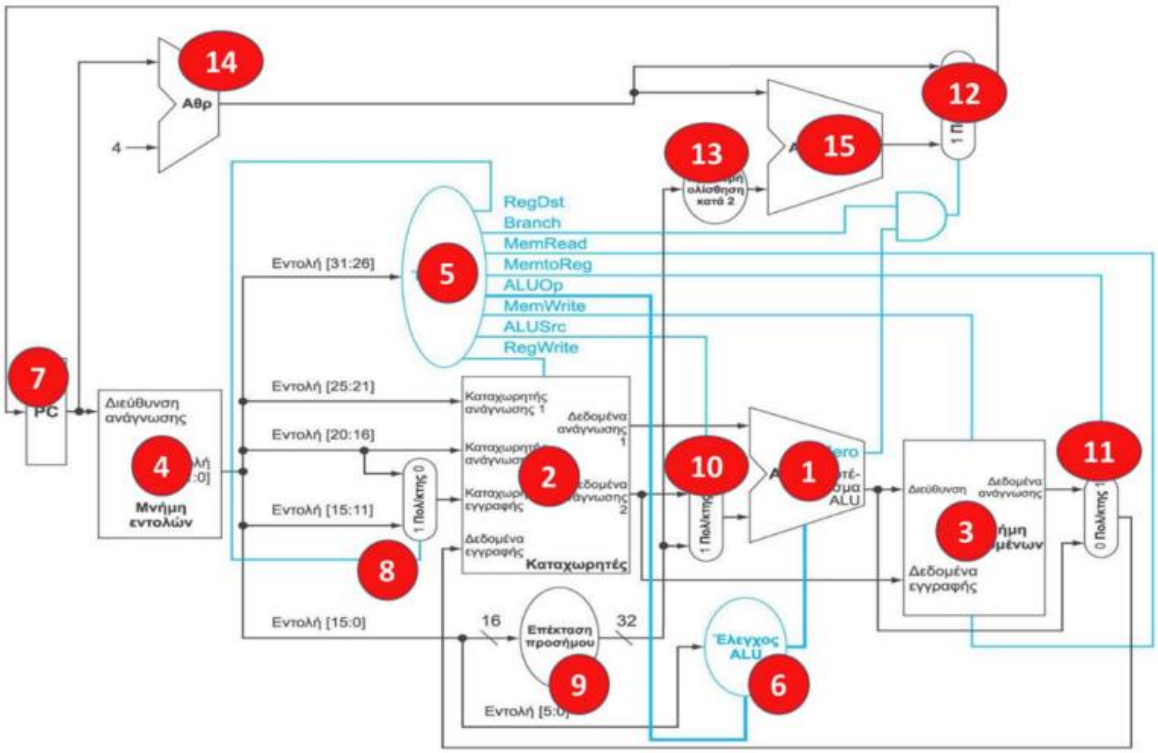
1. ALU
2. Register file
3. Μνήμη δεδομένων
4. Μνήμη εντολών
5. Μονάδα ελέγχου
6. Μονάδα ελέγχου ALU
7. PC
8. 5-πλό πολυπλέκτη 2-σε-1
9. Μονάδα επέκτασης προσήμου 16-σε-32

10, 11, 12. 32-πλό πολυπλέκτη 2-σε-1

13. Μονάδα ολίσθησης αριστερά κατά 2

14, 15. Αθροιστή 32 bits

Structural μονάδα MIPS η οποία συνδυάζει τις υπόλοιπες μονάδες



Οι εντολές οι οποίες υποστηρίζονται είναι: load word, store word, branch not equal, add, subtract, add immediate.

1. ALU

Το κύκλωμα ALU έχει δύο εισόδους δεδομένων και μία είσοδο operation η οποία ορίζει την πράξη των εισόδων. Η έξοδος `alu_res` είναι το αποτέλεσμα της πράξης και η έξοδος `zero` είναι ένα βοηθητικό σήμα που προσδιορίζει αν το αποτέλεσμα της εξόδου είναι μηδέν.

D1 (32 bit)	D2 (32 bit)	Operation	Alu resolution (32 bit)
X	X	0011 (addition)	D1 + D2
X	X	0110 (subtraction)	D1 - D2

Alu resolution (32 bit)	Zero
0	1
X	0

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;
USE IEEE.numeric_std.all;

entity ALU is port (
      -- ALU inputs
      D1 : in std_logic_vector(31 downto 0);
      D2 : in std_logic_vector(31 downto 0);
      operation : in std_logic_vector(3 downto 0);
      -- ALU outputs
      alu_res : out std_logic_vector(31 downto 0);
      zero : out std_logic
);
end ALU;

architecture behavioral of ALU is
```

```
	signal res : std_logic_vector(31 downto 0);

begin
	process(D1, D2, operation)
	begin
		case operation is
			-- addittion
			when "0011" =>
			res <= std_logic_vector(unsigned(D1) + unsigned(D2));
			-- subtract
			when "0110" =>
			res <= std_logic_vector(unsigned(D1) - unsigned(D2));
			when others => NULL;
		end case;
	end process;
	alu_res <= res;
	zero <= '1' when res = x"00000000" else
		'0';
end behavioral;
```

- Testbench

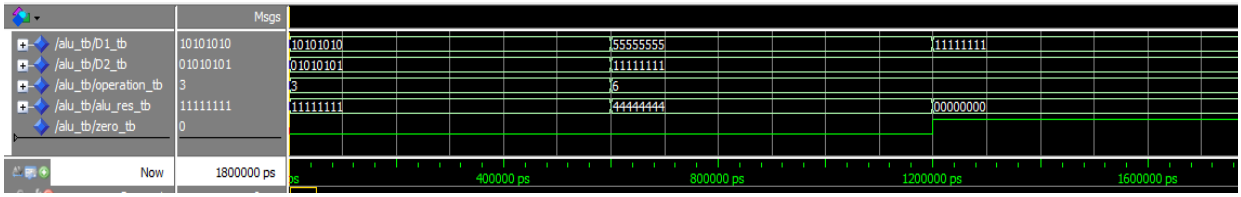
Το testbench ελέγχει πρώτα την πρόσθεση, μετά την αφαίρεση και τέλος την περίπτωση που το αποτέλεσμα της πράξης είναι 0.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity ALU_tb is
end ALU_tb;

architecture test of ALU_tb is

  component ALU
  port (
    D1 : in std_logic_vector(31 downto 0);
    D2 : in std_logic_vector(31 downto 0);
    operation : in std_logic_vector(3 downto 0);
    alu_res : out std_logic_vector(31 downto 0);
    zero : out std_logic -- zero
  );
  end component;

  -- inputs
  signal D1_tb : std_logic_vector(31 downto 0);
  signal D2_tb : std_logic_vector(31 downto 0);
  signal operation_tb : std_logic_vector(3 downto 0);

  -- outputs
  signal alu_res_tb : std_logic_vector(31 downto 0);
  signal zero_tb : std_logic;


begin
  U1: ALU port map (
    D1 => D1_tb,
    D2 => D2_tb,
```

```
    operation => operation_tb,
    alu_res => alu_res_tb,
    zero => zero_tb);

  -- stimulus process
  process
    begin

      -- addition
      D1_tb <= x"10101010";
      D2_tb <= x"01010101";
      operation_tb <= "0011";
      wait for 600 ns;

      -- subtract
      D1_tb <= x"55555555";
      D2_tb <= x"11111111";
      operation_tb <= "0110";
      wait for 600 ns;

      -- subtract equal
      D1_tb <= x"11111111";
      D2_tb <= x"11111111";
      operation_tb <= "0110";
      wait for 600 ns;

      -- runs for 1800 ns

  end process;
end test;
```


Msgs
/alu_tb/D1_tb
/alu_tb/D2_tb
/alu_tb/operation_tb
/alu_tb/alu_res_tb
/alu_tb/zero_tb
10101010
01010101
3
11111111
0
Now
1800000 ps
10101010
55555555
11111111
01010101
11111111
3
5
11111111
44444444
00000000
400000 ps
800000 ps
1200000 ps
1600000 ps

2. Register file

Το κύκλωμα Register_file έχει δύο εισόδους καταχωρητών ανάγνωσης, μία είσοδο καταχωρητή εγγραφής η οποία έχει για τιμή την διεύθυνση στην οποία θα γίνει εγγραφή, μία είσοδο δεδομένων εγγραφής και εισόδους για ρολόι, reset και RegWrite. Το αρχείο καταχωρητών `regfile` έχει 16 θέσεις των 32 bit. Το reset είναι ασύγχρονο και όταν έχει τιμή '1' τότε μηδενίζονται τα δεδομένα των καταχωρητών. Όταν RegWrite = '1' τότε γίνεται η διαδικασία εγγραφής δεδομένων σε μία επιλεγμένη θέση καταχωρητή. Για εξόδους έχει τα δεδομένα των καταχωρητών των εισόδων ανάγνωσης.

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

entity register_file is port (
    -- number of address bits = 5
      rAddr1: in std_logic_vector(4 downto 0);
      rAddr2: in std_logic_vector(4 downto 0);
      wAddr : in std_logic_vector(4 downto 0);
      --  number of bits = 32
      wD : in std_logic_vector(31 downto 0);
      RegWrite : in std_logic;
      clk : in std_logic;
      reset : in std_logic;
      rD1 : out std_logic_vector(31 downto 0);
      rD2 : out std_logic_vector(31 downto 0));
end register_file;

architecture behavioral of register_file is
```

```
	type regArray is array(0 to 15) of std_logic_vector(31 downto 0);


	signal regfile : regArray;
begin
	process(clk, reset)
	begin
		-- reset
		if reset = '1' then
			regfile <= (others => (others => '0'));
		end if;
		if (clk'event and clk='1') then
		  -- write to register
			if RegWrite='1' then
				regfile(to_integer(unsigned(wAddr))) <= wD;
			end if;
		end if;
	end process;
	-- read from register
	rD1 <= regfile(to_integer(unsigned(rAddr1)));
	rD2 <= regfile(to_integer(unsigned(rAddr2)));
end behavioral;
```

- Testbench

Στο testbench γίνεται reset και στη συνέχεια γράφονται δεδομένα σε όλες τις θέσεις των καταχωρητών. Τέλος διαβάζονται τα δεδομένα των καταχωρητών 2 και 3.

- Κώδικας

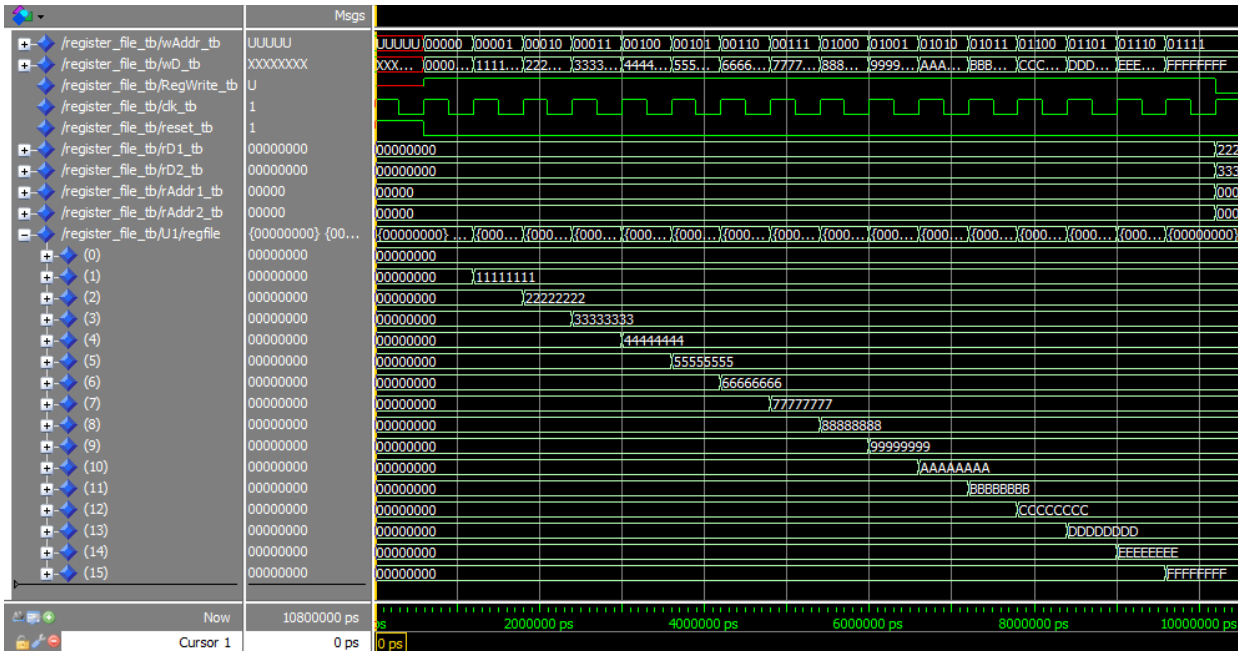
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY register_file_tb is
END register_file_tb;

ARCHITECTURE test OF register_file_tb IS

	COMPONENT register_file PORT(
		rAddr1: in std_logic_vector(4 downto 0);
		rAddr2: in std_logic_vector(4 downto 0);
		wAddr : in std_logic_vector(4 downto 0);
		wD : in std_logic_vector(31 downto 0);
		RegWrite : in std_logic;
		clk : in std_logic;
		reset : in std_logic;
		rD1 : out std_logic_vector(31 downto 0);
		rD2 : out std_logic_vector(31 downto 0));
	END COMPONENT;

	-- inputs
	SIGNAL rAddr1_tb : std_logic_vector(4 downto 0);
	SIGNAL rAddr2_tb : std_logic_vector(4 downto 0);
	SIGNAL wAddr_tb : std_logic_vector(4 downto 0);
	SIGNAL wD_tb : std_logic_vector(31 downto 0);
	SIGNAL RegWrite_tb : std_logic;
```

```
	SIGNAL clk_tb : std_logic;
	SIGNAL reset_tb : std_logic;

	-- outputs
	SIGNAL rD1_tb : std_logic_vector(31 downto 0);
	SIGNAL rD2_tb : std_logic_vector(31 downto 0);

BEGIN

	U1: register_file PORT MAP (
		rAddr1 => rAddr1_tb,
		rAddr2 => rAddr2_tb,
		wAddr => wAddr_tb,
		wD => wD_tb,
		RegWrite => RegWrite_tb,
		clk => clk_tb,
		reset => reset_tb,
		rD1 => rD1_tb,
		rD2 => rD2_tb);

	-- clock process
	clk_p: PROCESS
	BEGIN
		clk_tb <= '1';
		WAIT FOR 300 ns;
		clk_tb <= '0';
		WAIT FOR 300 ns;
	END PROCESS clk_p;

	-- stimulus process
```

```
PROCESS
BEGIN
	-- reset
	reset_tb <= '1';
	rAddr1_tb <= "00000";
	rAddr2_tb <= "00000";
	WAIT FOR 600 ns;

	-- write to register
	reset_tb <= '0';
	RegWrite_tb <= '1';

	wAddr_tb <= "00000";
	wD_tb <= x"00000000";
	WAIT FOR 600 ns;

	wAddr_tb <= "00001";
	wD_tb <= x"11111111";
	WAIT FOR 600 ns;

	wAddr_tb <= "00010";
	wD_tb <= x"22222222";
	WAIT FOR 600 ns;

	wAddr_tb <= "00011";
	wD_tb <= x"33333333";
	WAIT FOR 600 ns;

	wAddr_tb <= "00100";
	wD_tb <= x"44444444";
```

```
WAIT FOR 600 ns;

wAddr_tb <= "00101";
wD_tb <= x"55555555";
WAIT FOR 600 ns;

wAddr_tb <= "00110";
wD_tb <= x"66666666";
WAIT FOR 600 ns;

wAddr_tb <= "00111";
wD_tb <= x"77777777";
WAIT FOR 600 ns;

wAddr_tb <= "01000";
wD_tb <= x"88888888";
WAIT FOR 600 ns;

wAddr_tb <= "01001";
wD_tb <= x"99999999";
WAIT FOR 600 ns;

wAddr_tb <= "01010";
wD_tb <= x"AAAAAAAA";
WAIT FOR 600 ns;

wAddr_tb <= "01011";
wD_tb <= x"BBBBBBBB";
WAIT FOR 600 ns;
```

```
            wAddr_tb <= "01100";
            wD_tb <= x"CCCCCCCC";
            WAIT FOR 600 ns;

            wAddr_tb <= "01101";
            wD_tb <= x"DDDDDDDD";
            WAIT FOR 600 ns;

            wAddr_tb <= "01110";
            wD_tb <= x"EEEEEEEE";
            WAIT FOR 600 ns;

            wAddr_tb <= "01111";
            wD_tb <= x"FFFFFFFF";
            WAIT FOR 600 ns;

            -- read from register
            RegWrite_tb <= '0';
            rAddr1_tb <= "00010";
            rAddr2_tb <= "00011";
            WAIT FOR 600 ns;

            -- runs for 10800 ns
      END PROCESS;
END test;
```


Msgs
/register_file_tb/wAddr_tb
/register_file_tb/wD_tb
/register_file_tb/RegWrite_tb
/register_file_tb/clk_tb
/register_file_tb/reset_tb
/register_file_tb/rD1_tb
/register_file_tb/rD2_tb
/register_file_tb/rAddr1_tb
/register_file_tb/rAddr2_tb
/register_file_tb/U1/regfile
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
UUUUU
XXXXXXXX
U
1
1
00000000
00000
{00000000} {00...
11111111
22222222
33333333
44444444
55555555
66666666
77777777
88888888
99999999
AAAAAAAA
BBBBBBBB
CCCCCCCC
DDDDDDDD
EEEEEEEE
FFFFFFFF
Now
10800000 ps
Cursor 1
0 ps
2000000 ps
4000000 ps
6000000 ps
8000000 ps
10000000 ps

3. Μνήμη δεδομένων (Data memory)

Το κύκλωμα data_memory έχει για εισόδους μία διεύθυνση, τα δεδομένα εγγραφής και εισόδους για ρολόι, reset, MemWrite και MemRead. Για έξοδο έχει τα δεδομένα ανάγνωσης. Το αρχείο μνήμης δεδομένων data_mem_file έχει χωρητικότητα 16 θέσεων μνήμης των 32 bit. Το reset είναι ασύγχρονο και όταν έχει τιμή '1' τότε μηδενίζονται τα δεδομένα των καταχωρητών. Όταν MemWrite = '1' τότε γράφεται στην διεύθυνση τα δεδομένα εγγραφής και όταν MemRead = '1' τότε τα δεδομένα ανάγνωσης παίρνουν την τιμή των δεδομένων στην διεύθυνση.

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;
USE IEEE.numeric_std.all;

entity data_memory is port(
	-- data memory inputs
	address : in std_logic_vector(31 downto 0);
	write_data : in std_logic_vector(31 downto 0);
	clk : in std_logic;
	reset : in std_logic;
	MemWrite : in std_logic;
	MemRead : in std_logic;
	-- data memory output
	read_data : out std_logic_vector(31 downto 0)
);
end data_memory;

architecture behavioral of data_memory is

	type data_mem_array is array(0 to 15) of std_logic_vector(31 downto 0);
```

```
	-- data memory file to zero
	signal data_mem_file : data_mem_array := ((others => (others => '0')));
begin
	process(clk, reset)
	begin
		-- reset
		if reset = '1' then
			data_mem_file <= (others => (others => '0'));
		end if;
		-- clock
		if (clk'event and clk='1') then
			if (MemWrite = '1') then
			 -- write data to address
				data_mem_file(to_integer(unsigned(address))) <=
			write_data;
			end if;

			if (MemRead = '1') then
			 -- read data from address
				read_data <=
			data_mem_file(to_integer(unsigned(address)));
			else read_data <= x"00000000";
			end if;
		end if;
	end process;
end behavioral;
```

- Testbench

Στο testbench γίνεται reset και στη συνέχεια γράφονται δεδομένα σε όλους τους καταχωρητές. Τέλος διαβάζονται τα δεδομένα του κσταχωρητή 10 και ξαναγίνεται reset.

- Κώδικας

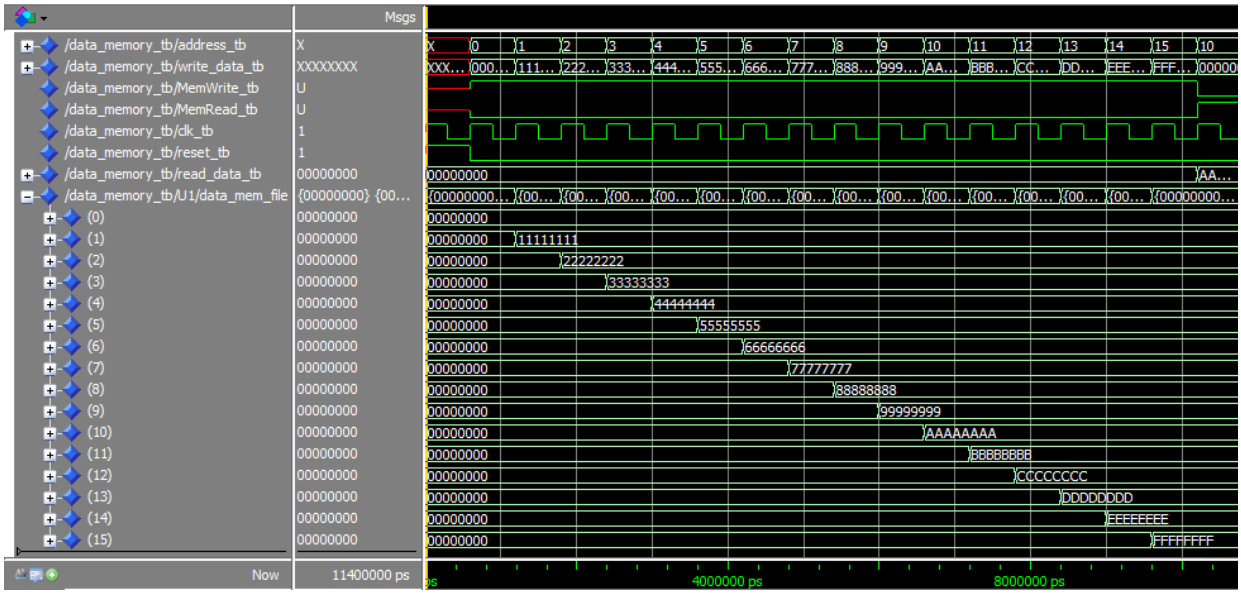
```
library IEEE;
use IEEE.std_logic_1164.all;

entity data_memory_tb is
end data_memory_tb;

architecture test of data_memory_tb is

  component data_memory
  port (
    address : in std_logic_vector(31 downto 0);
    write_data : in std_logic_vector(31 downto 0);
    clk : in std_logic;
    reset : in std_logic;
    MemWrite : in std_logic;
    MemRead : in std_logic;
    read_data : out std_logic_vector(31 downto 0)
  );
  end component;

  -- inputs
  signal address_tb : std_logic_vector(31 downto 0);
  signal write_data_tb : std_logic_vector(31 downto 0);
  signal MemWrite_tb : std_logic;
```

```
  signal MemRead_tb : std_logic;
  signal clk_tb : std_logic;
  signal reset_tb : std_logic;

  -- output
  signal read_data_tb : std_logic_vector(31 downto 0);


begin
  U1: data_memory port map (
    address  => address_tb,
    write_data => write_data_tb,
    clk => clk_tb,
    reset => reset_tb,
    MemWrite => MemWrite_tb,
    MemRead => MemRead_tb,
    read_data => read_data_tb);

  -- clock process
clk_p: process
  begin
   clk_tb <= '1';
   wait for 300 ns;
   clk_tb <= '0';
   wait for 300 ns;
end process clk_p;

  -- stimulus process
  process
    begin
```

```
-- reset
reset_tb <= '1';
wait for 600 ns;

-- write to data memory
reset_tb <= '0';
MemRead_tb <= '0';
MemWrite_tb <= '1';
address_tb <= x"00000000";
write_data_tb <= x"00000000";
wait for 600 ns;

address_tb <= x"00000001";
write_data_tb <= x"11111111";
wait for 600 ns;

address_tb <= x"00000002";
write_data_tb <= x"22222222";
wait for 600 ns;

address_tb <= x"00000003";
write_data_tb <= x"33333333";
wait for 600 ns;

address_tb <= x"00000004";
write_data_tb <= x"44444444";
wait for 600 ns;

address_tb <= x"00000005";
```

```
write_data_tb <= x"55555555";
wait for 600 ns;

address_tb <= x"00000006";
write_data_tb <= x"66666666";
wait for 600 ns;

address_tb <= x"00000007";
write_data_tb <= x"77777777";
wait for 600 ns;

address_tb <= x"00000008";
write_data_tb <= x"88888888";
wait for 600 ns;

address_tb <= x"00000009";
write_data_tb <= x"99999999";
wait for 600 ns;

address_tb <= x"0000000A";
write_data_tb <= x"AAAAAAAA";
wait for 600 ns;

address_tb <= x"0000000B";
write_data_tb <= x"BBBBBBBB";
wait for 600 ns;

address_tb <= x"0000000C";
write_data_tb <= x"CCCCCCCC";
wait for 600 ns;
```

```
        address_tb <= x"0000000D";
        write_data_tb <= x"DDDDDDDD";
        wait for 600 ns;

        address_tb <= x"0000000E";
        write_data_tb <= x"EEEEEEEE";
        wait for 600 ns;

        address_tb <= x"0000000F";
        write_data_tb <= x"FFFFFFFF";
        wait for 600 ns;

        -- read from data memory
        address_tb <= x"0000000A";
        write_data_tb <= x"00000000";
        MemRead_tb <= '1';
        MemWrite_tb <= '0';
        wait for 600 ns;

        -- reset
        reset_tb <= '1';
        MemRead_tb <= '0';
        MemWrite_tb <= '0';
        wait for 600 ns;

        -- runs for 11400 ns
    end process;
end test;
```


Msgs
/data_memory_tb/address_tb
/data_memory_tb/write_data_tb
/data_memory_tb/MemWrite_tb
/data_memory_tb/MemRead_tb
/data_memory_tb/clk_tb
/data_memory_tb/reset_tb
/data_memory_tb/read_data_tb
/data_memory_tb/U1/data_mem_file
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
Now
11400000 ps
4000000 ps
8000000 ps

4. Μνήμη εντολών (Instruction memory)

Το κύκλωμα `instruction_memory` έχει για εισόδους την διεύθυνση ανάγνωσης, το ρολόι και το reset. Η έξοδος είναι η εντολή του επεξεργαστή η οποία διαβάζεται από τη διεύθυνση ανάγνωσης. Το αρχείο μνήμης εντολών `instr_mem_file` έχει χωρητικότητα 16 θέσεων μνήμης των 32 bit οι οποίες έχουν αποθηκευμένες της εντολές. Οι εντολές οι οποίες αποθηκεύονται είναι:

Addi $0, $0, 0

Addi $2, $2, 0

Addi $2, $4, 0

Addi $3, $0, 1

Addi $5, $0, 3

L1: add $6, $3, $0

Sw $6, $3, $0

Addi $3, $3, 1

Addi $4, $4, 1

Addi $5, $5, -1

Bne $5, $0, L1

Το reset είναι ασύγχρονο και όταν έχει τιμή ‘1’ τότε μηδενίζεται η διεύθυνση ανάγνωσης για να αρχίσει το κύκλωμα να διαβάζει εντολές από την πρώτη θέση μνήμης.

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;
USE IEEE.numeric_std.all;

entity instruction_memory is port (
      -- MIPS inputs
```

```
      read_address : in std_logic_vector(31 downto 0);
      clk : in std_logic;
      reset: in std_logic;
      -- instruction memory output
      instruction : out std_logic_vector(31 downto 0)
);
end instruction_memory;

architecture behavioral of instruction_memory is

      type instr_mem_array is array(0 to 15) of std_logic_vector(31 downto
0);

      -- instruction memory initilization
      signal instr_mem_file : instr_mem_array := (
x"20000000", -- 0x0000 0000: addi $0, $0, 0
                                    (001000 00000 00000 00000 00000 000000)
x"20420000", -- 0x0000 0001: addi $2, $2, 0
                                    (001000 00010 00010 00000 00000 000000)
x"20820000", -- 0x0000 0002: addi $2, $4, 0
                                    (001000 00100 00010 00000 00000 000000)
x"20030001", -- 0x0000 0003: addi $3, $0, 1
                                    (001000 00000 00011 00000 00000 000001)
x"20050003", -- 0x0000 0004: addi $5, $0, 3
                                    (001000 00000 00101 00000 00000 000011)
x"00603020", -- 0x0000 0005: L1: add $6, $3, $0
                                    (000000 00011 00000 00110 00000 100000)
x"AC860000", -- 0x0000 0006: sw $6, 0($4)
                                    (101011 00100 00110 00000 00000 000000)
x"20630001", -- 0x0000 0007: addi $3, $3, 1
```

```
                                    (001000 00011 00011 00000 00000 000001)
x"20840001", -- 0x0000 0008: addi $4, $4, 1
                                    (001000 00100 00100 00000 00000 000001)
x"20A5FFFF", -- 0x0000 0009: addi $5, $5, -1
                                    (001000 00101 00101 11111 11111 111111)
x"14A0FFF9", -- 0x0000 000A: bne $5, $0, L1
                                    (000101 00101 00000 11111 11111 111001)
x"00000000",
x"00000000",
x"00000000",
x"00000000",
x"00000000");

begin
      process(clk, reset)
      begin
            if (reset = '1') then
                  instruction <= x"00000000";
            end if;
            if (clk'event and clk='1') then
                  instruction <=
            instr_mem_file(to_integer(unsigned(read_address)));
            end if;
      end process;
end behavioral;
```

- Testbench

Στο testbench γίνεται reset και στη συνέχεια διαβάζονται όλοι οι καταχωρητές της μνήμης εντολών.

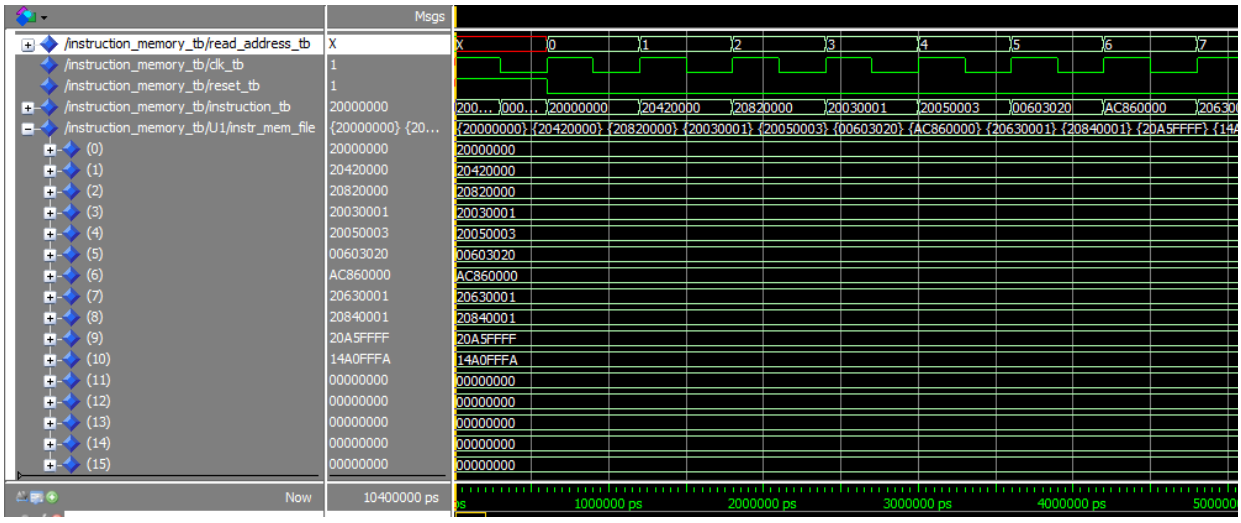
- Κώδικας

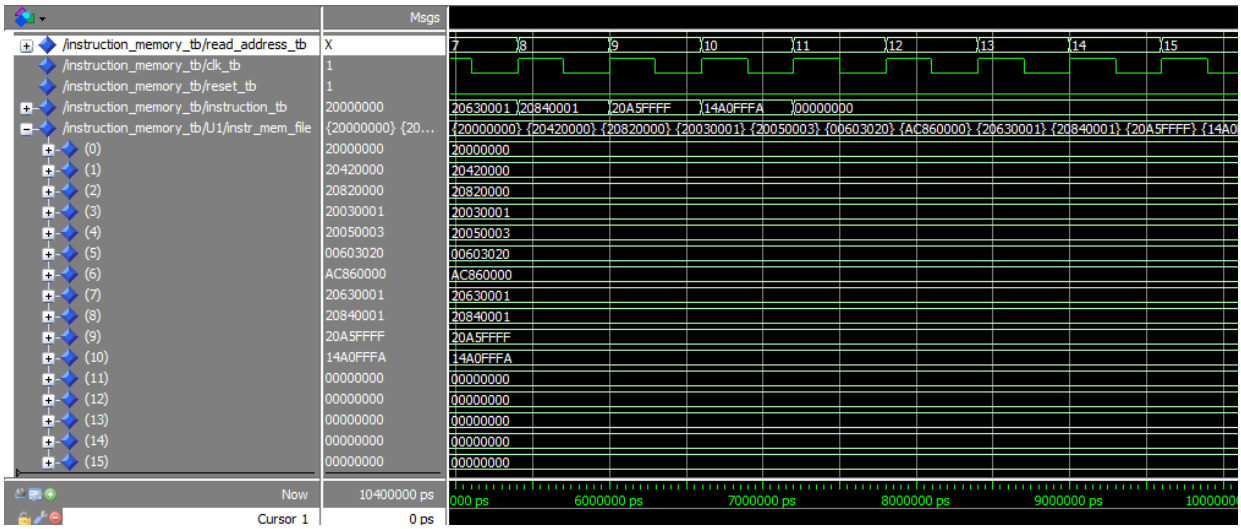
```
library IEEE;
use IEEE.std_logic_1164.all;

entity instruction_memory_tb is
end instruction_memory_tb;

architecture test of instruction_memory_tb is

  component instruction_memory
  port (
    read_address : in std_logic_vector(31 downto 0);
    clk : in std_logic;
    reset: in std_logic;
    instruction : out std_logic_vector(31 downto 0)
);
  end component;

  -- inputs
  signal read_address_tb : std_logic_vector(31 downto 0);
  signal clk_tb : std_logic;
  signal reset_tb : std_logic;

  -- output
  signal instruction_tb : std_logic_vector(31 downto 0);


begin
```

```
  U1: instruction_memory port map (
    read_address => read_address_tb,
    clk => clk_tb,
    reset => reset_tb,
    instruction => instruction_tb);

  -- clock process
clk_p: process
  begin
    clk_tb <= '1';
    wait for 300 ns;
    clk_tb <= '0';
    wait for 300 ns;
end process clk_p;

  -- stimulus process
  process
    begin
      -- reset
      reset_tb <= '1';
      wait for 600 ns;

      -- read instruction
      reset_tb <= '0';
      read_address_tb <= x"00000000";
      wait for 600 ns;

      read_address_tb <= x"00000001";
      wait for 600 ns;
```

```
read_address_tb <= x"00000002";
wait for 600 ns;

read_address_tb <= x"00000003";
wait for 600 ns;

read_address_tb <= x"00000004";
wait for 600 ns;

read_address_tb <= x"00000005";
wait for 600 ns;

read_address_tb <= x"00000006";
wait for 600 ns;

read_address_tb <= x"00000007";
wait for 600 ns;

read_address_tb <= x"00000008";
wait for 600 ns;

read_address_tb <= x"00000009";
wait for 600 ns;

read_address_tb <= x"0000000A";
wait for 600 ns;

read_address_tb <= x"0000000B";
wait for 600 ns;
```

```
        read_address_tb <= x"0000000C";
        wait for 600 ns;

        read_address_tb <= x"0000000D";
        wait for 600 ns;

        read_address_tb <= x"0000000E";
        wait for 600 ns;

        read_address_tb <= x"0000000F";
        wait for 600 ns;

        -- runs for 10400 ns

    end process;
end test;
```




Msgs
/instruction_memory_tb/read_address_tb
/instruction_memory_tb/clk_tb
/instruction_memory_tb/reset_tb
/instruction_memory_tb/instruction_tb
/instruction_memory_tb/U1/instr_mem_file
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
X
1
1
20000000
{20000000} {20...
20000000
20420000
20820000
20030001
20050003
00603020
AC860000
20630001
20840001
20A5FFFF
14A0FFFA
00000000
00000000
00000000
00000000
00000000
7
8
9
10
11
12
13
14
15
20630001
20840001
20A5FFFF
14A0FFFA
00000000
{20000000} {20420000} {20820000} {20030001} {20050003} {00603020} {AC860000} {20630001} {20840001} {20A5FFFF} {14A0
Now
10400000 ps
Cursor 1
0 ps
000 ps
6000000 ps
7000000 ps
8000000 ps
9000000 ps
1000000

5. Μονάδα ελέγχου (Control unit)

Το κύκλωμα `control_unit` έχει για είσοδο τον κώδικα διαδικασίας και για εξόδους τα σήματα RegDst, Branch, MemRead, MemtoReg, ALUop, MemWrite, ALUSrc και RegWrite.

Εντολή	opcode	Reg Dst	Branch	Mem Read	Memto Reg	ALUop	Mem Write	ALUSrc	RegWrite
load word	100011	0	0	1	1	00	0	1	1
store word	101011	X	0	0	X	00	1	1	0
branch not equal	000101	X	1	0	X	01	0	0	0
add	000000	1	0	0	0	10	0	0	1
subtract	000000	1	0	0	0	10	0	0	1
add immediate	001000	0	0	0	0	11	0	1	1

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;

entity control_unit is port (
	-- operation code for 31-26 of instruction memory
	opcode : in std_logic_vector(5 downto 0);
	-- control signals of operation
	RegDst : out std_logic;
	Branch : out std_logic;
	MemRead : out std_logic;
	MemtoReg : out std_logic;
	ALUOp : out std_logic_vector(1 downto 0);
	MemWrite : out std_logic;
	ALUSrc : out std_logic;
	RegWrite : out std_logic
```

```
);
end control_unit;

architecture behavioral of control_unit is
	begin
	process(opcode)
	Begin
		case opcode is
			-- Register type instructions : and, sub
			when "000000" =>
				RegDst <= '1'; -- register address 15-11 to write
				Branch <= '0';
				MemRead <= '0';
				MemtoReg <= '0';
				ALUOp <= "10";
				MemWrite <= '0';
				ALUSrc <= '0';
				RegWrite <= '1'; -- write data to register address
			-- Immediate type instructions
			when "001000" => -- addi (add immediate word)
				RegDst <= '0'; -- register address 20-16 to write
				Branch <= '0';
				MemRead <= '0';
				MemtoReg <= '0';
				ALUOp <= "11";
				MemWrite <= '0';
				ALUSrc <= '1'; -- alu uses integer value 15-0
				RegWrite <= '1'; -- write data to register address
			when "100011" => -- lw (load word)
				RegDst <= '0';
```

```
                        Branch <= '0';
                        MemRead <= '1'; -- read from data memory
                        MemtoReg <= '1';
                        ALUOp <= "00";
                        MemWrite <= '0';
                        ALUSrc <= '1'; -- alu uses offset value 15-0
                        RegWrite <= '1'; -- write data to register address
                  when "101011" => -- sw (store word)
                        RegDst <= 'X';
                        Branch <= '0';
                        MemRead <= '0';
                        MemtoReg <= 'X';
                        ALUOp <= "00";
                        MemWrite <= '1'; -- write to data memory
                        ALUSrc <= '1'; -- alu uses offset value 15-0
                        RegWrite <= '0';
                  when "000101" => -- bne (branch not equal)
                        RegDst <= 'X';
                        Branch <= '1'; -- add value to program counter
                        MemRead <= '0';
                        MemtoReg <= 'X';
                        ALUOp <= "01";
                        MemWrite <= '0';
                        ALUSrc <= '0';
                        RegWrite <= '0';
                  when others => NULL;
            end case;
      end process;
end behavioral;
```

- Testbench

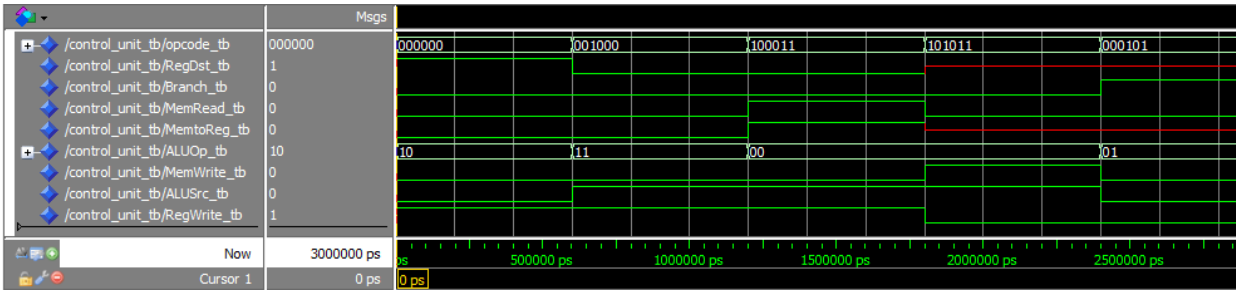
Στο testbench ελέγχονται τα σήματα του control unit για τις εντολές add και sub, addi, lw, sw και bne.

- Κώδικας

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY control_unit_tb is
END control_unit_tb;

ARCHITECTURE test OF control_unit_tb IS

COMPONENT control_unit PORT(
   opcode : in std_logic_vector(5 downto 0);
   RegDst : out std_logic;
   Branch : out std_logic;
   MemRead : out std_logic;
   MemtoReg : out std_logic;
   ALUOp : out std_logic_vector(1 downto 0);
   MemWrite : out std_logic;
   ALUSrc : out std_logic;
   RegWrite : out std_logic);
END COMPONENT;

  -- input
SIGNAL opcode_tb : std_logic_vector(5 downto 0);
```

```
-- outputs
SIGNAL RegDst_tb : std_logic;
SIGNAL Branch_tb : std_logic;
SIGNAL MemRead_tb : std_logic;
SIGNAL MemtoReg_tb : std_logic;
SIGNAL ALUOp_tb : std_logic_vector(1 downto 0);
SIGNAL MemWrite_tb : std_logic;
SIGNAL ALUSrc_tb : std_logic;
SIGNAL RegWrite_tb : std_logic;

BEGIN

U1: control_unit PORT MAP (
opcode => opcode_tb,
RegDst => RegDst_tb,
Branch => Branch_tb,
MemRead => MemRead_tb,
MemtoReg => MemtoReg_tb,
ALUOp => ALUOp_tb,
MemWrite => MemWrite_tb,
ALUSrc => ALUSrc_tb,
RegWrite => RegWrite_tb);

-- stimulus process
PROCESS
BEGIN
    -- add, sub
      opcode_tb <= "000000";
      WAIT FOR 600 ns;
```

```
        -- addi
        opcode_tb <= "001000";
        WAIT FOR 600 ns;

        -- lw (load word)
        opcode_tb <= "100011";
        WAIT FOR 600 ns;

        -- sw (store word)
        opcode_tb <= "101011";
        WAIT FOR 600 ns;

        -- bne (branch not equal)
        opcode_tb <= "000101";
        WAIT FOR 600 ns;

        -- runs for 3000 ns
END PROCESS;
END test;
```



6. Μονάδα ελέγχου ALU

Το κύκλωμα ALU_control έχει για είσοδο το σήμα ALUop και το τμήμα funct της εντολής του επεξεργαστή και επιστρέφει στην ALU το operation το οποίο καθορίζει αν θα γίνει πρόσθεση ή αφαίρεση. Για πρόσθεση το operation είναι 0011 και για αφαίρεση είναι 0110.

Εντολή	Τύπος εντολής	funct	Aluop	Operation
load word	I	xxxxxx	00	0011
store word	I	xxxxxx	00	0011
branch not equal	I	xxxxxx	01	0110
add	R	100000	10	0011
subtract	R	100010	10	0110
add immediate	I	xxxxxx	11	0011

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;

entity ALU_control is port (
      -- operation and function for ALU
      ALUop : in std_logic_vector(1 downto 0);
      funct : in std_logic_vector(5 downto 0);
      -- ALU operation
      operation : out std_logic_vector(3 downto 0)
);
end ALU_control;

architecture behavioral of ALU_control is
```

```
begin
  process(ALUop, funct)
  Begin
      if ALUop = "01" then             -- branch not equal
            operation <= "0110";
      elsif ALUop = "10" then
            if funct = "100000" then            -- add
                  operation <= "0011";
            elsif funct = "100010" then         -- subtract
                  operation <= "0110";
            end if;
      elsif ALUop = "11" then          -- add immediate
            operation <= "0011";
      else -- ALUop = "00"
            operation <= "0011";           -- load word and store word
      end if;
  end process;
end behavioral;
```

- Testbench

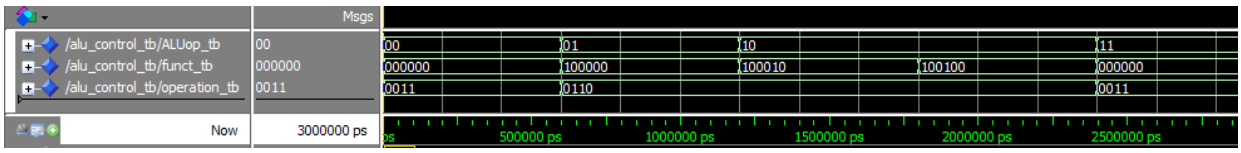
Στο testbench ελέγχεται το operation για τα ALUop, funct των εντολών lw, sw, bne, add, sub και addi.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ALU_control_tb is
end ALU_control_tb;

architecture test of ALU_control_tb is

  component ALU_control
  port (
    ALUop : in std_logic_vector(1 downto 0);
    funct : in std_logic_vector(5 downto 0);
    operation : out std_logic_vector(3 downto 0)
  );
  end component;

  -- inputs
  signal ALUop_tb : std_logic_vector(1 downto 0);
  signal funct_tb : std_logic_vector(5 downto 0);

  -- output
  signal operation_tb : std_logic_vector(3 downto 0);


begin
  U1: ALU_control port map (
    ALUop => ALUop_tb,
```

```
  funct => funct_tb,
  operation => operation_tb);

-- stimulus process
process
  begin

    -- load word and store word
    ALUop_tb <= "00";
    funct_tb <= "000000";
    wait for 600 ns;

    -- branch not equal
    ALUop_tb <= "01";
    funct_tb <= "100000";
    wait for 600 ns;

    -- add
    ALUop_tb <= "10";
    funct_tb <= "100010";
    wait for 600 ns;

    -- subtract
    ALUop_tb <= "10";
    funct_tb <= "100100";
    wait for 600 ns;

    -- addi
    ALUop_tb <= "11";
    funct_tb <= "000000";
```

```
      wait for 600 ns;

      -- runs for 3000 ns

  end process;
end test;
```



7. PC

Το κύκλωμα PC έχει για είσοδο την διεύθυνση ανάγνωσης, το ρολόι και το reset και για έξοδο την διεύθυνση ανάγνωσης της εισόδου. Συγχρονίζει με βάση το ρολόι το διάβασμα της διεύθυνσης στη μνήμη εντολών. Το reset είναι ασύγχρονο και όταν έχει τιμή ‘1’ τότε θέτει τη διεύθυνση σε x00000000.

PCin (32 bit)	reset	PCout (32 bit)
XXXXXXXX	0	PCin
XXXXXXXX	1	x00000000

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;

entity PC is port (
	-- PC inputs
	PCin : in std_logic_vector(31 downto 0);
	reset : in std_logic;
	clk : in std_logic;
	-- PC output
	PCout : out std_logic_vector(31 downto 0)
);
end PC;

architecture behavioral of PC is
	signal pout: std_logic_vector(31 downto 0);

begin
	process(clk, reset)
	begin
```

```
            if reset = '1' then
                  pout <= x"00000000";
            end if;
            if (clk'event and clk='1') then
                  pout <= PCin;
            end if;
      end process;
      PCout <= pout;
end behavioral;
```

- Testbench

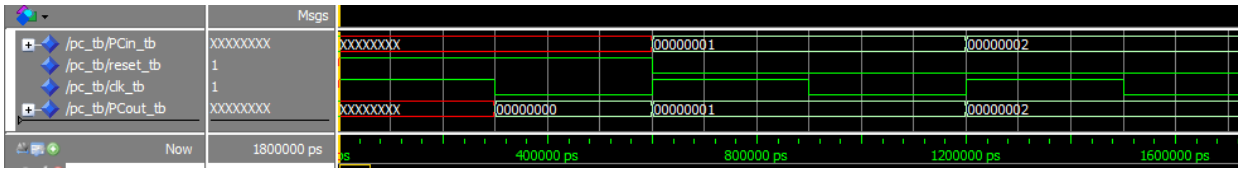
Στο testbench ελέγχεται το reset και το PCout για PCin ίσο με x00000001 και x00000002.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity PC_tb is
end PC_tb;

architecture test of PC_tb is

  component PC
  port (
    PCin : in std_logic_vector(31 downto 0);
  reset : in std_logic;
```

```
    clk : in std_logic;
    PCout : out std_logic_vector(31 downto 0)
  );
  end component;

  -- inputs
  signal PCin_tb : std_logic_vector(31 downto 0);
  signal reset_tb : std_logic;
  signal clk_tb : std_logic;

  -- output
  signal PCout_tb : std_logic_vector(31 downto 0);



begin
  U1: PC port map (
    PCin => PCin_tb,
    reset => reset_tb,
    clk => clk_tb,
    PCout => PCout_tb);

  -- clock process
clk_p: process
begin
      clk_tb <= '1';
      wait for 300 ns;
      clk_tb <= '0';
      wait for 300 ns;
end process clk_p;
```

```
  -- stimulus process
  process
    begin

      reset_tb <= '1';
      wait for 600 ns;

      reset_tb <= '0';
      PCin_tb <= x"00000001";
      wait for 600 ns;

      PCin_tb <= x"00000002";
      wait for 600 ns;

      -- runs for 1800 ns

  end process;
end test;
```



8. 5-πλός Πολυπλέκτης 2-σε-1

Το κύκλωμα `mux2to1_5bit` έχει δύο εισόδους 5 bit, μία είσοδο select και μία έξοδο 5 bit η οποία ανάλογα με την τιμή της select παίρνει την τιμή μίας εισόδου.

A1 (5 bit)	A2 (5 bit)	s	D (5 bit)
XXXXX	XXXXX	1	A1
XXXXX	XXXXX	0	A2

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux2to1_5bit is port (
  a1: in std_logic_vector(4 downto 0);
  a2: in std_logic_vector(4 downto 0);
  s : in std_logic;
  d: out std_logic_vector(4 downto 0)
  );
end mux2to1_5bit;

architecture dataflow of mux2to1_5bit is
begin
  d <= a1 when s = '1' else
       a2;
end dataflow;
```

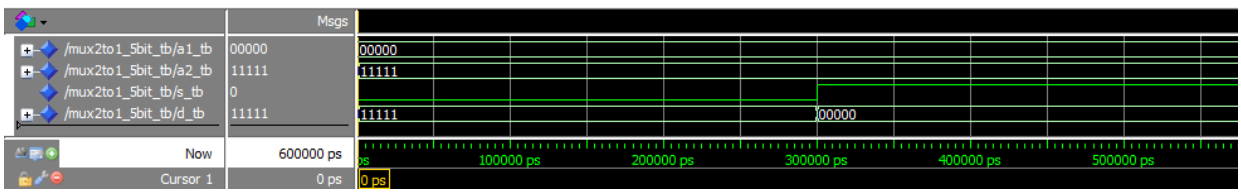
- Testbench

Στο testbench ελέγχεται η έξοδος για τις εισόδους 00000 και 11111 με select ίσο με 0 και 1.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux2to1_5bit_tb is
end mux2to1_5bit_tb;

architecture test of mux2to1_5bit_tb is

  -- mux2to1 5 bit
  component mux2to1_5bit
  port (
    a1: in std_logic_vector(4 downto 0);
    a2: in std_logic_vector(4 downto 0);
    s : in std_logic;
    d: out std_logic_vector(4 downto 0)
  );
  end component;

  -- inputs
  signal a1_tb, a2_tb: std_logic_vector(4 downto 0) := (others => '0');
  signal s_tb : std_logic := '0';

  -- outputs
```

```
  signal d_tb: std_logic_vector(4 downto 0);

begin
  U1: mux2to1_5bit port map (
    a1 => a1_tb,
    a2 => a2_tb,
    s => s_tb,
    d => d_tb);

  -- stimulus process
  process
    begin

      a1_tb <= "00000";
      a2_tb <= "11111";
      s_tb <= '0';
      wait for 300 ns;

      s_tb <= '1';
      wait for 300 ns;

  end process;
end test;
```



9. Μονάδα επέκτασης προσήμου 16-σε-32 (Sign Extender)

Το κύκλωμα sign_extender έχει μία είσοδο 16 bit και την μετατρέπει σε 32 bit ανάλογα με το πρόσημο της.

Sin (16 bit)	Sout (32 bit)
sin(15) = '0'	x“0000” & Sin
sin(15) = '1'	X"FFFF” & Sin

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_signed.all;

entity sign_extender is port (
  sin : in std_logic_vector(15 downto 0);
  sout: out std_logic_vector(31 downto 0));
end sign_extender;

architecture dataflow of sign_extender is
begin
  sout(31 downto 0) <= x"0000" & sin(15 downto 0) when sin(15) = '0' else
                       x"ffff" & sin(15 downto 0);
end dataflow;
```

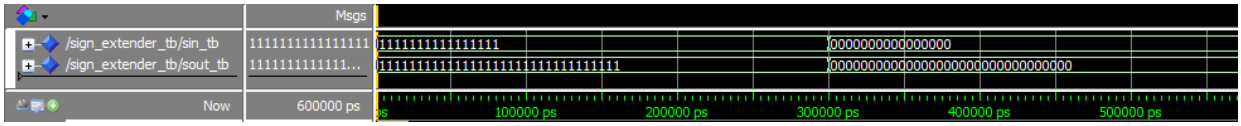
- Testbench

Στο testbench ελέγχεται η έξοδος για είσοδο με πρόσημοα ‘1’ και ‘0’.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sign_extender_tb is
end sign_extender_tb;

architecture test of sign_extender_tb is

  -- sign extender 32 bit
  component sign_extender
  port (
    sin : in std_logic_vector(15 downto 0);
    sout: out std_logic_vector(31 downto 0)
  );
  end component;

  -- input
  signal sin_tb: std_logic_vector(15 downto 0) := (others => '0');

  -- output
  signal sout_tb: std_logic_vector(31 downto 0);


begin
  U1: sign_extender port map (
```

```
    sin => sin_tb,
    sout => sout_tb);

  -- stimulus process
  process
    begin

      sin_tb <= x"FFFF";
      wait for 300 ns;

      sin_tb <= x"0000";
      wait for 300 ns;

  end process;
end test;
```



10. 32-πλός Πολυπλέκτης 2-σε-1

Το κύκλωμα `mux2to1_32bit` έχει δύο εισόδους 32 bit, μία είσοδο select και μία έξοδο 32 bit η οποία ανάλογα με την τιμή της select παίρνει την τιμή μίας εισόδου.

A1 (32 bit)	A2 (32 bit)	s	D (32 bit)
X"XXXXXXXX"	X"XXXXXXXX"	1	A1
X"XXXXXXXX"	X"XXXXXXXX"	0	A2

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux2to1_32bit is port (
  a1: in std_logic_vector(31 downto 0);
  a2: in std_logic_vector(31 downto 0);
  s : in std_logic;
  d: out std_logic_vector(31 downto 0)
  );
end mux2to1_32bit;

architecture dataflow of mux2to1_32bit is
begin
  d <= a1 when s = '1' else
       a2;
end dataflow;
```

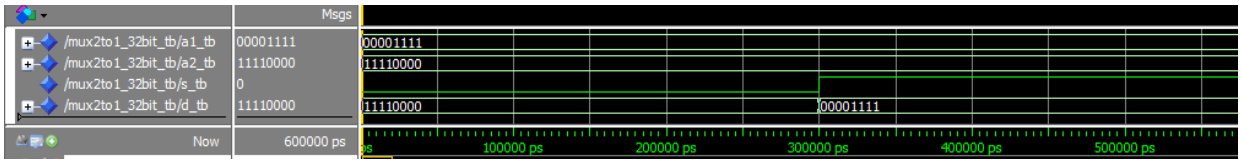
- Testbench

Στο testbench ελέγχεται η έξοδος για τις εισόδους x00001111 και x11110000 με select ίσο με 0 και 1.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux2to1_32bit_tb is
end mux2to1_32bit_tb;

architecture test of mux2to1_32bit_tb is

  -- mux2to1 32 bit
  component mux2to1_32bit
  port (
    a1: in std_logic_vector(31 downto 0);
    a2: in std_logic_vector(31 downto 0);
    s : in std_logic;
    d: out std_logic_vector(31 downto 0)
  );
  end component;

  -- inputs
  signal a1_tb, a2_tb: std_logic_vector(31 downto 0) := (others => '0');
  signal s_tb : std_logic := '0';
```

```
  -- outputs
  signal d_tb: std_logic_vector(31 downto 0);


begin
  U1: mux2to1_32bit port map (
    a1 => a1_tb,
    a2 => a2_tb,
    s => s_tb,
    d => d_tb);

  -- stimulus process
  process
    begin

      a1_tb <= x"00001111";
      a2_tb <= x"11110000";
      s_tb <= '0';
      wait for 300 ns;

      s_tb <= '1';
      wait for 300 ns;

  end process;
end test;
```


Msgs
/mux2to1_32bit_tb/a1_tb
/mux2to1_32bit_tb/a2_tb
/mux2to1_32bit_tb/s_tb
/mux2to1_32bit_tb/d_tb
00001111
11110000
0
11110000
11110000
00001111
Now
600000 ps
100000 ps
200000 ps
300000 ps
400000 ps
500000 ps

11. Μονάδα ολίσθησης αριστερά κατά 2 (32 bit) (Shift left)

Το κύκλωμα `shift_left2_32bit` έχει μία είσοδο 32 bit και ολισθαίνει την τιμή της αριστερά κατά 2 μηδενικά bit.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_signed.all;

entity shift_left2_32bit is port (
  sin : in std_logic_vector(31 downto 0);
  sout: out std_logic_vector(31 downto 0));
end shift_left2_32bit;

architecture dataflow of shift_left2_32bit is
begin
  sout(31 downto 0) <= sin(29 downto 0) & "00";
end dataflow;
```

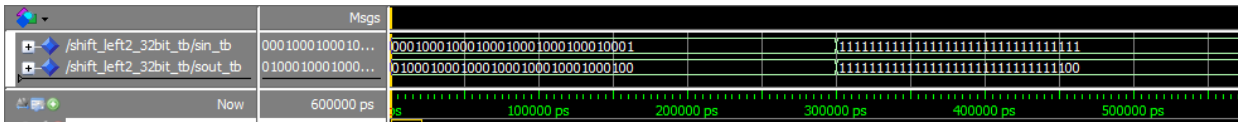
- Testbench

Στο testbench ελέγχεται η έξοδος για είσοδο x11111111 και xFFFFFFFF.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity shift_left2_32bit_tb is
end shift_left2_32bit_tb;

architecture test of shift_left2_32bit_tb is

  -- shift left by 2 32 bit
  component shift_left2_32bit
  port (
    sin : in std_logic_vector(31 downto 0);
    sout: out std_logic_vector(31 downto 0)
  );
  end component;

  -- input
  signal sin_tb: std_logic_vector(31 downto 0);

  -- output
  signal sout_tb: std_logic_vector(31 downto 0);


begin
  U1: shift_left2_32bit port map (
    sin => sin_tb,
    sout => sout_tb);
```

```
  -- stimulus process
  process
    begin

      sin_tb <= x"11111111";
      wait for 300 ns;

      sin_tb <= x"FFFFFFFF";
      wait for 300 ns;

  end process;
end test;
```



12. Αθροιστής 32 bit (Adder)

Το κύκλωμα adder_32bit έχει δύο εισόδους 32 bit και μία έξοδο η οποία είναι το άθροισμα τους.

A (32 bit)	B (32 bit)	Cout (32 bit)
X"XXXXXXXX"	X"XXXXXXXX"	A + B

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_signed.all;

entity adder_32bit is port (
  a, b : in std_logic_vector(31 downto 0);
  cout: out std_logic_vector(31 downto 0));
end adder_32bit;

architecture dataflow of adder_32bit is
begin
  cout(31 downto 0) <= a(31 downto 0) + b(31 downto 0);
end dataflow;
```

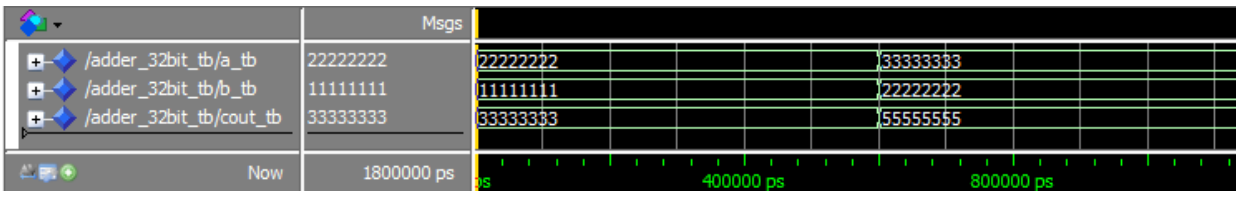
- Testbench

Στο testbench ελέγχεται η έξοδος για εισόδους με αριθμούς x22222222,x11111111 και x22222222, x33333333.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;

entity adder_32bit_tb is
end adder_32bit_tb;

architecture test of adder_32bit_tb is

  -- adder 32 bit
  component adder_32bit
  port (
    a, b : in std_logic_vector(31 downto 0);
    cout: out std_logic_vector(31 downto 0)
  );
  end component;

  -- inputs
  signal a_tb, b_tb: std_logic_vector(31 downto 0) := (others => '0');

  -- outputs
  signal cout_tb: std_logic_vector(31 downto 0);


begin
  U1: adder_32bit port map (
    a => a_tb,
```

```
    b => b_tb,
    cout => cout_tb);

  -- stimulus process
  process
    begin

      a_tb <= x"22222222";
      b_tb <= x"11111111";
      wait for 600 ns;

      a_tb <= x"33333333";
      b_tb <= x"22222222";
      wait for 600 ns;

  end process;
end test;
```



13.MIPS

Το κύκλωμα MIPS χρησιμοποιεί ως component όλα τα προηγούμενα κυκλώματα εκτός από τη μονάδα ολίσθησης αριστερά διότι το PC αυξάνεται κατά 1 και δεν χρειάζεται να γίνει πολλαπλασιασμός του offset. Τα κυκλώματα συνδυάζονται σύμφωνα με την πρώτη εικόνα.

Για την εντολή bne χρησιμοποιείται μια πύλη and με εισόδους το σήμα Branch και την άρνηση του zero.

- Κώδικας

```
library ieee;
use ieee.std_logic_1164.all;

entity MIPS is port (
-- MIPS inputs
reset : in std_logic;
clk : in std_logic
);
end MIPS;

architecture structural of MIPS is

  signal MIPS_clk : std_logic;
  signal MIPS_reset : std_logic;

  -- 1. ALU component
  component ALU
  port (
    D1 : in std_logic_vector(31 downto 0);
    D2 : in std_logic_vector(31 downto 0);
```

```
    operation : in std_logic_vector(3 downto 0);
    alu_res : out std_logic_vector(31 downto 0);
    zero : out std_logic); -- zero
  end component;

  -- ALU inputs and outputs
  signal ALU_D1 : std_logic_vector(31 downto 0);
  signal ALU_D2 : std_logic_vector(31 downto 0);
  signal ALU_alu_res : std_logic_vector(31 downto 0);
  signal ALU_zero : std_logic;

  -- 2. Register file component
  component register_file
  port (
    rAddr1: in std_logic_vector(4 downto 0);
rAddr2: in std_logic_vector(4 downto 0);
wAddr : in std_logic_vector(4 downto 0);
    wD : in std_logic_vector(31 downto 0);
RegWrite : in std_logic;
clk : in std_logic;
reset : in std_logic;
rD1 : out std_logic_vector(31 downto 0);
rD2 : out std_logic_vector(31 downto 0));
end component;

-- Register file inputs and outputs
signal RF_rAddr1 : std_logic_vector(4 downto 0);
signal RF_rAddr2 : std_logic_vector(4 downto 0);
signal RF_wAddr : std_logic_vector(4 downto 0);
signal RF_wD : std_logic_vector(31 downto 0);
```

```
signal RF_rD2 : std_logic_vector(31 downto 0);

-- 3. Data memory component
component data_memory
  port (
    address : in std_logic_vector(31 downto 0);
    write_data : in std_logic_vector(31 downto 0);
  clk : in std_logic;
  reset : in std_logic;
    MemWrite : in std_logic;
    MemRead : in std_logic;
    read_data : out std_logic_vector(31 downto 0));
  end component;

  -- Data memory output
  signal DM_read_data : std_logic_vector(31 downto 0);

  -- 4. Instruction memory component
  component instruction_memory
  port (
    read_address : in std_logic_vector(31 downto 0);
  clk : in std_logic;
  reset: in std_logic;
  instruction : out std_logic_vector(31 downto 0));
  end component;

  -- Instruction memory output
  signal IM_instruction : std_logic_vector(31 downto 0);

  -- 5. Control unit component
```

```
  component control_unit
  port (
     opcode : in std_logic_vector(5 downto 0);
   RegDst : out std_logic;
   Branch : out std_logic;
   MemRead : out std_logic;
   MemtoReg : out std_logic;
   ALUOp : out std_logic_vector(1 downto 0);
   MemWrite : out std_logic;
   ALUSrc : out std_logic;
   RegWrite : out std_logic);
 end component;

 -- Control unit input and outputs
signal CU_opcode : std_logic_vector(5 downto 0);
signal CU_RegDst : std_logic;
signal CU_Branch : std_logic;
signal CU_MemRead : std_logic;
signal CU_MemtoReg : std_logic;
signal CU_ALUOp : std_logic_vector(1 downto 0);
signal CU_MemWrite : std_logic;
signal CU_ALUSrc : std_logic;
signal CU_RegWrite : std_logic;

-- 6. ALU Control unit component
component ALU_control
  port (
    ALUop : in std_logic_vector(1 downto 0);
    funct : in std_logic_vector(5 downto 0);
    operation : out std_logic_vector(3 downto 0));
```

```
end component;

    -- ALU Control unit output
signal ALUcu_operation : std_logic_vector(3 downto 0);

-- 7. PC component
component PC
port (
  PCin : in std_logic_vector(31 downto 0);
reset : in std_logic;
  clk : in std_logic;
  PCout : out std_logic_vector(31 downto 0));
end component;

-- PC inputs and output
signal PC_PCin : std_logic_vector(31 downto 0);
signal PC_PCout : std_logic_vector(31 downto 0);

-- 8. Mux 2to1 5 bit component
component mux2to1_5bit
port (
  a1: in std_logic_vector(4 downto 0);
  a2: in std_logic_vector(4 downto 0);
  s : in std_logic;
  d: out std_logic_vector(4 downto 0));
end component;

-- 9. Sign extender component
component sign_extender
port (
```

```
  sin : in std_logic_vector(15 downto 0);
  sout: out std_logic_vector(31 downto 0));
end component;

-- Sign extender output
signal SE_sout: std_logic_vector(31 downto 0);

-- 10, 11, 12. Mux 2to1 32 bit component
component mux2to1_32bit
port (
  a1: in std_logic_vector(31 downto 0);
  a2: in std_logic_vector(31 downto 0);
  s : in std_logic;
  d: out std_logic_vector(31 downto 0));
end component;

-- 12. Mux 2to1 32 bit output
signal Mux2to132b_2s : std_logic;

-- 13. Shift left by 2 component
component shift_left2_32bit
port (
  sin : in std_logic_vector(31 downto 0);
  sout: out std_logic_vector(31 downto 0));
end component;

-- Shift left by 2 output
-- signal SL_sout: std_logic_vector(31 downto 0);

-- 14, 15. Adder component
```

```
  component adder_32bit
  port (
    a, b : in std_logic_vector(31 downto 0);
    cout: out std_logic_vector(31 downto 0));
  end component;

  -- Adder outputs
  signal Add_cout: std_logic_vector(31 downto 0);
  signal Add1_cout: std_logic_vector(31 downto 0);

begin

  MIPS_clk <= clk;
  MIPS_reset <= reset;

  -- ALU
  U1: ALU port map (
    D1 => ALU_D1,
    D2 => ALU_D2,
    operation => ALUcu_operation,
    alu_res => ALU_alu_res,
    zero => ALU_zero);

  -- Register file
  U2: register_file port map (
      rAddr1 => RF_rAddr1,
      rAddr2 => RF_rAddr2,
      wAddr => RF_wAddr,
      wD => RF_wD,
      RegWrite => CU_RegWrite,
```

```
      clk => MIPS_clk,
      reset => MIPS_reset,
      rD1 => ALU_D1,
      rD2 => RF_rD2);

      RF_rAddr1 <= IM_instruction(25 downto 21);
      RF_rAddr2 <= IM_instruction(20 downto 16);

-- Data memory
U3: data_memory port map (
    address  => ALU_alu_res,
    write_data => RF_rD2,
    clk => MIPS_clk,
    reset => MIPS_reset,
    MemWrite => CU_MemWrite,
    MemRead => CU_MemRead,
    read_data => DM_read_data);

  -- Instruction memory
  U4: instruction_memory port map (
    read_address => PC_PCout,
    clk => MIPS_clk,
    reset => MIPS_reset,
    instruction => IM_instruction);

  -- Control unit
  U5: control_unit PORT MAP (
      opcode => CU_opcode,
      RegDst => CU_RegDst,
      Branch => CU_Branch,
```

```
      MemRead => CU_MemRead,
      MemtoReg => CU_MemtoReg,
      ALUOp => CU_ALUOp,
      MemWrite => CU_MemWrite,
      ALUSrc => CU_ALUSrc,
      RegWrite => CU_RegWrite);

      CU_opcode <= IM_instruction(31 downto 26);

-- ALU Control unit
U6: ALU_control port map (
    ALUop => CU_ALUOp,
    funct => IM_instruction(5 downto 0),
    operation => ALUcu_operation);

  -- PC
  U7: PC port map (
    PCin => PC_PCin,
    reset => MIPS_reset,
    clk => MIPS_clk,
    PCout => PC_PCout);

  -- Register destination Mux 2to1 5 bit
  U8: mux2to1_5bit port map (
    a1 => IM_instruction(15 downto 11),
    a2 => IM_instruction(20 downto 16),
    s => CU_RegDst, -- if 0 then a2 else a1
    d => RF_wAddr);

  -- Sign extender
```

```
U9: sign_extender port map (
  sin => IM_instruction(15 downto 0),
  sout => SE_sout);

-- Muxs 2to1 32 bit
-- ALU source mux
U10: mux2to1_32bit port map (
  a1 => SE_sout,
  a2 => RF_rD2,
  s => CU_ALUSrc,  -- if 0 then a2 else a1
  d => ALU_D2);

-- Memory to Register file mux
U11: mux2to1_32bit port map (
  a1 => DM_read_data,
  a2 => ALU_alu_res,
  s => CU_MemtoReg,  -- if 0 then a2 else a1
  d => RF_wD);

-- Branch mux
U12: mux2to1_32bit port map (
  a1 => Add_cout,
  a2 => Add1_cout,
  s => Mux2to132b_2s, -- if 0 then a2 else a1
  d => PC_PCin);

-- And
Mux2to132b_2s <= (not ALU_zero) and CU_Branch;
```

```
--  -- Shift left by 2
--  U13: shift_left2_32bit port map (
--    sin => SE_sout,
--    sout => SL_sout);

  -- Adders
  U14: adder_32bit port map (
    a => PC_PCout,
    b => x"00000001",
    cout => Add1_cout);

  U15: adder_32bit port map (
    a => Add1_cout,
    b => SE_sout,
    cout => Add_cout);

end structural;
```

- Testbench

Το testbench έχει ρολόι περιόδου 10 ns. Για τον έλεγχο πρώτα γίνεται reset για να μηδενιστούν τα δεδομένα των καταχωρητών και στη συνέχεια εκτελείται το πρόγραμμα. Ο χρόνος ο οποίος χρειάζεται για να εκτελεσθούν οι εντολές πριν την εντολή bne είναι 220 ns και στη συνέχεια το πρόγραμμα εκτελείται για 50 ns. Άρα συνολικά ο χρόνος εκτέλεσης είναι 280 ns.

- Κώδικας

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity MIPS_tb is
end MIPS_tb;

architecture test of MIPS_tb is

  component MIPS
  port (
    reset : in std_logic;
  clk : in std_logic
);
  end component;

  -- inputs
  signal reset_tb : std_logic;
  signal clk_tb : std_logic;


begin
  U1: MIPS port map (
    reset => reset_tb,
    clk => clk_tb);

  -- clock process
clk_p: process
begin
clk_tb <= '1';
wait for 5 ns;
clk_tb <= '0';
wait for 5 ns;
```

```
end process clk_p;

  -- stimulus process
  process
    begin

      reset_tb <= '1';
      wait for 10 ns;

      reset_tb <= '0';
      wait for 220 ns;

      -- branch not equal
      wait for 50 ns;

      -- runs for 280 ns

  end process;
end test;
```

- Αποτέλεσμα

Στην εκτέλεση του testbench τα σήματα τα οποία εμφανίζονται είναι το reset, το ρολόι, η τρέχουσα εντολή, όλες οι θέσεις μνήμης του register file και οι θέσεις 0 έως 2 της μνήμης δεδομένων. Παρατηρούμε ότι το πρόγραμμα αποθηκεύει στους καταχωρητές 3 και 5 τους αριθμούς 1 και 3 αντίστοιχα. Στη συνέχεια αρχίζει η επανάληψη όπου στον καταχωρητή 6 αποθηκεύεται το περιεχόμενο του 3 και στη θέση της μνήμης δεδομένων με αριθμό το περιεχόμενο του καταχωρητή 4 αποθηκεύεται το περιεχόμενο του καταχωρητή 6. Στη συνέχεια της επανάληψης προθέτονται στους καταχωρητές 3, 4, και 5 οι αριθμοί 1, 1 και -1 αντίστοιχα. Αν ο καταχωρητής 5 δεν έχει τιμή ίση με 0 τότε συνεχίζεται η επανάληψη.

Στη πρώτη επανάληψη στο καταχωρητή 6 αποθηκεύεται ο αριθμός 1 και στη θεση 0 της μνήμης δεδομένων αποθηκεύεται το 1. Μετά από τις προσθέσεις οι καταχωρητές 3, 4 και 5 έχουν τις τιμές 2, 1

και 2. Ο καταχωρητής 5 δεν έχει τιμή ίση με 0 οπότε συνεχίζεται η επανάληψη. Στη δεύτερη επανάληψη στο καταχωρητή 6 αποθηκεύεται ο αριθμός 2 και στη θεση 1 της μνήμης δεδομένων αποθηκεύεται το 2. Μετά από τις προσθέσεις οι καταχωρητές 3, 4 και 5 έχουν τις τιμές 3, 2 και 1. Ο καταχωρητής 5 δεν έχει τιμή ίση με 0 οπότε συνεχίζεται η επανάληψη. Στη πρώτη τρίτη στο καταχωρητή 6 αποθηκεύεται ο αριθμός 3 και στη θεση 2 της μνήμης δεδομένων αποθηκεύεται το 3. Μετά από τις προσθέσεις οι καταχωρητές 3, 4 και 5 έχουν τις τιμές 4, 3 και 0. Ο καταχωρητής 5 έχει τιμή ίση με 0 οπότε τελειώνει η επανάληψη και τερματίζεται το πρόγραμμα.