Σχεδίαση Ψηφιακών Συστημάτων - Εργασία Θεωρίας (Μέρος 7)

Χρήστος Μαργιώλης

Ιούλιος 2020

Περιεχόμενα

1 Κώδικας και τεκμηρίωση **1**
1.1 alu.vhd . . . 1
1.2 regfile_ext.vhd . . . 3
1.3 instrmem.vhd . . . 5
1.4 ctrl.vhd . . . 7
1.5 alu_ctrl.vhd . . . 8
1.6 adder32.vhd . . . 9
1.7 pc.vhd . . . 10
1.8 mips.vhd . . . 11
1.9 mips_tb.vhd . . . 15

2 Εκτέλεση **17**
2.1 mips_tb . . . 17

1 Κώδικας και τεκμηρίωση

Για καλύτερη κατανόηση του κώδικα του MIPS, παραθέτω και τις υλοποιήσεις (ξανά) όλων των κυκλωμάτων που χρησιμοποιήθηκαν. Μερικά από τα κυκλώματα υπέστησαν μικρές τροποποιήσεις.

1.1 alu.vhd

Αριθμητική και λογική μονάδα.

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

entity alu is
generic (
        sz:             natural := 32
);
port (
        alu_in1:        in std_logic_vector(sz-1 downto 0);
        alu_in2:        in std_logic_vector(sz-1 downto 0);
        alu_ctrl:       in std_logic_vector(3 downto 0);
        alu_out:        out std_logic_vector(sz-1 downto 0);
        alu_zero:       out std_logic
);
end alu;

architecture behav of alu is
signal sig:             std_logic_vector(sz-1 downto 0);

begin
```

```
        process (alu_ctrl) begin
                case alu_ctrl is
                when "0000" => sig <= alu_in1 and alu_in2;
                when "0001" => sig <= alu_in1 or alu_in2;
                when "0010" =>
                        sig <= std_logic_vector(signed(alu_in1) + signed(alu_in2));
                when "0110" =>
                        sig <= std_logic_vector(signed(alu_in1) - signed(alu_in2));
                when others => sig <= (others => 'X');
                end case;
        end process;
        alu_zero <= '1' when sig = x"00000000" else '0';
        alu_out <= sig;
end behav;
```

1.2 regfile_ext.vhd

Register file.

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

entity regfile_ext is
generic (
        sz:             natural := 32;
        addrw:          natural := 5
);
port (
        idata:          in std_logic_vector(sz-1 downto 0);
        raddr1:         in std_logic_vector(addrw-1 downto 0);
        raddr2:         in std_logic_vector(addrw-1 downto 0);
        waddr:          in std_logic_vector(addrw-1 downto 0);
        we:             in std_logic;
        clk:            in std_logic;
        rst:            in std_logic;
        odata1:         out std_logic_vector(sz-1 downto 0);
        odata2:         out std_logic_vector(sz-1 downto 0)
);
end regfile_ext;

architecture behav of regfile_ext is

signal arrsz:           natural := 32;
type regarr             is array(0 to arrsz-1) of std_logic_vector(sz-1 downto 0);
-- Array used for initialization when rst = 1.
signal s_init:          regarr := (others => x"ffffffff");
signal regf:            regarr;

begin
        process (clk) begin
                if (rst = '1') then
                        regf <= s_init;
                elsif (clk'event and clk = '0') then
                        if (we = '1') then
                                regf(to_integer(unsigned(waddr))) <= idata;
                        end if;
                end if;
        end process;
        odata1 <= regf(to_integer(unsigned(raddr1)));
```

```
        odata2 <= regf(to_integer(unsigned(raddr2)));
end behav;
```

1.3 instrmem.vhd

Μνήμη εντολών.

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

entity instrmem is port (
        addr:   in std_logic_vector(31 downto 0);
        c:      out std_logic_vector(31 downto 0)
);
end instrmem;

architecture dataflow of instrmem is

type instr_arr is array(0 to 31) of std_logic_vector (31 downto 0);
constant instr_mem: instr_arr := (
        "00000000010001100010000000100000", -- 0 add $4, $2, $6
        "00000000010001100010100000100010", -- 1 sub $5, $2, $6
        "11111100100001010000000000000000", -- 2 read $4, $5
        "00000000000000000000000000000000", -- 3
        "00000000000000000000000000000000", -- 4
        "00000000000000000000000000000000", -- 5
        "00000000000000000000000000000000", -- 6
        "00000000000000000000000000000000", -- 7
        "00000000000000000000000000000000", -- 8
        "00000000000000000000000000000000", -- 9
        "00000000000000000000000000000000", -- 10
        "00000000000000000000000000000000", -- 11
        "00000000000000000000000000000000", -- 12
        "00000000000000000000000000000000", -- 13
        "00000000000000000000000000000000", -- 14
        "00000000000000000000000000000000", -- 15
        "00000000000000000000000000000000", -- 16
        "00000000000000000000000000000000", -- 17
        "00000000000000000000000000000000", -- 18
        "00000000000000000000000000000000", -- 19
        "00000000000000000000000000000000", -- 20
        "00000000000000000000000000000000", -- 21
        "00000000000000000000000000000000", -- 22
        "00000000000000000000000000000000", -- 23
        "00000000000000000000000000000000", -- 24
        "00000000000000000000000000000000", -- 25
        "00000000000000000000000000000000", -- 26
```

```
        "00000000000000000000000000000000", -- 27
        "00000000000000000000000000000000", -- 28
        "00000000000000000000000000000000", -- 29
        "00000000000000000000000000000000", -- 30
        "00000000000000000000000000000000"  -- 31
);

begin
        -- Our addresses are multiples of 4, so ignore the last 2 bits.
        c <= instr_mem(to_integer(unsigned(addr(31 downto 2))));
end dataflow;
```

1.4 ctrl.vhd

Μονάδα ελέγχου.

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

entity ctrl is port (
        funct:          in std_logic_vector(5 downto 0);
        reg_dst:        out std_logic;
        reg_wr:         out std_logic;
        alu_src:        out std_logic;
        branch:         out std_logic;
        mem_rd:         out std_logic;
        mem_wr:         out std_logic;
        mem_toreg:      out std_logic;
        alu_op:         out std_logic_vector(1 downto 0)
);
end ctrl;

architecture dataflow of ctrl is
begin
        reg_dst         <= '1' when funct = "000000" else '0';
        reg_wr          <= '1' when funct = "000000" or funct = "100011" else '0';
        alu_src         <= '1' when funct = "100011" or funct = "101011" else '0';
        branch          <= '1' when funct = "000100" else '0';
        mem_rd          <= '1' when funct = "100011" else '0';
        mem_wr          <= '1' when funct = "101011" else '0';
        mem_toreg       <= '1' when funct = "100011" else '0';
        with funct select
                alu_op <= "10" when "000000",
                          "01" when "000100",
                          "00" when others;
end dataflow;
```

1.5 alu_ctrl.vhd

Μονάδα ελέγχου ALU.

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

entity alu_ctrl is port (
        funct:          in std_logic_vector(5 downto 0);
        alu_op:         in std_logic_vector(1 downto 0);
        op:             out std_logic_vector(3 downto 0)
);
end alu_ctrl;

architecture dataflow of alu_ctrl is
begin
        op <= "0010" when (alu_op = "00" or (alu_op = "10" and funct = "100000")) else
              "0110" when (alu_op = "01" or (alu_op = "10" and funct = "100010")) else
              "0000" when (alu_op = "10" and funct = "100100") else
              "0001" when (alu_op = "10" and funct = "100101") else
              "0111" when (alu_op = "10" and funct = "101010") else
              "1111";
end dataflow;
```

1.6 adder32.vhd

Αθροιστής 32-bit.

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

entity adder32 is port (
        x:      in std_logic_vector(31 downto 0);
        y:      in std_logic_vector(31 downto 0);
        z:      out std_logic_vector(31 downto 0)
);
end adder32;

architecture dataflow of adder32 is
begin
        z <= std_logic_vector(unsigned(x) + unsigned(y));
end dataflow;
```

1.7 pc.vhd

Program counter.

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

entity pc is port (
        clk:    in std_logic;
        rst:    in std_logic;
        ipc:    in std_logic_vector(31 downto 0);
        opc:    out std_logic_vector(31 downto 0)
);
end pc;

architecture behav of pc is
begin
        process (clk, rst) begin
                if (rst = '1') then
                        opc <= x"00000000";
                elsif (clk'event and clk = '0') then
                        opc <= ipc;
                end if;
        end process;
end behav;
```

1.8 mips.vhd

'Οπως λέει η εκφώνηση, ο MIPS έχει μόνο δύο εισόδους: τον ορολογιακό παλμό και το σήμα reset, και δεν έχει εξόδους. Στην αρχιτεκτονική του δηλώνουμε ως components όλα τα παραπάνω κυκλώματα. 'Επειτα δημιουργούμε βοηθητικά σήματα για το πέρασμα αποτελεσμάτων από το ένα κύκλωμα στο άλλο.

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

entity mips is port (
        m_clk:          in std_logic;
        m_rst:          in std_logic
);
end mips;

architecture struct of mips is

component alu is
generic (
        sz:             natural := 32
);
port (
        alu_in1:        in std_logic_vector(sz-1 downto 0);
        alu_in2:        in std_logic_vector(sz-1 downto 0);
        alu_ctrl:       in std_logic_vector(3 downto 0);
        alu_out:        out std_logic_vector(sz-1 downto 0);
        alu_zero:       out std_logic
);
end component;

component regfile_ext is
generic (
        sz:             natural := 32;
        addrw:          natural := 5
);
port (
        idata:          in std_logic_vector(sz-1 downto 0);
        raddr1:         in std_logic_vector(addrw-1 downto 0);
        raddr2:         in std_logic_vector(addrw-1 downto 0);
        waddr:          in std_logic_vector(addrw-1 downto 0);
        we:             in std_logic;
        clk:            in std_logic;
        rst:            in std_logic;
        odata1:         out std_logic_vector(sz-1 downto 0);
        odata2:         out std_logic_vector(sz-1 downto 0)
);
```

```
end component;

component instrmem is port (
        addr:           in std_logic_vector(31 downto 0);
        c:              out std_logic_vector(31 downto 0)
);
end component;

component ctrl is port (
        funct:          in std_logic_vector(5 downto 0);
        reg_dst:        out std_logic;
        reg_wr:         out std_logic;
        alu_src:        out std_logic;
        branch:         out std_logic;
        mem_rd:         out std_logic;
        mem_wr:         out std_logic;
        mem_toreg:      out std_logic;
        alu_op:         out std_logic_vector(1 downto 0)
);
end component;

component alu_ctrl is port (
        funct:          in std_logic_vector(5 downto 0);
        alu_op:         in std_logic_vector(1 downto 0);
        op:             out std_logic_vector(3 downto 0)
);
end component;

component adder32 is port (
        x:              in std_logic_vector(31 downto 0);
        y:              in std_logic_vector(31 downto 0);
        z:              out std_logic_vector(31 downto 0)
);
end component;

component pc is port (
        clk:            in std_logic;
        rst:            in std_logic;
        ipc:            in std_logic_vector(31 downto 0);
        opc:            out std_logic_vector(31 downto 0)
);
end component;

constant c_pc_add_val:  std_logic_vector(31 downto 0) := x"00000004";
signal s_adder_out:     std_logic_vector(31 downto 0);
signal s_pc_out:        std_logic_vector(31 downto 0);
```

```
signal s_alu_out:       std_logic_vector(31 downto 0);
signal s_reg_dst:       std_logic;
signal s_reg_wr:        std_logic;
signal s_alu_src:       std_logic;
signal s_branch:        std_logic;
signal s_mem_rd:        std_logic;
signal s_mem_wr:        std_logic;
signal s_mem_toreg:     std_logic;
signal s_alu_op:        std_logic_vector(1 downto 0);
signal s_op:            std_logic_vector(3 downto 0);
signal s_reg_out1:      std_logic_vector(31 downto 0);
signal s_reg_out2:      std_logic_vector(31 downto 0);
signal s_alu_zero:      std_logic;
signal s_instr:         std_logic_vector(31 downto 0);

begin
        alu_map: alu port map (
                alu_in1 => s_reg_out1,
                alu_in2 => s_reg_out2,
                alu_ctrl => s_op,
                alu_out => s_alu_out,
                alu_zero => s_alu_zero
        );

        regfile_ext_map: regfile_ext port map (
                idata => s_alu_out,
                raddr1 => s_instr(25 downto 21),
                raddr2 => s_instr(20 downto 16),
                waddr => s_instr(15 downto 11),
                we => s_reg_wr,
                clk => m_clk,
                rst => m_rst,
                odata1 => s_reg_out1,
                odata2 => s_reg_out2
        );

        instrmem_map: instrmem port map (
                addr => s_pc_out,
                c => s_instr
        );

        ctrl_map: ctrl port map (
                funct => s_instr(31 downto 26),
                reg_dst => s_reg_dst,
                reg_wr => s_reg_wr,
                alu_src => s_alu_src,
```

```
                branch => s_branch,
                mem_rd => s_mem_rd,
                mem_wr => s_mem_wr,
                mem_toreg => s_mem_toreg,
                alu_op => s_alu_op
        );

        alu_ctrl_map: alu_ctrl port map (
                funct => s_instr(5 downto 0),
                alu_op => s_alu_op,
                op => s_op
        );

        adder32_map: adder32 port map (
                x => s_pc_out,
                y => c_pc_add_val,
                z => s_adder_out
        );

        pc_map: pc port map (
                clk => m_clk,
                rst => m_rst,
                ipc => s_adder_out,
                opc => s_pc_out
        );
end struct;
```

1.9 mips_tb.vhd

Testbench για δοκιμή του MIPS.

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

entity mips_tb is
end mips_tb;

architecture behav of mips_tb is

signal s_m_clk:         std_logic;
signal s_m_rst:         std_logic;

component mips is port (
        m_clk:          in std_logic;
        m_rst:          in std_logic
);
end component;

begin
        uut: mips port map (
                m_clk => s_m_clk,
                m_rst => s_m_rst
        );

        process begin
                -- Reset everything.
                s_m_rst <= '1';
                s_m_clk <= '0';
                wait for 250 ns;

                s_m_clk <= '1';
                wait for 250 ns;

                -- 1st cycle.
                s_m_rst <= '0';
                s_m_clk <= '0';
                wait for 250 ns;

                s_m_clk <= '1';
                wait for 250 ns;

                -- 2nd cycle.
                s_m_clk <= '0';
```

```
                wait for 250 ns;

                s_m_clk <= '1';
                wait for 250 ns;

                -- 3rd cycle.
                s_m_clk <= '0';
                wait for 250 ns;

                s_m_clk <= '1';
                wait for 250 ns;
        end process;
end behav;
```

2 Εκτέλεση

2.1 mips_tb

Στην εκφώνηση της άσκησης ζητήθηκε ο MIPS να έχει μόνο τον ορολογιακό παλμό και το reset ως εισόδους, και όχι εξόδο. Παρ'όλα αυτά για την καλύτερη κατανόηση του κυκλώματος, έβαλα επιπλέον σήματα στο κύκλωμα (δεν περιλαμβάνονται στον κώδικα πια εφόσον δεν ζητούνται).