контрольная 2

```
entity ...
    port (
        clk, rst : in std_logic;
        input_port16 : in std_logic_vector (15 downto 0);
        input_port24 : in std_logic_vector (23 downto 0);
        output_port16 : out std_logic_vector (15 downto 0);
        output_port24 : out std_logic_vector (23 downto 0)
    );
end ...

architecture ...
    type state_type is ( Fetch, ... );
    signal state : state_type := Fetch;

    signal pc : std_logic_vector (7 downto 0) := (others => '0');

    signal instr : std_logic_vector (15 downto 0) := (others => '0');
    signal opcode : std_logic_vector (7 downto 0) := (others => '0');
    signal operand : std_logic_vector (7 downto 0) := (others => '0');

    signal regA : std_logic_vector (15 downto 0) := (others => '0');
        regB
        :
        ;

    signal reg24 : std_logic_vector (23 downto 0) := (others => '0');
```

```
signal regTMP24 : std_logic_vector (23 downto 0) := (others => '0');
signal regTMP16 : std_logic_vector (15 downto 0) := (others => '0');

type memory_type is array (0 to 255) of std_logic_vector(15
                                                          downto 0);

signal ROM : memory_type := (
   0 => "00 ... 00",    // 16-bit
);

signal RAM : memory_type := (
   0 => "00 ... 00",    // 16-bit
);

begin
      process (clk, rst)
      begin

         if rst = '0' then
            pc <= (others '0');
            instr <= -//-
            opcode
            operand
            // registers A-D, 24, 16, 32 било мој чао!
              ⋮
           state <= Fetch;
        elsif rising_edge(clk) then
           case state is
```

```
when Fetch =>
    instr <= ROM(pc);
    state <= Decode;

when Decode =>
    opcode <= instr(15 downto 8);
    operand <= instr(7 downto 0);
    state <= Execute;

when Execute =>
    case opcode is
        when "00000000" =>        -- NOP
            null;
        when "00000001" =>        -- LD A
            regA <= RAM(operand);
        when "00000010" =>        -- LD B
            regB <= RAM(operand);
        when "00000011" =>        -- LD C
            regC <= RAM(operand);
        when "00000100" =>        -- LD D
            regD <= RAM(operand);

        when "00000101" =>        -- MVI AL
            regA(7 downto 0) <= operand;
        when "00000110" =>        -- MVI BL
            regB(7 downto 0) <= operand;
        when "00000111" =>        -- MVI CL
            regC(7 downto 0) <= operand;
```

load од RAM меморија до 16-bit регистри

Внес до регистер вредност операнд 8-bit (до LSB бајт)

```
when "0000 1000" =>        -- MVI DL
    regD(7 downto 0) <= operand;
```

Внес во регистар вредност операнд 8-бит (во MSB бајт)

```
when "0000 1001" =>        -- MVI AH
    regA(15 downto 8) <= operand;
when "0000 1010" =>        -- MVI BH
    regB(15 downto 8) <= operand;
when "0000 1011" =>        -- MVI CH
    regC(15 downto 8) <= operand;

when "0000 1100" =>        -- MVI DH
    regD(15 downto 8) <= operand;
```

Влез и излез 16-бит порти

```
when "0000 1111" =>        -- OUT 16
    output_port16 <= reg A;
when "0001 0000" =>        -- IN 16
    reg A <= input_port16;
```

Влез и излез 24-бит порти

```
when "0001 0001" =>        -- OUT 24
    output_port24 <= reg24;
when "0001 0010" =>        -- IN 24
    reg24 <= input_port24;
```

```
when "0001 0010" =>        -- OUT 24L
-- од AL, BL, CL на output_port24
output_port24(23 downto 16) <= regA (7 downto 0);
output_port24 (15 downto 8) <= regB (7 downto 0);
output_port24 (7 downto 0) <= regC (7 downto 0);
```

```
when "0001 0011" =>      -- IN 24L
  -- og input_port24 ma AL, BL, CL
     regA(7 downto 0) <= input_port24(23 downto 16);
     regB(7 downto 0) <= input_port24(15 downto 8);
     regC(7 downto 0) <= input_port24(7 downto 0);

when "0001" =>      -- OUT 24
  -- og AH, BH, CH ma output_port24
  output_port24(23 downto 16) <= regA(15 downto 8);
  output_port24(15 downto 8)  <= regB(15 downto 8);
  output_port24(7 downto 0)   <= regC(15 downto 8);

when "0001" =>      -- IN 24
  -- og input_port24 ma AH, BH, CH
     regA(15 downto 8) <= input_port24(23 downto 16);
     regB(15 downto 8) <= input_port24(15 downto 8);
     regC(15 downto 8) <= input_port24(7 downto 0);

when "0001 0111" =>      -- BITREV16
  -- ABCDEFGH -> HGFEDCAB
  regTMP16 <= regA;
  for i in 0 to 15 loop
      regA(i) <= regTMP16(15-i);
  end loop;

when "0001 1000" =>      -- BITREV24
  -- ABCDEFGH -> HGFEDCBA
  regTMP24 <= reg;
  for i in 0 to 23 loop
      reg24 <= regTMP24(23-i).
```

```
        end loop;

    when "0001 1001" => -- RGB 888 TO 565 // TO 565
        regA(15 downto 11) <= reg24(23 downto 19); -- R
        regA(10 downto 5) <= reg24(15 downto 10); -- G
        regA(4 downto 0) <= reg24(7 downto 3); -- B

    when "0001 1010" => -- RGB 565 TO 888 // TO 888
        reg24(23 downto 16) <= regA(15 downto 11) & regA(15 downto 13);
        reg24(15 downto 8) <= regA(10 downto 5) & regA(10 downto 9);
        reg24(7 downto 0) <= regA(4 downto 0) & regA(4 downto 2);
            ⋮
    when "1111 1111" => -- HLT
            state <= Execute;
    when others =>
            null;
  end case;

  if opcode /= "1111 1111" then
        state <= Write_back;
  end if;
when WriteBack =>
```

// ako ima alu_result ili tomorrow ACC, voraiu Go neto to oneaniy6au A (regA) 6o cenoja ahstu kaj uuno ce kopuaw (so if!)

```
  pc <= pc + 1;
  state <= Fetch;
 end case;
end if;
```

end process;

end Behavioral;

IN 24	0001 0010
BIT REV 24	0001 0111
TO 565	0001 1001
BITREV 16	00010111
OUT 16	00001111
HLT	1111 1111

```
signal RAM : memory_type := (
  0 => "0001 0010 0000 0000",
  :
  5 => "1111 1111 0000 0000",
  others => (others => '0')
);
```

```
signal RAM : memory_type := (
```

MVI AL	125	0 => "0000 0101	125 → $7D_{16}$
MVI BL	65	1 => "0000 0110	65 → 41_{16}"
MVI CL	25	2 => "0000 0111	25 → 19_{16}"
OUT 24L		3 =>	
HLT		4 =>	

```
  others => (others => '0')
);
```