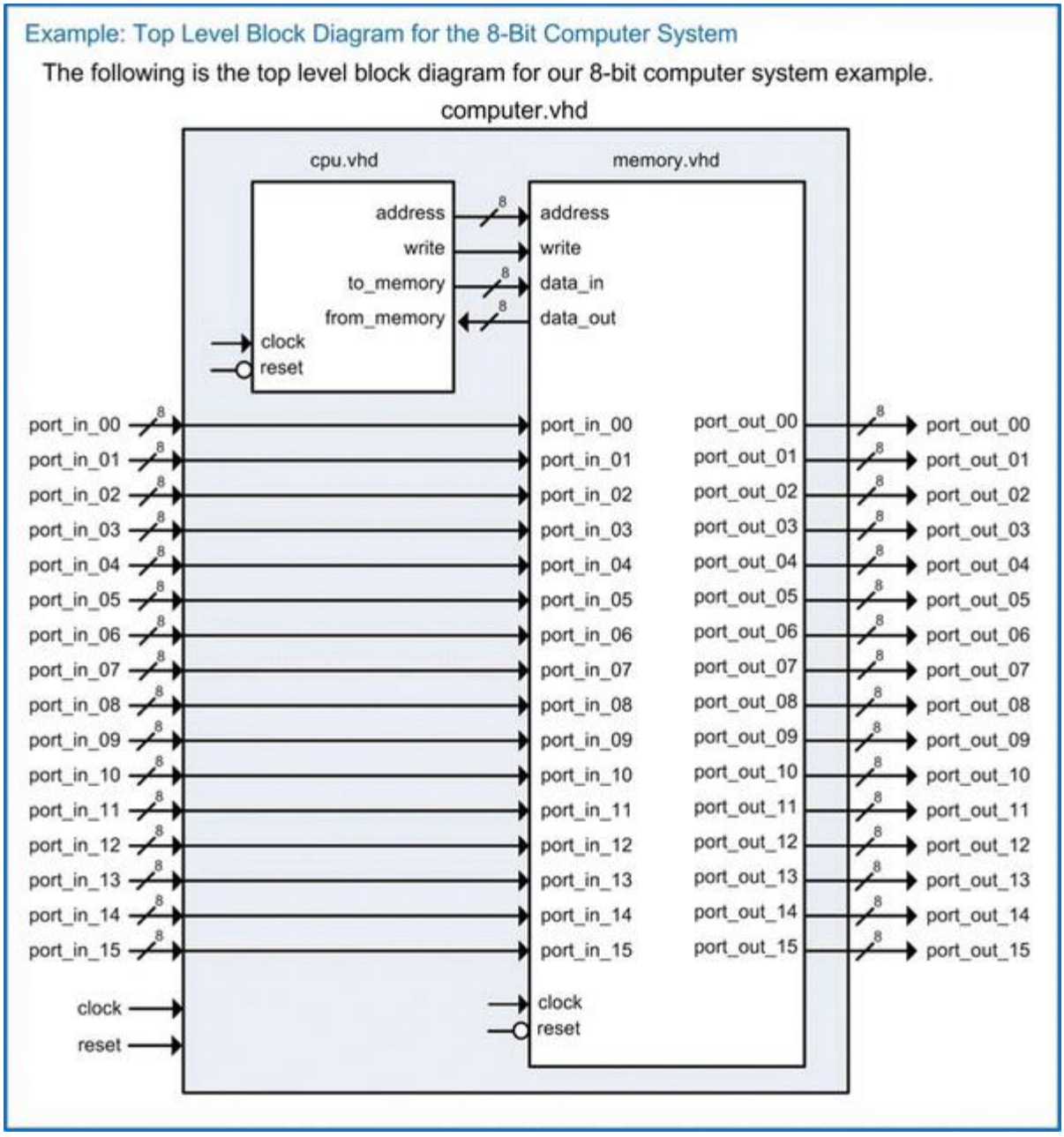
9. Увод у пројектовање рачунарских система

Задатак 13.3
Имплементирати 8-битни рачунарски систем. Детаљи имплементације и извршавање инструкција је приказано на блок дијаграму на слици. Блок дијаграм садржи VHDL фајл и имена ентитета.



Слика. 13.8 Блок дијаграм највишег нивоа за 8-битни рачунарски систем

Решење задатка 13.3

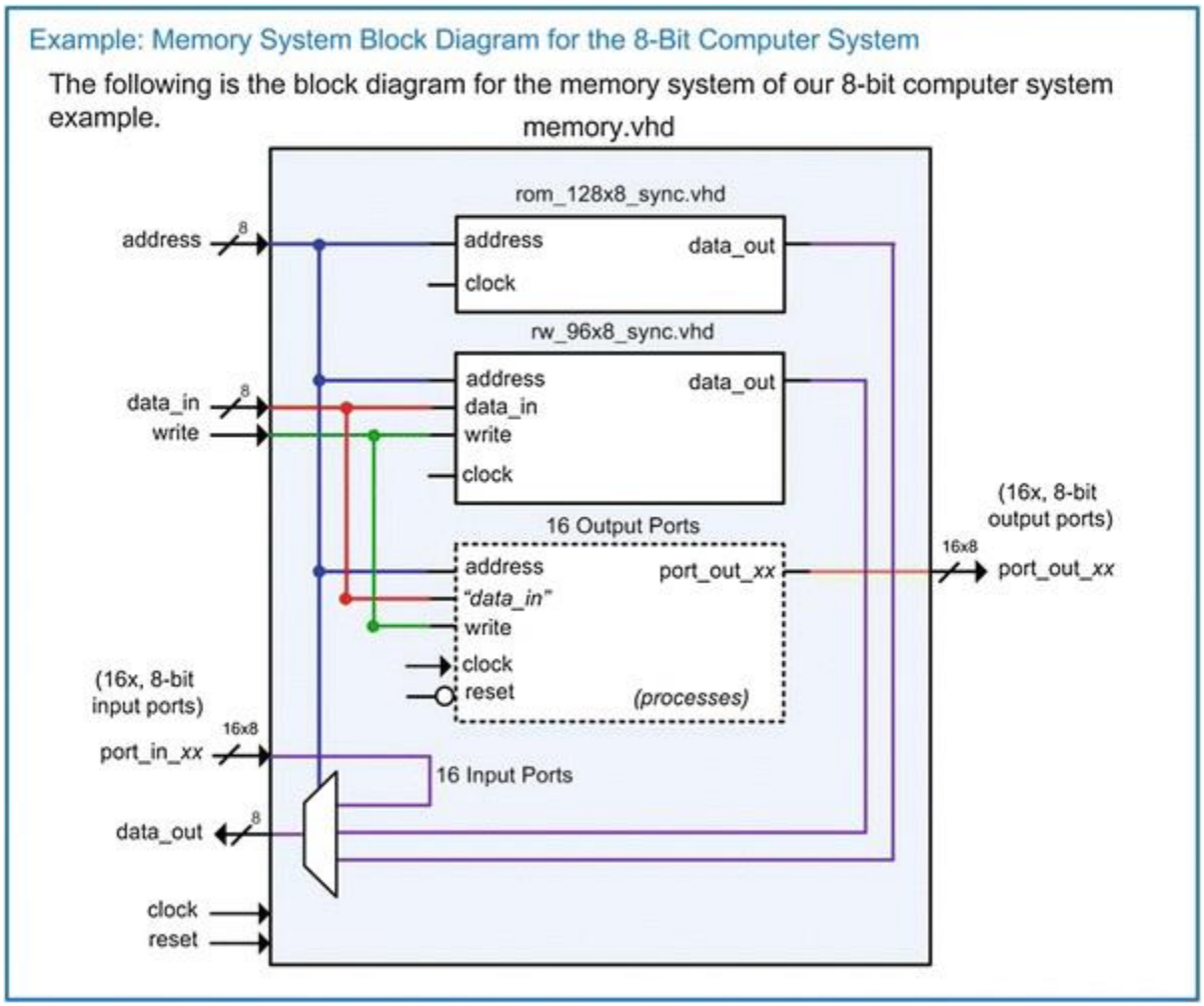
Сет инструкција за 8-битни рачунарски систем:

Example: Instruction Set for the 8-Bit Computer System

The following is a base set of instructions that the 8-bit computer system will be able to perform. Each instruction is given a descriptive mnemonic, which allows the system implementation and the programming to be more intuitive. Each instruction is also provided with a unique binary opcode. Some instructions have an operand, which provides additional information necessary for the instruction. If an instruction contains an operand, a description is provided as to how it is used (e.g., as data or as an address).

Mnemonic	Opcode, Operand	Description
"Loads and Stores"		
LDA_IMM	x"86", <*data*>	Load Register A using Immediate Addressing
LDA_DIR	x"87", <*addr*>	Load Register A using Direct Addressing
LDB_IMM	x"88", <*data*>	Load Register B with Immediate Addressing
LDB_DIR	x"89", <*addr*>	Load Register B with Direct Addressing
STA_DIR	x"96", <*addr*>	Store Register A to Memory using Direct Addressing
STB_DIR	x"97", <*addr*>	Store Register B to Memory using Direct Addressing
"Data Manipulations"		
ADD_AB	x"42"	A = A + B (plus)
SUB_AB	x"43"	A = A - B (minus)
AND_AB	x"44"	A = A · B (AND)
OR_AB	x"45"	A = A + B (OR)
INCA	x"46"	A = A + 1 (plus)
INCB	x"47"	B = B + 1 (plus)
DECA	x"48"	A = A - 1 (minus)
DECB	x"49"	B = B - 1 (minus)
"Branches"		
BRA	x"20", <*addr*>	Branch Always to Address Provided
BMI	x"21", <*addr*>	Branch to Address Provided if N=1
BPL	x"22", <*addr*>	Branch to Address Provided if N=0
BEQ	x"23", <*addr*>	Branch to Address Provided if Z=1
BNE	x"24", <*addr*>	Branch to Address Provided if Z=0
BVS	x"25", <*addr*>	Branch to Address Provided if V=1
BVC	x"26", <*addr*>	Branch to Address Provided if V=0
BCS	x"27", <*addr*>	Branch to Address Provided if C=1
BCC	x"28", <*addr*>	Branch to Address Provided if C=0

Блок дијаграм за меморијски систем за 8-битни рачунарски систем:



VHDL имплементација меморијског система за програм за 8-битни рачунарски систем:

```
constant LDA_IMM : std_logic_vector (7 downto 0) := x"86";
constant LDA_DIR : std_logic_vector (7 downto 0) := x"87";
constant LDB_IMM : std_logic_vector (7 downto 0) := x"88";
constant LDB_DIR : std_logic_vector (7 downto 0) := x"89";
constant STA_DIR : std_logic_vector (7 downto 0) := x"96";
constant STB_DIR : std_logic_vector (7 downto 0) := x"97";
constant ADD_AB : std_logic_vector (7 downto 0) := x"42";
constant SUB_AB : std_logic_vector (7 downto 0) := x"43";
constant AND_AB : std_logic_vector (7 downto 0) := x"44";
constant OR_AB : std_logic_vector (7 downto 0) := x"45";
constant INCA : std_logic_vector (7 downto 0) := x"46";
constant INCB : std_logic_vector (7 downto 0) := x"47";
constant DECA : std_logic_vector (7 downto 0) := x"48";
constant DECB : std_logic_vector (7 downto 0) := x"49";
constant BRA : std_logic_vector (7 downto 0) := x"20";
constant BMI : std_logic_vector (7 downto 0) := x"21";
constant BPL : std_logic_vector (7 downto 0) := x"22";
constant BEQ : std_logic_vector (7 downto 0) := x"23";
constant BNE : std_logic_vector (7 downto 0) := x"24";
constant BVS : std_logic_vector (7 downto 0) := x"25";
constant BVC : std_logic_vector (7 downto 0) := x"26";
constant BCS : std_logic_vector (7 downto 0) := x"27";
constant BCC : std_logic_vector (7 downto 0) := x"28";
```

```
type rom_type is array (0 to 127) of std_logic_vector(7 downto 0);
constant ROM : rom_type := (0 => LDA_IMM,
                            1 => x"AA",
                            2 => STA_DIR,
                            3 => x"E0",
                            4 => BRA,
                            5 => x"00",
                            others => x"00");

enable : process (address)
begin
      if ((to_integer(unsigned(address)) >= 0) and
         (to_integer(unsigned(address)) <= 127)) then
            EN <= '1';
      else
            EN <= '0';
      end if;
end process;

memory : process (clock)
begin
      if (clock'event and clock='1') then
            if (EN='1') then
                  data_out <= ROM(to_integer(unsigned(address)));
            end if;
      end if;
end process;
```

VHDL имплементација меморијског система за податке за 8-битни рачунарски систем:

```
type rw_type is array (128 to 223) of std_logic_vector(7 downto 0);
signal RW : rw_type;

enable : process (address)
begin
      if ((to_integer(unsigned(address)) >= 128) and
         (to_integer(unsigned(address)) <= 223)) then
            EN <= '1';
      else
            EN <= '0';
      end if;
end process;

memory : process (clock)
begin
      if (clock'event and clock='1') then
            if (EN='1' and write='1') then
                  RW(to_integer(unsigned(address))) <= data_in;
            elsif (EN='1' and write='0') then
                  data_out <= RW(to_integer(unsigned(address)));
            end if;
      end if;
end process;
```

VHDL имплементација излазних портова

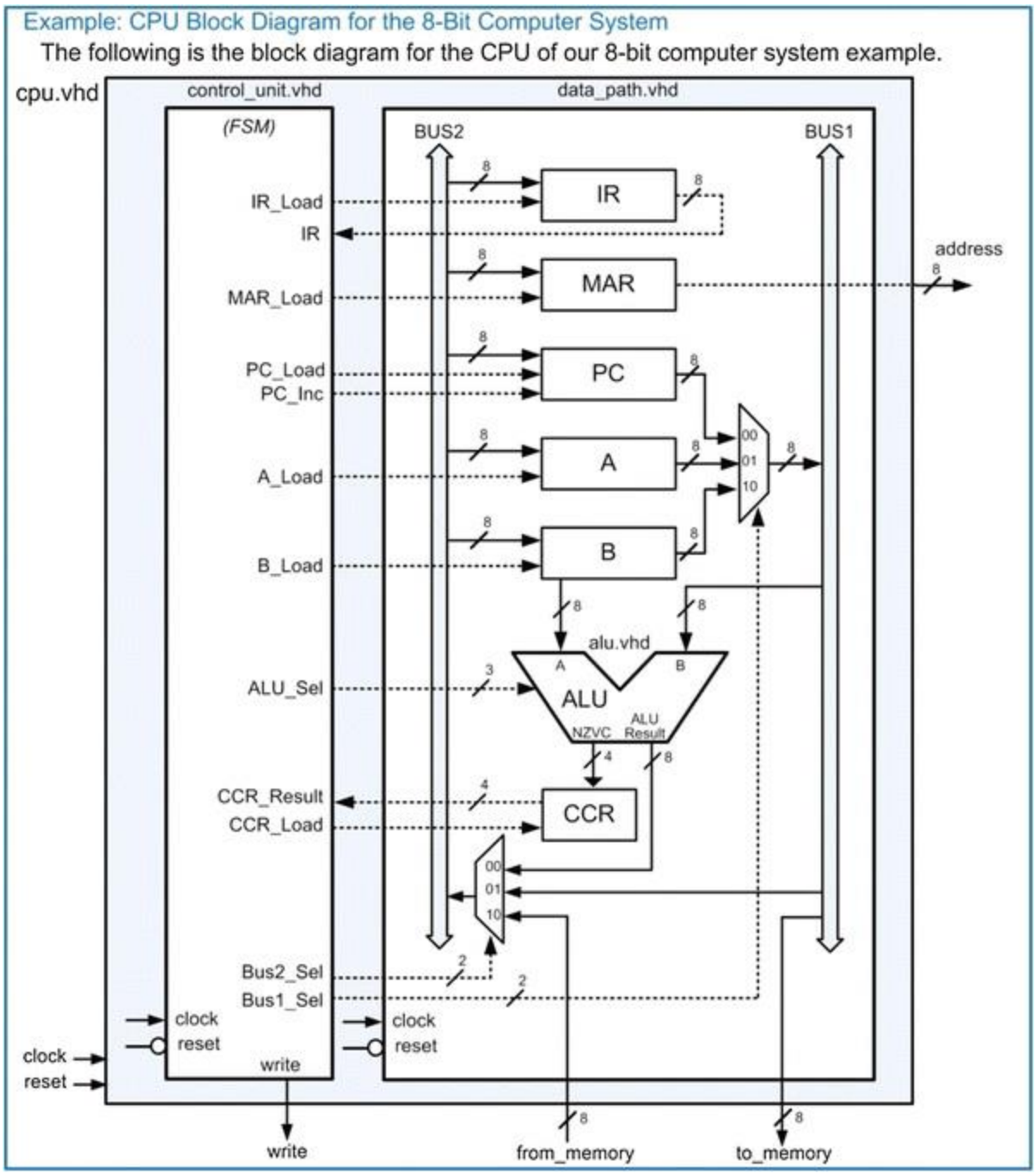
```
-- : ADDRESS x"E0"
U3 : process (clock, reset)
begin
      if (reset = '0') then
            port_out_00 <= x"00";
      elsif (clock'event and clock='1') then
            if (address = x"E0" and write = '1') then
                  port_out_00 <= data_in;
            end if;
```

```
      end if;
end process;
-- port_out_01 description : ADDRESS x"E1"
U4 : process (clock, reset)
begin
      if (reset = '0') then
            port_out_01 <= x"00";
      elsif (clock'event and clock='1') then
            if (address = x"E1" and write = '1') then
                  port_out_01 <= data_in;
            end if;
      end if;
end process;:
"the rest of the output port models go here…"
```

VHDL имплементација за Memory data_out Bus

```
MUX1 : process (address, rom_data_out, rw_data_out,port_in_00, port_in_01,
port_in_02, port_in_03,port_in_04, port_in_05, port_in_06, port_in_07,port_in_08,
port_in_09, port_in_10, port_in_11,port_in_12, port_in_13, port_in_14, port_in_15)
begin
      if ((to_integer(unsigned(address)) >= 0) and
         (to_integer(unsigned(address)) <= 127)) then
            data_out <= rom_data_out;
      elsif ((to_integer(unsigned(address)) >= 128) and
            (to_integer(unsigned(address)) <= 223)) then
            data_out <= rw_data_out;
      elsif (address = x"F0") then data_out <= port_in_00;
      elsif (address = x"F1") then data_out <= port_in_01;
      elsif (address = x"F2") then data_out <= port_in_02;
      elsif (address = x"F3") then data_out <= port_in_03;
      elsif (address = x"F4") then data_out <= port_in_04;
      elsif (address = x"F5") then data_out <= port_in_05;
      elsif (address = x"F6") then data_out <= port_in_06;
      elsif (address = x"F7") then data_out <= port_in_07;
      elsif (address = x"F8") then data_out <= port_in_08;
      elsif (address = x"F9") then data_out <= port_in_09;
      elsif (address = x"FA") then data_out <= port_in_10;
      elsif (address = x"FB") then data_out <= port_in_11;
      elsif (address = x"FC") then data_out <= port_in_12;
      elsif (address = x"FD") then data_out <= port_in_13;
      elsif (address = x"FE") then data_out <= port_in_14;
      elsif (address = x"FF") then data_out <= port_in_15;
      else data_out <= x"00";
end if;
end process;
```

Имплементација за CPU



VHDL имплементација за CPU Data Path

```
MUX_BUS1 : process (Bus1_Sel, PC, A, B)
begin
      case (Bus1_Sel) is
      when "00" => Bus1 <= PC;
      when "01" => Bus1 <= A;
      when "10" => Bus1 <= B;
      when others => Bus1 <= x"00";
end case;
end process;

MUX_BUS2 : process (Bus2_Sel, ALU_Result, Bus1, from_memory)
begin
      case (Bus2_Sel) is
            when "00" => Bus2 <= ALU_Result;
            when "01" => Bus2 <= Bus1;
```

```
            when "10" => Bus2 <= from_memory;
            when others => Bus2 <= x"00";
      end case;
end process;

address <= MAR;
to_memory <= Bus1;

INSTRUCTION_REGISTER : process (Clock, Reset)
begin
if (Reset = '0') then
IR <= x"00";
elsif (Clock'event and Clock = '1') then
if (IR_Load = '1') then
IR <= Bus2;
end if;
end if;
end process;

MEMORY_ADDRESS_REGISTER : process (Clock, Reset)
begin
if (Reset = '0') then
MAR <= x"00";
elsif (Clock'event and Clock = '1') then
if (MAR_Load = '1') then
MAR <= Bus2;
end if;
end if;
end process;

PROGRAM_COUNTER : process (Clock, Reset)
begin
if (Reset = '0') then
PC_uns <= x"00";
elsif (Clock'event and Clock = '1') then
if (PC_Load = '1') then
PC_uns <= unsigned(Bus2);
elsif (PC_Inc = '1') then
PC_uns <= PC_uns + 1;
end if;
end if;
end process;

PC <= std_logic_vector(PC_uns);

A_REGISTER : process (Clock, Reset)
begin
if (Reset = '0') then
A <= x"00";
elsif (Clock'event and Clock = '1') then
if (A_Load = '1') then
A <= Bus2;
end if;
end if;
end process;
B_REGISTER : process (Clock, Reset)
begin
if (Reset = '0') then
B <= x"00";
elsif (Clock'event and Clock = '1') then
if (B_Load = '1') then
B <= Bus2;
end if;
end if;
end process;
```

```
CONDITION_CODE_REGISTER : process (Clock, Reset)
begin
if (Reset = '0') then
CCR_Result <= x"0";
elsif (Clock'event and Clock = '1') then
if (CCR_Load = '1') then
CCR_Result <= NZVC;
end if;
end if;
end process;
```

VHDL имплементација за ALU

```
ALU_PROCESS : process (A, B, ALU_Sel)
      variable Sum_uns : unsigned(8 downto 0);
begin
      if (ALU_Sel = "000") then – ADDITION
            --- Sum Calculation ----------------------------------
            Sum_uns := unsigned('0' & A) + unsigned('0' & B);
            Result <= std_logic_vector(Sum_uns(7 downto 0));
            --- Negative Flag (N) -------------------------------
            NZVC(3) <= Sum_uns(7);
            --- Zero Flag (Z) -----------------------------------
            if (Sum_uns(7 downto 0) = x"00") then
                  NZVC(2) <= '1';
            else
                  NZVC(2) <= '0';
            end if;
            --- Overflow Flag (V) -------------------------------
            if ((A(7)='0' and B(7)='0' and Sum_uns(7)='1') or
               (A(7)='1' and B(7)='1' and Sum_uns(7)='0')) then
                  NZVC(1) <= '1';
            else
                  NZVC(1) <= '0';
            end if;
            --- Carry Flag (C) ----------------------------------
            NZVC(0) <= Sum_uns(8);
            elsif (ALU_Sel =: “other ALU functionality goes here”
end if;
end process;
```

VHDL имплементација за контролну јединицу

```
type state_type is (S_FETCH_0, S_FETCH_1, S_FETCH_2, S_DECODE_3, S_LDA_IMM_4,
S_LDA_IMM_5, S_LDA_IMM_6, S_LDA_DIR_4, S_LDA_DIR_5, S_LDA_DIR_6, S_LDA_DIR_7,
S_STA_DIR_4, S_STA_DIR_5, S_STA_DIR_6, S_STA_DIR_7, S_STA_DIR_8, S_ADD_AB_4,
S_BRA_4, S_BRA_5, S_BRA_6, S_BEQ_4, S_BEQ_5, S_BEQ_6, S_BEQ_7);
signal current_state, next_state : state_type;

STATE_MEMORY : process (Clock, Reset)
begin
      if (Reset = '0') then
            current_state <= S_FETCH_0;
      elsif (clock'event and clock = '1') then
            current_state <= next_state;
      end if;
end process;


NEXT_STATE_LOGIC : process (current_state, IR, CCR_Result)
begin
      if (current_state = S_FETCH_0) then
            next_state <= S_FETCH_1;
      elsif (current_state = S_FETCH_1) then
```

```
            next_state <= S_FETCH_2;
      elsif (current_state = S_FETCH_2) then
            next_state <= S_DECODE_3;
      elsif (current_state = S_DECODE_3) then
            -- select execution path
            if (IR = LDA_IMM) then -- Load A Immediate
                  next_state <= S_LDA_IMM_4;
            elsif (IR = LDA_DIR) then -- Load A Direct
                  next_state <= S_LDA_DIR_4;
            elsif (IR = STA_DIR) then -- Store A Direct
                  next_state <= S_STA_DIR_4;
            elsif (IR = ADD_AB) then -- Add A and B
                  next_state <= S_ADD_AB_4;
            elsif (IR = BRA) then -- Branch Always
                  next_state <= S_BRA_4;
            elsif (IR=BEQ and CCR_Result(2)='1') then – BEQ and Z=1
                  next_state <= S_BEQ_4;
            elsif (IR=BEQ and CCR_Result(2)='0') then – BEQ and Z=0
                  next_state <= S_BEQ_7;
            else
                  next_state <= S_FETCH_0;
            end if;
      elsif… : "paths for each instruction go here…" :
end if;
end process;


OUTPUT_LOGIC : process (current_state)
begin
      case(current_state) is
            when S_FETCH_0 => -- Put PC onto MAR to read Opcode
                  IR_Load <= '0';
                  MAR_Load <= '1';
                  PC_Load <= '0';
                  PC_Inc <= '0';
                  A_Load <= '0';
                  B_Load <= '0';
                  ALU_Sel <= "000";
                  CCR_Load <= '0';
                  Bus1_Sel <= "00"; -- "00"=PC, "01"=A, "10"=B
                  Bus2_Sel <= "01"; -- "00"=ALU_Result, "01"=Bus1,
                  "10"=from_memory
                  write <= '0';
            when S_FETCH_1 => -- Increment PC
                  IR_Load <= '0';
                  MAR_Load <= '0';
                  PC_Load <= '0';
                  PC_Inc <= '1';
                  A_Load <= '0';
                  B_Load <= '0';
                  ALU_Sel <= "000";
                  CCR_Load <= '0';
                  Bus1_Sel <= "00"; -- "00"=PC, "01"=A, "10"=B
                  Bus2_Sel <= "00"; -- "00"=ALU, "01"=Bus1, "10"=from_memory
                  write <= '0';
                  : "output assignments for all other states go here…" :
      end case;
end process;
```