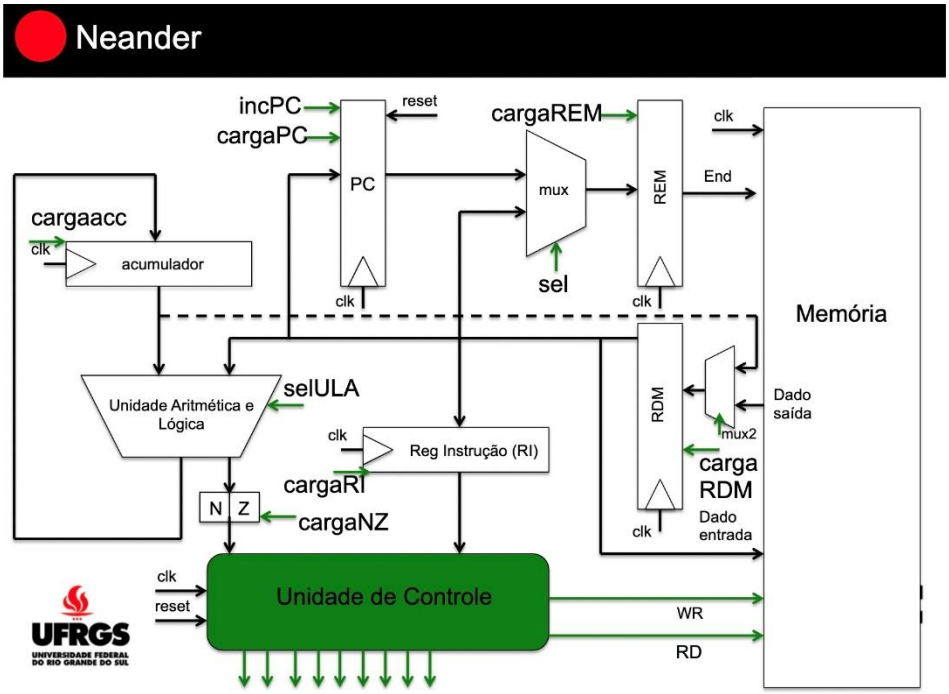
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMATICA
Trabalho 1 - Sistemas Digitais – Parte 1
Prof. Fernanda Kastensmidt

Objetivo do Trabalho Completo a ser enviado em Fevereiro 10/2: projetar e descrever em VHDL o processador AHMES (https://www.inf.ufrgs.br/arq/wiki/doku.php?id=insahmes) implementar 2 programas em sua memória e mostrar através de simulação lógica sem e com atraso o funcionamento.

Parte 1 – a ser realizado na aula 24/1/23

O DATAPATH do AHMES é o mesmo do NEANDER mas a ULA e o registrador ACC muda.



Conjunto de instruções do AHMES

Código	Instrução	Operação
0000 xxxx	NOP	Nenhuma operação
0001 xxxx	STA end	Armazena acumulador no endereço “end” da memória
0010 xxxx	LDA end	Carrega o acumulador com o conteúdo do endereço “end” da memória
0011 xxxx	ADD end	Soma o conteúdo do endereço “end” da memória ao acumulador
0100 xxxx	OR end	Efetua operação lógica “OU” do conteúdo do endereço “end” da memória ao acumulador
0101 xxxx	AND end	Efetua operação lógica “E” do conteúdo do endereço “end” da memória ao acumulador
0110 xxxx	NOT	Inverte todos os bits do acumulador
0111 xxxx	SUB end	Subtrai o conteúdo do endereço “end” da memória do acumulador
1000 xxxx	JMP end	Desvio incondicional para o endereço “end” da memória
1001 00xx	JN end	Desvio condicional, se “N=1”, para o endereço “end” da memória
1001 01xx	JP end	Desvio condicional, se “N=0”, para o endereço “end” da memória
1001 10xx	JV end	Desvio condicional, se “V=1”, para o endereço “end” da memória
1001 11xx	JNV end	Desvio condicional, se “V=0”, para o endereço “end” da memória
1010 00xx	JZ end	Desvio condicional, se “Z=1”, para o endereço “end” da memória
1010 01xx	JNZ end	Desvio condicional, se “Z=0”, para o endereço “end” da memória
1011 00xx	JC end	Desvio condicional, se “C=1”, para o endereço “end” da memória
1011 01xx	JNC end	Desvio condicional, se “C=0”, para o endereço “end” da memória
1011 10xx	JB end	Desvio condicional, se “B=1”, para o endereço “end” da memória
1011 11xx	JNB end	Desvio condicional, se “B=0”, para o endereço “end” da memória

1110 xx00	SHR	Desloca o acumulador para a direita; o bit 7 do acumulador recebe 0; o Carry recebe o bit 0 do acumulador
1110 xx01	SHL	Desloca o acumulador para a esquerda, o bit 0 do acumulador recebe 0; o Carry recebe o bit 7 do acumulador
1110 xx00	ROR	Gira o acumulador para a direita; o bit 7 do acumulador recebe o Carry; o Carry recebe o bit 0 do acumulador
1110 xx01	ROL	Gira o acumulador para a esquerda, o bit 0 do acumulador recebe o Carry; o Carry recebe o bit 7 do acumulador
1111 xxxx	HLT	Para o ciclo de busca-decodificação-execução

AULA 1 (24/1/2023)

Descrever o DATAPATH do processador RAMSES em VHDL em uma entidade apenas chamada de datapath_ramses.

1) ***Cole aqui o código completo em VHDL do datapath***

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

entity datapath_ahmes is
    Port
    (
        CLOCK : in STD_LOGIC;
        RESET : in STD_LOGIC := '0';
        DOUT : out STD_LOGIC_VECTOR (7 downto 0);
        N : out STD_LOGIC;
        Z : out STD_LOGIC;
        V : out STD_LOGIC;
        C : out STD_LOGIC;
        B : out STD_LOGIC
    );
end datapath_ahmes;

architecture Behavioral of datapath_ahmes is
    -- PC
    signal reg_PC: std_logic_vector (7 downto 0);
    signal load_PC: std_logic := '0';
    signal inc_PC: std_logic := '0';

    -- AC
    signal reg_AC: std_logic_vector (7 downto 0);
    signal load_AC: std_logic := '0';

    -- ULA
    signal ULA_X: std_logic_vector (7 downto 0); -- é o reg_AC
    signal ULA_Y: std_logic_vector (7 downto 0); -- é o reg_RDM;
    signal ULA_out: std_logic_vector (7 downto 0);
    signal sel_ULA: std_logic_vector (3 downto 0);

    -- FLAGS
    signal reg_N: std_logic := '0';
```

```
signal reg_Z: std_logic := '0';
signal reg_V: std_logic := '0';
signal reg_C: std_logic := '0';
signal reg_B: std_logic := '0';
signal load_flag: std_logic := '0';

-- MUX_REM
signal MUX_REM_out: std_logic_vector (7 downto 0);
signal sel_REM_MUX: std_logic :='0';

-- REM
signal reg_REM: std_logic_vector (7 downto 0);
signal load_REM: std_logic := '0';

-- MUX_RDM
signal MUX_RDM_out: std_logic_vector (7 downto 0);
signal sel_RDM_MUX: std_logic :='0';

-- RDM
signal reg_RDM: std_logic_vector (7 downto 0);
signal load_RDM: std_logic := '0';

-- RI
signal reg_RI: std_logic_vector (7 downto 0);
signal load_RI: std_logic := '0';

-- DECOD
signal reg_DECOD: std_logic_vector(7 downto 0);
signal DECOD_out: std_logic_vector(23 downto 0);
signal DECOD_sel: std_logic_vector(4 downto 0);

-- MEM
signal reg_mem: std_logic_vector (7 downto 0);
signal read_mem: std_logic := '0';
signal write_mem: std_logic := '0';

-- variables
component memoria
   PORT
   (
      clka : IN STD_LOGIC;
      wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      addra : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
end component;

begin -- inicio behavioral
   -- MEM
   --MEM: memoria
   --    PORT MAP
```

```
--    (
--        clka => CLOCK,
--        wea => write_mem,
--        addra => reg_mem,
--        dina => reg_AC,
--        douta => reg_mem
--    );


process (CLOCK, RESET) -- PC
    begin
        if (RESET ='1') then
            reg_PC <= "00000000";
        elsif (rising_edge(CLOCK)) then -- na subida do clock
            if (load_PC='1') then
                reg_PC<= reg_RDM;
            elsif(inc_PC='1') then
                reg_PC <= std_logic_vector(unsigned(reg_PC) + 1);
            else
                reg_PC <= reg_PC;
            end if;
        end if;
    end process;
process (CLOCK, RESET) -- AC
    begin
        if (RESET='1') then
            reg_AC <= "00000000";
        elsif (rising_edge(CLOCK)) then -- na subida do clock
            if (load_AC='1') then
            reg_AC <= ULA_out;
            else
            reg_AC <= reg_AC;
            end if;
        end if;
    end process;
-- ULA
ULA_X <= reg_AC;  -- recupera valor X do acumulador
ULA_Y <= reg_MEM;  -- recupera valor Y de uma posicao da memória
process(sel_ULA, ULA_X, ULA_Y, reg_C)
                                variable C_temp: std_logic := '0';

    begin
        case sel_ULA is
            when "0000" =>
                if((ULA_X + ULA_Y) > 127) then
                    reg_V <= '1';
                else
                    reg_V <= '0';
                end if;

                reg_B <= '0';
                reg_C <= '0';
```

```
    ULA_out <= (ULA_X + ULA_Y); -- operacao ADD
when "0001" =>
    reg_V <= '0';
    reg_B <= '0';
    reg_C <= '0';
    ULA_out <= (ULA_X OR ULA_Y); -- operacao OR
when "0010" =>
    reg_V <= '0';
    reg_B <= '0';
    reg_C <= '0';
    ULA_out <= (ULA_X AND ULA_Y); -- operacao AND
when "0011" =>
    reg_V <= '0';
    reg_B <= '0';
    reg_C <= '0';
    ULA_out <= (NOT ULA_X); -- operacao NOT
when "0100" =>
    if((ULA_X - ULA_Y) < 0) then
        reg_C <= '1';
        reg_B <= '1';
    else
        reg_V <= '0';
        reg_B <= '0';
        reg_C <= '0';
    end if;

    reg_V <= '0';
    ULA_out <= (ULA_X - ULA_Y); -- operacao SUB
when "0101" => -- operacao SHR
    reg_V <= '0';
    reg_B <= '0';
    reg_C <= ULA_X(0);
    ULA_out(7) <= '0';
    ULA_OUT(6) <= ULA_X(7);
    ULA_OUT(5) <= ULA_X(6);
    ULA_OUT(4) <= ULA_X(5);
    ULA_OUT(3) <= ULA_X(4);
    ULA_OUT(2) <= ULA_X(3);
    ULA_OUT(1) <= ULA_X(2);
    ULA_OUT(0) <= ULA_X(1);
when "0110" => -- operacao SHL
    reg_V <= '0';
    reg_B <= '0';
    reg_C <= ULA_X(7);
    ULA_out <= (ULA_X + ULA_X);
when "0111" => -- operacao ROR
    reg_V <= '0';
    reg_B <= '0';
    C_temp := ULA_X(0);
    ULA_out(7) <= reg_C;
    ULA_OUT(6) <= ULA_X(7);
    ULA_OUT(5) <= ULA_X(6);
```

```
            ULA_OUT(4) <= ULA_X(5);
            ULA_OUT(3) <= ULA_X(4);
            ULA_OUT(2) <= ULA_X(3);
            ULA_OUT(1) <= ULA_X(2);
            ULA_OUT(0) <= ULA_X(1);
            reg_C <= C_temp;
         when "1000" => -- operacao ROL
            reg_V <= '0';
            reg_B <= '0';
            C_temp := ULA_X(7);
            ULA_out <= (ULA_X + ULA_X);
            ULA_out(0) <= reg_C;
            reg_C <= C_temp;
         when others => ULA_out <= "XXXXXXXX";
      end case;
   end process;

process (CLOCK, RESET) -- FLAGS
   begin
      if (RESET = '1') then
         reg_N <= '0';
         reg_Z <= '0';
         reg_V <= '0';
         reg_B <= '0';
         --reg_C <= '0';
      elsif(rising_edge(CLOCK)) then -- na subida do clock
         if reg_AC = "00000000" then
            reg_Z <= '1';
         else
            reg_Z <= '0';
         end if;
         reg_N <= reg_AC(7); -- msb do registrador AC
      end if;
   end process;
process (sel_REM_MUX, reg_PC, MUX_RDM_out) -- MUX_REM
   begin
      if (sel_REM_MUX = '0') then
         MUX_REM_out <= reg_PC;
      else
         MUX_REM_out <= MUX_RDM_out;
      end if;
   end process;
process(CLOCK, RESET) -- REM
   begin
      if (RESET='1') then
         reg_REM <= "00000000";
      elsif (rising_edge(CLOCK)) then -- na subida do clock
         if (load_REM ='1') then
            reg_REM <= MUX_REM_out;
         else
            reg_REM <= reg_REM;
         end if;
```

```
        end if;
    end process;
process (sel_RDM_MUX, reg_mem, reg_AC) -- MUX_RDM
    begin
        if(sel_REM_MUX = '0') then
            MUX_RDM_out <= reg_MEM;
        else
            MUX_RDM_out <= reg_AC;
        end if;
    end process;
process (CLOCK, RESET) -- RDM
    begin
        if (RESET = '1') then
            reg_RDM <= "00000000";
        elsif (rising_edge(CLOCK)) then
            if (load_RDM = '1') then
                reg_RDM <= reg_AC;
            else
                reg_RDM <= MUX_RDM_out;
            end if;
        end if;
    end process;
process (CLOCK, RESET) -- RI
    begin
        if (RESET='1') then
            reg_RI <= "00000000";
        elsif (rising_edge(CLOCK)) then
            if (load_RI = '1') then
                reg_RI <= reg_RDM;
            else
                reg_RI <= reg_RI;
            end if;
        end if;
    end process;

-- DECOD
reg_DECOD <= reg_RI;
process(reg_decod)
    begin
        DECOD_out <= "00000000000000000000000";
        case reg_DECOD is
            when "00000000" => DECOD_out(0) <= '1';  -- 00 NOP
            when "00010000" => DECOD_out(1) <= '1';  -- 16 STA
            when "00100000" => DECOD_out(2) <= '1';  -- 32 LDA
            when "00110000" => DECOD_out(3) <= '1';  -- 48 ADD
            when "01000000" => DECOD_out(4) <= '1';  -- 64 OR
            when "01010000" => DECOD_out(5) <= '1';  -- 80 AND
            when "01100000" => DECOD_out(6) <= '1';  -- 96 NOT
            when "01110000" => DECOD_out(7) <= '1';  -- 112 SUB
            when "10000000" => DECOD_out(8) <= '1';  -- 128 JMP
            when "10010000" => DECOD_out(9) <= '1';  -- 144 JN
            when "10010100" => DECOD_out(10) <= '1'; -- 148 JP
```

```
            when "10011000" => DECOD_out(11) <= '1'; -- 152 JV
            when "10011100" => DECOD_out(12) <= '1'; -- 156 JNV
            when "10100000" => DECOD_out(13) <= '1'; -- 160 JZ
            when "10100100" => DECOD_out(14) <= '1'; -- 164 JNZ
            when "10110000" => DECOD_out(15) <= '1'; -- 176 JC
            when "10110100" => DECOD_out(16) <= '1'; -- 180 JNC
            when "10111000" => DECOD_out(17) <= '1'; -- 184 JB
            when "10111100" => DECOD_out(18) <= '1'; -- 188 JNB
            when "11100000" => DECOD_out(19) <= '1'; -- 224 SHR
            when "11100001" => DECOD_out(20) <= '1'; -- 225 SHL
            when "11100010" => DECOD_out(21) <= '1'; -- 226 ROR
            when "11100011" => DECOD_out(22) <= '1'; -- 227 ROL
            when "11110000" => DECOD_out(23) <= '1'; -- 240 HLT
            when others => DECOD_out <= "000000000000000000000000";
        end case;
    end process;
end Behavioral;
```

2) Qual componente FPGA escolheste para a síntese? Artix 7 – xc7a100t-3csg324

3) Quantos registradores tem o datapath do AHMES? 13

4) Quantas operações diferentes tem a ULA? 9

5) A área do DATAPTH em # LUTs: _____ e #ffps: _____

*****ENTREGAR DIA da PARTE 1 29/1/2022**** Editando esse DOC e submetendo no MS-TEAMS**