UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DISCIPLINA: SISTEMAS DIGITAIS
Prof. Fernanda Lima Kastensmidt

Aula Remota NEANDER – parte I

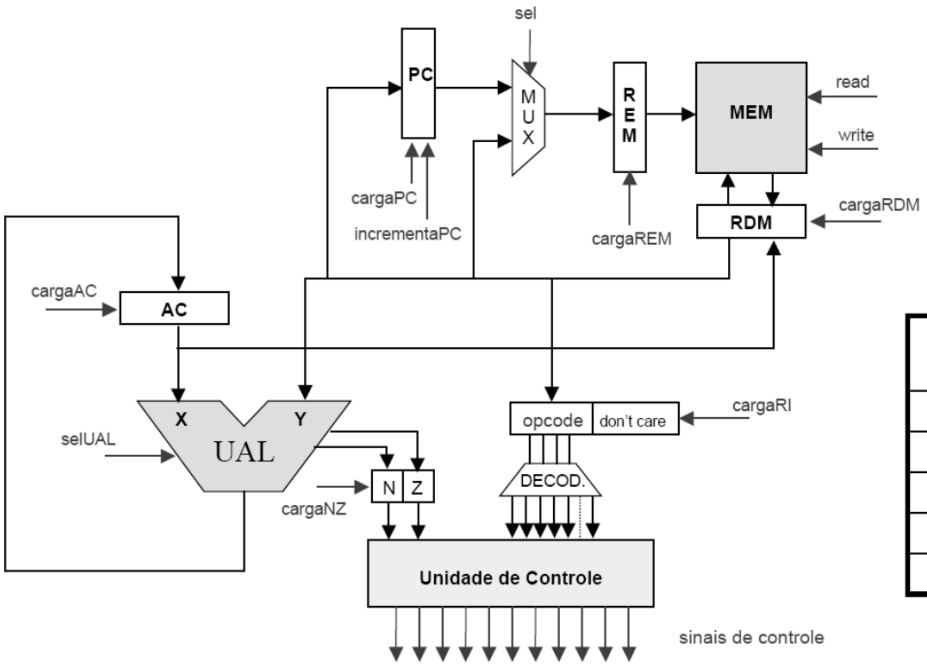
Giordano Souza de Paula - 00308054

Projeto do Processador Neander em VHDL

O computador NEANDER foi criado com intenções didáticas pelo prof. Raul Weber da UFRGS. Neste site há referencias e link para o simulador: http://www.dcc.ufrj.br/~gabriel/neander.php

O objetivo deste trabalho de SD é implementar o NEANDER usando a linguagem de descrição de hardware VHDL, simular esse circuito em um simulador lógico sem atraso, depois e realizar a síntese lógica, mapeamento tecnológico, posicionamento e roteamento para um FPGA, realizar a simulação com atraso e prototipar o processador em uma placa de prototipação.

O computador NEANDER tem as seguintes características:
• Largura de dados e endereços de 8 bits
• Dados representados em complemento de dois
• 1 acumulador de 8 bits (AC)
• 1 apontador de programa de 8 bits (PC)
• 1 registrador de estado com 2 códigos de condição: negativo (N) e zero (Z)



operações da UAL	selUAL
X + Y	000
X and Y	001
X or Y	010
Not X	011
Y	100

A aula 1 remota do NEANDER pede para descrever em VHDL o Datapath do Neander, ou seja, tudo que está na figura, menos a parte de controle (que será uma FSM e a memória BRAM).

1) INSIRA AQUI O VHDL DO DATAPATH

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

entity neander is
    Port
    (
        CLOCK : in STD_LOGIC;
        RESET : in STD_LOGIC;
        DOUT : out STD_LOGIC_VECTOR (7 downto 0);
        N : out STD_LOGIC;
        Z : out STD_LOGIC
    );
end neander;

architecture Behavioral of neander is

    -- PC
    signal regPC: std_logic_vector (7 downto 0);
    signal cargaPC: std_logic := '0';
    signal incrementaPC: std_logic := '0';

    -- AC
    signal regAC: std_logic_vector (7 downto 0);
    signal saidaAC: std_logic_vector (7 downto 0);
    signal cargaAC: std_logic := '0';

    -- ULA
    signal ULAX: std_logic_vector (7 downto 0);
    signal ULAY: std_logic_vector (7 downto 0);
    signal saidaULA: std_logic_vector (7 downto 0);
    signal selULA: std_logic_vector (7 downto 0);

    -- FLAGS N E Z
    signal regN: std_logic := '0';
    signal regZ: std_logic := '1';
    signal cargaNZ: std_logic := '0';

    -- REM
```

```
    signal regREM: std_logic_vector (7 downto 0);
    signal cargaREM: std_logic := '0';

    -- RDM
    signal regRDM: std_logic_vector (7 downto 0);
    signal cargaRDM: std_logic := '0';

    -- RI
    signal regRI: std_logic_vector (7 downto 4);
    signal cargaRI: std_logic := '0';

    -- MUX2x1
    signal saidaMUX: std_logic_vector (7 downto 0);
    signal selMUX: std_logic :='0';

    -- MEMORY
    signal memOut: std_logic_vector (7 downto 0);
    signal writeMem: std_logic := '0';

    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)
         );

    -- MEM
    MEM: memoria
        PORT MAP
        (
            clka => CLOCK,
            wea => writeMEM,
            addra => regREM,
            dina => regAC,
            douta => regMEM
        );

begin
    -- PC
    process (CLOCK, RESET)
        begin
            if (RESET ='1') then
                regPC <= "00000000";
            elsif (CLOCK'event and CLOCK='1') then -- na subida do clock
                if (cargaPC='1') then
                    regPC<= regRDM;
```

```
            elsif(incrementaPC='1') then
               regPC <= std_logic_vector(unsigned(regPC) + 1);
            else
               regPC <= regPC;
            end if;
         end if;
      end process;

   -- AC
   process (CLOCK, RESET)
      begin
         if (RESET='1') then
            regAC <= "00000000";
         elsif (CLOCK'event and CLOCK='1') then -- na subida do clock
            if (cargaAC='1') then
            regAC <= saidaULA;
            else
            regAC <= regAC;
            end if;
         end if;
      end process;

   -- ULA
   ULAX <= regAC;  -- recupera valor X do acumulador
   ULAY <= saidaMEM;  -- recupera valor Y de uma posicao da memória
   process(selULA, ULAX, ULAY)
      begin
         case selULA is
            when "000" => saidaULA <= (ULAX + ULAY);
            when "001" => saidaULA <= (ULAX AND ULAY);
            when "010" => saidaULA <= (ULAX OR ULAY);
            when "011" => saidaULA <= (NOT ULAX);
            when "100" => saidaULA <= ULAY;
            when others => saidaULA <= "00000000";
         end case;
      end process;
   -- FLAGS N E Z
   process (CLOCK, RESET)
      begin
         if (RESET='1') then
            regN <= '0';
            regZ <= '0';
            elsif(CLOCK'event and CLOCK='1') then -- na subida do clock
               if regAC = "00000000" then
                  regZ <= '1';
               else
                  regZ <= '0';
               end if;
               regN <= regAC(7); -- msb do registrador AC
```

```
            end if;
    end process;
-- REM
process(CLOCK, RESET)
    begin
        if (RESET='1') then
            regREM <= "00000000";
        elsif (CLOCK'event and CLOCK='1') then -- na subida do clock
            if (cargaREM ='1') then
                regREM <= saidaMUX;
            else
                regREM<= regREM;
            end if;
        end if;
    end process;
-- RDM
process (CLOCK, RESET)
    begin
        if (RESET='1') then
            regRDM <= "00000000";
        elsif (CLOCK'event and CLOCK='1') then
            if (cargaRDM='1') then
                regRDM <= regAC;
            else
                regRDM <= saidaMem;
            end if;
        end if;
    end process;
-- RI
process (CLOCK, RESET)
    begin
        if (RESET='1') then
            regRI<= "0000";
        elsif (CLOCK'event and CLOCK='1') then
            if (cargaRI='1') then
                regRI <= saidaMem(7 DOWNTO 4);
            else
                regRI <= regRI;
            end if;
        end if;
    end process
-- MUX2x1
process (selMUX, regPC, regRDM)
    begin
        if (selMUX = '0') then
            saidaMUX <= regPC;
        else
            saidaMUX <= regRDM;
        end if;
```

```
    end process;

end Behavioral;
```

Preencha:

Dados de Area do Datapath do Neander
FPGA device:
Numero de 4-LUTs:
Numero de ffps:
Numero de MULT e ADD DSP

FPGA: xc7s100fgga484-1 (Spartan-7)
LUTS: 9
FF: 17
DSP: 0