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

Aula Remota NEANDER – parte I

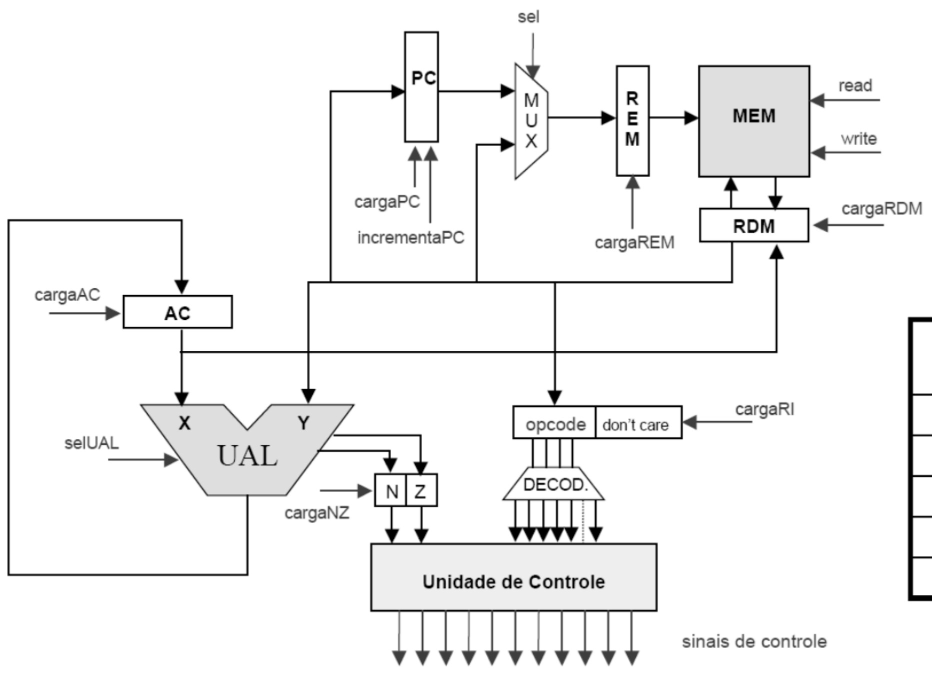
Felipe Bertoglio - 00262669

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;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;



entity neander is
   Port ( clk : in STD_LOGIC;
          out_MEM : in STD_LOGIC_VECTOR (7 downto 0);
          sel_MUX : in STD_LOGIC;
   load_PC : in STD_LOGIC;
          sel_ULA : in STD_LOGIC_VECTOR (2 downto 0);
          rst_PC, rst_AC, rst_RDM, rst_REM, rst_RI : in STD_LOGIC;
          ctrl_ULA, ctrl_PC, ctrl_AC, ctrl_REM, ctrl_RDM, ctrl_RI : in STD_LOGIC;
          reg_REM, reg_RI : out STD_LOGIC_VECTOR (7 downto 0);
          reg_N, reg_Z: out STD_LOGIC);
   end neander;

   architecture Behavioral of neander is
      signal X, Y, out_ULA, out_MUX : STD_LOGIC_VECTOR (7 downto 0);
      signal reg_PC, reg_AC, reg_RDM : STD_LOGIC_VECTOR (7 downto 0);


      begin

-- BRAM e FSM
   -- Não implementado;

-- Registradores, MUXs e ULA
         process(sel_ULA, X, Y, ctrl_ULA, reg_AC, reg_RDM)
         begin
            X <= reg_AC;
          Y <= reg_RDM;
          if ctrl_ULA = '1' then
             CASE sel_ULA is
                when "000" => out_ULA <= STD_LOGIC_VECTOR(signed(X) + signed(Y));
                when "001" => out_ULA <= X and Y;
                when "010" => out_ULA <= X or Y;
                when "011" => out_ULA <= not X;
                when "100" => out_ULA <= Y;
                when others => out_ULA <= X;
             end CASE;
             if out_ULA(7) = '1' then
                reg_N <= '1';
             else
                reg_N <= '0';
             end if;
             if out_ULA = "00000000" then
                reg_Z <= '1';
             else
                reg_Z <= '0';
             end if;
          else
             out_ULA <= X;
          end if;
       end process; -- ULA

       process(sel_MUX, reg_PC, reg_RDM)
       begin
          CASE sel_MUX is
             when '1' => out_MUX <= reg_PC;
             when '0' => out_MUX <= reg_RDM;
             when others => out_MUX <= reg_PC;
          end CASE;

       end process ; -- MUX


       process( ctrl_PC, rst_PC, load_PC, reg_RDM, clk)
       begin
          if ctrl_PC = '1' then
             if rst_PC = '1' then
                reg_PC <= "00000000";
             elsif(clk'event and clk = '1') then
                if (load_PC = '1') then
                   reg_PC <= reg_RDM;
                else
                   reg_PC <= STD_LOGIC_VECTOR(unsigned(reg_PC) + 1);
                end if ;
             end if;
```

```
        end if;
    end process; --reg_PC

    process( ctrl_AC, rst_AC,out_ULA, clk)
    begin
        if ctrl_AC = '1' then
            if rst_AC = '1' then
                reg_AC <= "00000000";
            elsif(clk'event and clk = '1') then
                reg_AC <= out_ULA;
            end if;
        end if;
    end process; --reg_AC

    process( ctrl_REM, rst_REM, out_MUX, clk)
    begin
        if ctrl_REM = '1' then
            if rst_REM = '1' then
                reg_REM <= "00000000";
            elsif(clk'event and clk = '1') then
                reg_REM <= out_MUX;
            end if;
        end if;
    end process; --reg_REM

    process( ctrl_RDM, rst_RDM, out_MEM, clk)
    begin
        if ctrl_RDM = '1' then
            if rst_RDM = '1' then
                reg_RDM <= "00000000";
            elsif(clk'event and clk = '1') then
                reg_RDM <= out_MEM;
            end if;
        end if;
    end process; --reg_RDM

    process( ctrl_RI, rst_RI, reg_RDM, clk)
    begin
        if ctrl_Ri = '1' then
            if rst_RI = '1' then
                reg_RI <= "00000000";
            elsif(clk'event and clk = '1') then
                reg_RI <= reg_RDM;
            end if;
        end if;
    end process; --reg_RI

end Behavioral;
```

Preencha:

Dados de Area do Datapath do Neander
FPGA device: Spartan 3e - XC3S100E
Numero de 4-LUTs: **56 out of   1,920   2%**
Numero de ffps: **40 out of   1,920   2%**

Numero de MULT e ADD DSP

```
Design Information
------------------
Command Line   : map -intstyle ise -p xc3s100e-cp132-5 -cm area -ir off -pr off
-c 100 -o neander_map.ncd neander.ngd neander.pcf
Target Device  : xc3s100e
Target Package : cp132
Target Speed   : -5
Mapper Version : spartan3e -- $Revision: 1.55 $
Mapped Date    : Wed Aug 03 19:56:54 2022

Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:           40 out of   1,920    2%
  Number of 4 input LUTs:               56 out of   1,920    2%
Logic Distribution:
  Number of occupied Slices:            37 out of     960    3%
    Number of Slices containing only related logic:      37 out of      37 100%
    Number of Slices containing unrelated logic:          0 out of      37   0%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:         56 out of   1,920    2%
  Number of bonded IOBs:                43 out of      83   51%
    IOB Latches:                         2
  Number of BUFGMUXs:                    2 out of      24    8%

Macro Statistics
# Adders/Subtractors                                  : 1
 8-bit adder                                          : 1
# Counters                                            : 1
 8-bit up counter                                     : 1
# Registers                                           : 32
 Flip-Flops                                           : 32
# Latches                                             : 2
 1-bit latch                                          : 2
```