UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

DISCIPLINA: SISTEMAS DIGITAIS

Prof. Fernanda Lima Kastensmidt

**2022-1**

**Trabalho 1 – Sistemas Digitais - Individual**

**Projeto do Processador Neander em VHDL**

**Aluno: Matheus Almeida Silva - 00316326**

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 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.**

1) Deve-se inserir a instrução de Subtração (SUB) conforme os modo de operandos da instrução ADD

e a instrução de XOR conforme o modelo de instrução da AND.

2) Programas a serem implementado no NEANDER na memoria embarcada BRAM (descreva .coe para inicializar a BRAM)

1. Soma de duas matrizes A e B 3x3 com dados de 8 bits, onde os dados das matrizes estão armazenados em memoria
2. Multiplicação de dois valores A e B por soma sucessiva
3. Programa a ser definido pelo aluno que use as instruções de subtração com no mínimo 10 instruções no total.
4. Programa que use a instrução de XOR com no mínimo 10 instruções no total.

\*\* **IMP: o endereço 0 da BRAM deve ter a instrução NOP. Logo a primeira instrução do programa estará no endereço 01 de BRAM.**

**TEMPLATE DE ENTREGA E APRESENTAÇÃO:**

1. Descrição do trabalho

O trabalho consiste em uma implementação do NEANDER utilizando a ferramenta ISE da Xilinx para descrever em VHDL o computador. Conforme a especificação do trabalho, foram adicionadas duas operações extras ao NEANDER original, sendo elas a operação XOR, com funcionalidade semelhante a AND existente e uma operação SUB de subtração, similar a ADD.

O datapath do NEANDER foi implementado previamente conforme a Parte 1 do trabalho e foram utilizados 4 programas de testes para a simulação posterior.

A descrição em VHDL recebeu como entrada os valores de clock e reset e tem como saída os sinais N, Z e a saída de dados que vai para a memória. A memória foi implementada como uma BRAM single port, de widht 8 e depth 256 conforme o computador original.

Para a Unidade de Controle se fez uma máquina de estados, com 10 estados, foram utilizados dois process a fim de otimizar o código, sendo um deles responsável por preservar o estado atual e prosseguir para o próximo e outro process para a implementação dos sinais de cada estados.

1. VHDL completo do Neander

-- Sistemas Digitais Para Computadores A

-- Matheus Almeida Silva - 00316326

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.std\_logic\_unsigned.ALL;

use IEEE.std\_logic\_arith.ALL;

entity neander is

Port ( clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

Z : out STD\_LOGIC;

N : out STD\_LOGIC;

outputData : out STD\_LOGIC\_VECTOR (7 downto 0));

end neander;

architecture Behavioral of neander is

COMPONENT memory

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;

-- Mux

signal selMux: std\_logic :='0';

signal outputMux: STD\_LOGIC\_VECTOR (7 downto 0);

--Decod

signal instruction: STD\_LOGIC\_VECTOR(15 downto 0);

signal decod: STD\_LOGIC\_VECTOR(3 downto 0);

-- REM

signal cargaREM : std\_logic := '0';

signal regREM: STD\_LOGIC\_VECTOR(7 downto 0);

signal outputREM: STD\_LOGIC\_VECTOR(7 downto 0);

-- ULA

signal selULA: std\_logic\_vector (2 downto 0) := "001";

signal input1ULA : STD\_LOGIC\_VECTOR(7 downto 0);

signal input2ULA : STD\_LOGIC\_VECTOR(7 downto 0);

signal outputULA: STD\_LOGIC\_VECTOR(7 downto 0);

signal regULA: STD\_LOGIC\_VECTOR(7 downto 0);

-- PC

signal cargaPC, incPC: std\_logic := '0';

signal regPC, outputPC: STD\_LOGIC\_VECTOR (7 downto 0);

--AC

signal cargaAC : std\_logic := '0';

signal inputAC: STD\_LOGIC\_VECTOR(7 downto 0);

signal outputAC: STD\_LOGIC\_VECTOR(7 downto 0);

--NZ

signal cargaNZ : std\_logic := '0';

signal regN: STD\_LOGIC;

signal regZ: STD\_LOGIC;

signal outputN: STD\_LOGIC;

signal outputZ: STD\_LOGIC;

--RI

signal cargaRI : std\_logic := '0';

signal inputRI: STD\_LOGIC\_VECTOR(7 downto 4);

signal outputRI: STD\_LOGIC\_VECTOR(7 downto 4);

-- Memoria

signal writeMem: STD\_LOGIC\_VECTOR(0 to 0) :="0";

signal outputMem: STD\_LOGIC\_VECTOR(7 downto 0);

-- Estados

type state\_type is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9);

signal state, nextState : state\_type;

begin

--Memoria

mem : memory

PORT MAP (

clka => clk,

wea => writeMem,

addra => outputREM,

dina => outputAC,

douta => outputMem

);

--Mux

process (selMux, outputPC, outputMem)

begin

if (selMux = '0') then

outputMux <= outputPC;

else

outputMux <= outputMem;

end if;

end process;

--PC

process (clk, rst)

begin

if rst ='1' then

regPC <= "00000000";

elsif (clk'event and clk='1') then

if (cargaPC='1') then

regPC<= outputMem;

elsif(incPC='1') then

regPC <= regPC + 1;

else

regPC <= regPC;

end if;

end if;

end process;

outputPC <= regPC;

--REM

process(clk, rst)

begin

if rst='1' then

regREM <= "00000000";

elsif (clk'event and clk='1') then

if (cargaREM ='1') then

regREM <= outputMux;

else

regREM<= regREM;

end if;

end if;

end process;

outputREM <= regREM;

--ULA

input1ULA <= outputAC;

input2ULA <= outputMem;

process(selULA, input1ULA, input2ULA)

begin

case selULA is

when "000" => regULA <= (input1ULA + input2ULA);

when "001" => regULA <= (input1ULA AND input2ULA);

when "010" => regULA <= (input1ULA OR input2ULA);

when "011" => regULA <= (NOT input1ULA);

when "100" => regULA <= input2ULA;

when "101" => regULA <= (input1ULA - input2ULA);

when "110" => regULA <= (input1ULA XOR input2ULA);

when others => regULA <= "00000000";

end case;

end process;

outputULA <= regULA;

--AC

process (clk, rst)

begin

if rst='1' then

inputAC <= "00000000";

elsif (clk'event and clk='1') then

if (cargaAC='1') then

inputAC <= outputULA;

else

inputAC <= inputAC;

end if;

end if;

end process;

outputAC <= inputAC;

--RI

process (clk, rst)

begin

if rst='1' then

inputRI<= "0000";

elsif (clk'event and clk='1') then

if (cargaRI='1') then

inputRI <= outputMem(7 DOWNTO 4);

else

inputRI <= inputRI;

end if;

end if;

end process;

outputRI <= inputRI;

--NZ

process (clk, rst)

begin

if rst='1' then

regN <= '0';

regZ <= '0';

elsif(clk'event and clk='1') then

if (cargaNZ='1') then

if outputAC = "00000000" then

regZ <= '1';

else

regZ <= '0';

end if;

regN <= outputAC(7);

end if;

end if;

end process;

outputZ <= regZ;

outputN <= regN;

--DECOD

decod <= outputRI (7 downto 4);

process(decod)

begin

instruction <= "0000000000000000";

case decod is

when "0000" => instruction(0) <= '1'; --NOP

when "0001" => instruction(1) <= '1'; --STA

when "0010" => instruction(2) <= '1'; --LDA

when "0011" => instruction(3) <= '1'; --ADD

when "0100" => instruction(4) <= '1'; --OR

when "0101" => instruction(5) <= '1'; --AND

when "0110" => instruction(6) <= '1'; --NOT

when "0111" => instruction(7) <= '1'; --NOP

when "1000" => instruction(8) <= '1'; --JMP

when "1001" => instruction(9) <= '1'; --JZ

when "1010" => instruction(10) <= '1'; --JN

when "1011" => instruction(11) <= '1'; --NOP

when "1100" => instruction(12) <= '1'; --SUB

when "1101" => instruction(13) <= '1'; --XOR

when "1110" => instruction(14) <= '1'; --NOP

when "1111" => instruction(15) <= '1'; --HLT

when others => instruction <= "0000000000000000";

end case;

end process;

--FSM

process(rst, clk)

begin

if rst = '1' then

state <= S0;

elsif(clk'event and clk='1') then

state <= nextState;

end if;

end process;

-- UC

process(instruction, state, nextState, outputZ, outputN, outputMem)

begin

cargaAC <='0';

cargaNZ <='0';

cargaPC <='0';

selULA <="000";

cargaRI <='0';

incPC <= '0';

writeMem <= "0";

selMUX <= '0';

cargaREM <='0';

case state is

when S0=>

cargaREM <='1';

nextState <=S1;

when S1=>

cargaREM <='0';

incPC<='1';

nextState <= S2;

when S2=>

incPC<='0';

cargaRI <='1';

nextState<=S3;

when S3=>

incPC <='0';

cargaRI<='0';

if instruction(6) = '1' then --NOT

selULA <= "011";

cargaAC <= '1';

cargaNZ <='1';

nextState <= S0;

elsif(instruction(9) = '1' and outputZ = '0') then --JZ para Z=0

incPC <= '1';

nextState <=S0;

elsif(instruction(10) = '1' and outputN = '0') then --JN para =0

incPC <='1';

nextState <=S0;

elsif(instruction(0) = '1') then --NOP

nextState <= S0;

elsif(instruction(15) = '1') then --HALT

incPC <= '0';

nextState <= S9;

else

selMux <= '0';

cargaREM <= '1';

nextState <= S4;

end if;

when S4 =>

selMux <= '0';

incPC <= '0';

cargaAC <= '0';

cargaNZ <='0';

cargaREM <= '0';

if(instruction(1) = '1' OR instruction(2) = '1' OR instruction(3) = '1' OR instruction(4) = '1' OR instruction(5) = '1' OR instruction(12) = '1' OR instruction(13) = '1') then

incPC <= '1';

end if;

nextState <= S5;

when S5 =>

incPC <= '0';

if(instruction(1) = '1' OR instruction(2) = '1' OR instruction(3) = '1' OR instruction(4) = '1' OR instruction(5) = '1' OR instruction(12) = '1' OR instruction(13) = '1') then

selMux <= '1';

cargaREM <= '1';

nextState <= S6;

else

cargaPC <= '1';

nextState <= S0;

end if;

when S6 =>

incPC <= '0';

selMux <='0';

cargaREM <= '0';

cargaPC <='0';

nextState <= S7;

when S7 =>

if instruction(1) ='1' then

writeMEM <= "1";

elsif instruction(2) ='1' then

selULA <= "100";

elsif instruction(3) ='1' then

selULA <= "000";

elsif instruction(4) ='1' then

selULA <= "010";

elsif instruction(5) ='1' then

selULA <= "001";

elsif instruction(12) ='1' then

selULA <= "101";

elsif instruction(13) ='1' then

selULA <= "110";

end if;

if instruction(1) = '1' then

nextState <= S0;

else

cargaAC <= '1';

cargaNZ <= '1';

nextState <= S0;

end if;

when S9 =>

nextState <=S9;

when others =>

nextState <= S0;

end case;

end process;

Z <= outputZ;

N <= outputN;

outputData <= outputMem;

end Behavioral;

1. Testbench VHDL completo

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\_testbench IS

END neander\_testbench;

ARCHITECTURE behavior OF neander\_testbench IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT neander

PORT(

clk : IN std\_logic;

rst : IN std\_logic;

Z : OUT std\_logic;

N : OUT std\_logic;

outputData : OUT std\_logic\_vector(7 downto 0)

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0';

signal rst : std\_logic := '0';

--Outputs

signal Z : std\_logic;

signal N : std\_logic;

signal outputData : std\_logic\_vector(7 downto 0);

-- Clock period definitions

constant clk\_period : time := 20 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: neander PORT MAP (

clk => clk,

rst => rst,

Z => Z,

N => N,

outputData => outputData

);

-- Clock process definitions

clk\_process :process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

-- Stimulus process

stim\_proc: process

begin

-- hold reset state for 100 ns.

rst <= '1';

wait for clk\_period\*5;

rst <= '0';

wait for clk\_period\*5;

wait;

end process;

END;

1. Explicação e descrição das aplicações em Assembly

Programa 1 – Soma de Matrizes 3x3

O primeiro programa consiste em uma soma de matrizes 3x3, sendo basicamente uma sequência de somas para os valores da matriz na memória, que estão dispostas como uma lista. Assim o primeiro valor(linha 1, coluna 1) da matriz m é somado com o primeiro valor da matriz n(linha 1, coluna 1) e armazenado na matriz r na mesma posição. Abaixo segue o código no Hidra e parte do coe resultante.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente com confiança média

Texto

Descrição gerada automaticamente com confiança média

Programa 2 – Multiplicação Por Somas Sucessivas

O segundo programa consiste em uma multiplicação por somas sucessivas utilizando um loop e valores auxiliares de controle. Abaixo segue o código no Hidra e parte do coe resultante.

Interface gráfica do usuário, Aplicativo, Tabela

Descrição gerada automaticamente com confiança média

Texto

Descrição gerada automaticamente

Programa 3 – Subtração de Matrizes 2x2

De maneira análoga ao programa 1, o programa 3 realiza a operação de subtração entre duas matrizes. Abaixo segue o código no Hidra e parte do coe resultante

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Programa 4 – Subtração e Xor de Matrizes 2x2

Praticamente igual ao Programa 3 com o detalhe de que após a operação de subtração é realizado um xor entre o valor resultante e o da matriz n. Abaixo segue o código no Hidra e parte do coe resultante.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente

1. Simulações sem e com atraso com detalhes e flechas mostrando inicio meio e final do programa e resultados

Programa 1 – Soma de Matrizes 3x3

Tela de computador

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente

Programa 2 – Multiplicação Por Somas Sucessivas

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

Programa 3 – Subtração de Matrizes 2x2

Interface gráfica do usuário

Descrição gerada automaticamente

Tela de computador

Descrição gerada automaticamente

Programa 4 – Subtração e Xor de Matrizes 2x2

Interface gráfica do usuário

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente

1. Dados de área, tempo de execução em ciclos de relógio e tempo em segundos deve ser apresentado dado um determinado clock usado.

|  |  |  |  |
| --- | --- | --- | --- |
| Programa | Numero de Instruções Executadas | Tempo de execução em # de ciclos de relógio (c.c.) | Tempo de execução em Segundos  (Neander operando a 50 MHz) |
| Soma de matrizes | 29 | 228 | 4560e-9s |
| Multiplicação por somas sucessivas | 13 | 70 | 1440e-9s |
| Programa com SUB | 14 | 108 | 2160e-9s |
| Programa com XOR | 18 | 140 | 2800e-9s |

**Dados de Area do Neander**

FPGA device: xc3s100e-5cp132

Numero de 4-LUTs: 102

Numero de ffps: 38

Numero de BRAM: 1

Numero de DSP: 1

Tabela

Descrição gerada automaticamente

1. **(1 ponto extra)** Se o Neander for prototipado na placa de prototipação, mostrar vídeos do funcionamento mostrando dados da memoria do Neander (debugger com memoria BRAM dual port, chaves para controlar os endereços de memória e display 7seg para mostrar os resultados).

**Não implementado.**