UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

DISCIPLINA: SISTEMAS DIGITAIS

Prof. Fernanda Lima Kastensmidt

**2022-1**

**Trabalho 1 – Sistemas Digitais – Individual**

**Aluno: Giordano Souza de Paula – Cartão: 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 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.**

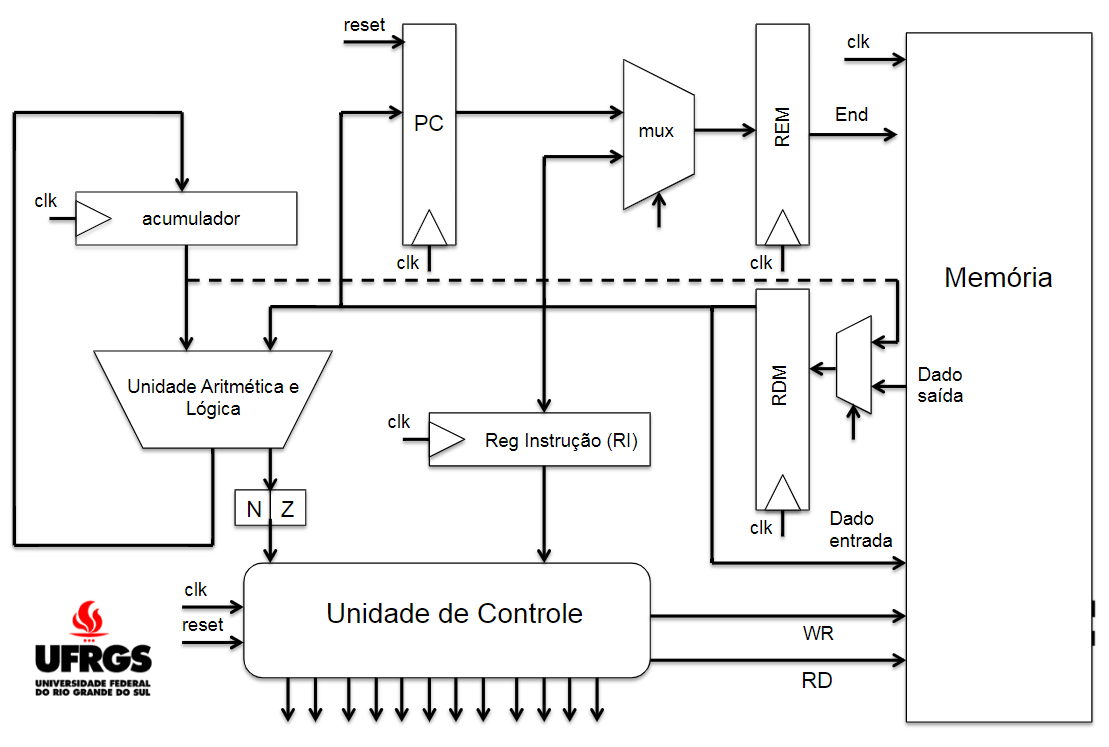
**TEMPLATE DE ENTREGA E APRESENTAÇÃO:**

1. Descrição do trabalho

O objetivo deste trabalho é empregar os conceitos aprendidos na disciplina de Sistemas Digitais ao implementar o processador didático Neander, definido no livro *Fundamentos de arquitetura de computadores* do professor Raul Weber usando a linguagem de descrição de hardware estudada – VHDL – com o acréscimo das instruções a nível de bits SUB A (subtração do valor definido no endereço A do valor atual do acumulador) e XOR A (ou exclusivo do valor definido no endereço A, com o valor atual do acumulador).

Para fins de teste de funcionalidade, foram implementados quatro programas, conforme descrito no tópico 2 do enunciado deste trabalho, sendo: soma de duas matrizes 3x3, multiplicação de dois valores A e B, subtração de dois valores A e B, ou exclusivo dentre os valores A e B.

O processador Neander tem o seguinte diagrama de blocos:



(Retirado do slide 8 das aulas 11 e 12, arquivo neander.pdf – 2019-2 – Prof. Fernanda L. Kastensmidt)

A largura de dados, endereços, registradores e apontador de instruções desse processador são de 8 bits, podendo armazenar números de decimais sem sinal de 0 a até 255. Além disso, o espaço que cada instrução ocupa em um byte de memória é de 4 bits, sendo os 4 bits mais significativos, enquanto os outros 4 bits nesse processador encontram-se sem função (no processador Ahmes, também idealizado pelo professor Raul Weber, esses são utilizados para definir o modo de endereçamento a ser utilizado em cada instrução).

1. VHDL completo do Neander

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

--use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

--use IEEE.STD\_LOGIC\_ARITH.ALL;

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 (2 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';

-- DECOD

signal instrucao: std\_logic\_vector(15 downto 0);

signal decod: std\_logic\_vector(3 downto 0);

-- MEMORY

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

signal writeMem: std\_logic\_vector(0 DOWNTO 0);

-- STATES

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

signal estado\_atual, proximo\_estado: state\_type;

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 => writeMEM,

addra => regREM,

dina => regAC,

douta => memOut

);

-- 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 <= memOut; -- recupera valor Y de uma posicao da memória

process(selULA, ULAX, ULAY)

begin

case selULA is

when "000" => saidaULA <= std\_logic\_vector(signed(ULAX) + signed(ULAY)); -- operacao ADD

when "001" => saidaULA <= (ULAX AND ULAY); -- operacao AND

when "010" => saidaULA <= (ULAX OR ULAY); -- operacao OR

when "011" => saidaULA <= (NOT ULAX); -- operacao NOT

when "100" => saidaULA <= ULAY; -- operacao NOP

when "110" => saidaULA <= std\_logic\_vector(signed(ULAX) - signed(ULAY)); -- operacao SUB

when "111" => saidaULA <= (ULAX XOR ULAY); -- operacao XOR

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 <= memOut;

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 <= memOut(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;

-- DECOD

decod <= regRDM(7 downto 4); -- bits mais significativos

process(decod)

begin

instrucao <= "0000000000000000";

case decod is

when "0000" => instrucao(0) <= '1'; -- instrucao 00 NOP

when "0001" => instrucao(1) <= '1'; -- instrucao 16 STA

when "0010" => instrucao(2) <= '1'; -- instrucao 32 LDA

when "0011" => instrucao(3) <= '1'; -- instrucao 48 ADD

when "0100" => instrucao(4) <= '1'; -- instrucao 64 OR

when "0101" => instrucao(5) <= '1'; -- instrucao 80 AND

when "0111" => instrucao(6) <= '1'; -- instrucao 96 NOT

when "1000" => instrucao(8) <= '1'; -- instrucao 128 JMP

when "1001" => instrucao(9) <= '1'; -- instrucao 144 JN

when "1010" => instrucao(10) <= '1'; -- instrucao 160 JZ

when "1011" => instrucao(11) <= '1'; -- instrucao 176 SUB

when "1100" => instrucao(12) <= '1'; -- instrucao 192 XOR

when "1101" => instrucao(13) <= '1'; -- instrucao 208 NOP - nao utilizado

when "1110" => instrucao(14) <= '1'; -- instrucao 224 NOP - nao utilizado

when "1111" => instrucao(15) <= '1'; -- instrucao 240 HLT

when others => instrucao <= "0000000000000000";

end case;

end process;

-- NEANDER FSM - maquina de estados mealy

process(CLOCK, RESET)

begin

if(RESET = '1') then

estado\_atual <= S0;

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

estado\_atual <= proximo\_estado;

end if;

end process;

-- UNIDADE DE CONTROLE

process(memOut, instrucao, proximo\_estado, regZ, regN)

begin

-- reseta

cargaAC <= '0';

cargaPC <= '0';

incrementaPC <= '0';

cargaNZ <= '0';

cargaRDM <= '0';

cargaREM <= '0';

writeMem <= "0";

selMUX <= '0';

selULA <= "000";

case estado\_atual is

when S0 =>

cargaRDM <= '1';

proximo\_estado <= S1;

when S1 =>

cargaREM <= '0';

incrementaPC <= '0';

proximo\_estado <= S2;

when S2 =>

incrementaPC <= '0';

cargaRDM <= '1';

proximo\_estado <= S3;

when S3 =>

incrementaPC <= '0';

cargaRDM <= '0';

if(instrucao(0) = '1') then --NOP

proximo\_estado <= S0;

elsif(instrucao(6) = '1') then --NOT

selULA <= "011";

cargaNZ <= '1';

cargaAC <= '1';

elsif(instrucao(9) = '1' and regN = '0') then --JN quando falso

incrementaPC <= '1';

proximo\_estado <= S0;

elsif(instrucao(10) = '1' and regZ = '0') then --JZ quando falso

incrementaPC <= '1';

proximo\_estado <= S0;

elsif(instrucao(15) = '1') then --HLT

incrementaPC <= '0';

proximo\_estado <= S8;

else

selMUX <= '0';

cargaREM <= '1';

proximo\_estado <= S4;

end if;

when S4 =>

selMUX <= '0';

incrementaPC <= '0';

cargaAC <= '0';

cargaNZ <= '0';

cargaREM <= '0';

if(instrucao(1) = '1'

or instrucao(2) = '1'

or instrucao(3) = '1'

or instrucao(4) = '1'

or instrucao(5) = '1'

or instrucao(11) = '1'

or instrucao(12) = '1') then

incrementaPC <= '1';

end if;

proximo\_estado <= S5;

when S5 =>

incrementaPC <= '0';

if(instrucao(1) = '1'

or instrucao(2) = '1'

or instrucao(3) = '1'

or instrucao(4) = '1'

or instrucao(5) = '1'

or instrucao(11) = '1'

or instrucao(12) = '1') then

selMUX <= '1';

cargaREM <= '1';

proximo\_estado <= S6;

else

cargaPC <= '1';

proximo\_estado <= S0;

end if;

when S6 =>

incrementaPC <= '0';

selMUX <= '0';

cargaREM <= '0';

cargaPC <= '0';

proximo\_estado <= S7;

when S7 =>

if(instrucao(1) = '1') then

writeMEM <= "1";

elsif(instrucao(2) = '1') then

selULA <= "100";

elsif(instrucao(3) = '1') then

selULA <= "000";

elsif(instrucao(4) = '1') then

selULA <= "010";

elsif(instrucao(5) = '1') then

selULA <= "001";

elsif(instrucao(11) = '1') then

selULA <= "101";

elsif(instrucao(12) = '1') then

selULA <= "110";

end if;

when S8 => --HLT

proximo\_estado <= S8;

when others =>

proximo\_estado <= S0;

end case;

end process;

DOUT <= saidaAC;

N <= regN;

Z <= regZ;

end Behavioral;

1. Testbench VHDL completo

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY tb\_neander is

END tb\_neander;

ARCHITECTURE behavior OF tb\_neander is

COMPONENT neander

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 COMPONENT;

--Inputs

signal CLOCK : STD\_LOGIC := '0';

signal RESET : STD\_LOGIC := '0';

--Outputs

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

signal N : STD\_LOGIC;

signal Z : STD\_LOGIC;

-- Clock period definitions

constant CLOCK\_TICK : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: neander PORT MAP (

CLOCK => CLOCK,

RESET => RESET,

DOUT => DOUT,

N => N,

Z => Z

);

-- Clock process definitions

clk\_process : process

begin

CLOCK <= '0';

wait for CLOCK\_TICK/2;

CLOCK <= '1';

wait for CLOCK\_TICK/2;

end process;

-- Stimulus process

stim\_proc: process

begin

-- hold reset state for 100 ns.

RESET <= '1';

wait for CLOCK\_TICK \* 10;

RESET <= '0';

wait for CLOCK\_TICK \* 10;

wait;

end process;

END;

1. Explicação e descrição das aplicações em Assembly

;Aplicação 1 - Soma de matrizes 3x3

;A matriz 3x3 A é somada com a matriz 3x3 B e o resultado é colocado na matriz R

;Nesse exemplo a matriz A é inicializada com [1 1 1], a matriz B com [2 2 2] resultando na

[1 1 1] [2 2 2]

[1 1 1] [2 2 2]

matriz R, com valores [3 3 3]

[3 3 3]

[3 3 3].

;CONSTANTES - No final do programa, logo depois da instrução HALT

ORG 56

index:

DB 9

matrizA11:

DB 1

matrizA12:

DB 1

matrizA13:

DB 1

matrizA21:

DB 1

matrizA22:

DB 1

matrizA23:

DB 1

matrizA31:

DB 1

matrizA32:

DB 1

matrizA33:

DB 1

matrizB11:

DB 2

matrizB12:

DB 2

matrizB13:

DB 2

matrizB21:

DB 2

matrizB22:

DB 2

matrizB23:

DB 2

matrizB31:

DB 2

matrizB32:

DB 2

matrizB33:

DB 2

matrizR11:

DB 2

matrizR12:

DB 2

matrizR13:

DB 2

matrizR21:

DB 2

matrizR22:

DB 2

matrizR23:

DB 2

matrizR31:

DB 2

matrizR32:

DB 2

matrizR33:

DB 2

fim:

HLT

;PROGRAMA PRINCIPAL

ORG 0

;O programa deve começar com instruções NOP

NOP

NOP

LDA matrizA11

ADD matrizB11

STA matrizR11

LDA matrizA12

ADD matrizB12

STA matrizR12

LDA matrizA13

ADD matrizB13

STA matrizR13

LDA matrizA21

ADD matrizB21

STA matrizR21

LDA matrizA22

ADD matrizB22

STA matrizR22

LDA matrizA23

ADD matrizB23

STA matrizR23

LDA matrizA31

ADD matrizB31

STA matrizR31

LDA matrizA32

ADD matrizB32

STA matrizR32

LDA matrizA33

ADD matrizB33

STA matrizR33

Arquivo .coe gerado:

memory\_initialization\_radix=10;

memory\_initialization\_vector=0,0,32,26,16,25,32,25,160,29,48,23,16,25,32,28,48,27,16,28,128,6,0,255,2,0,2,3,0,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;

------------------------------------------------------------------------------------------------------------------

;Aplicação 2 - Multiplicação de dois valores 1 e 2, resultado armazenado no espaço resultado

;Nesse exemplo, o valor 1 é inicializado como 2, e o valor 2 é inicializado como 3, então o resultado armazenado é 6 (no endereço 22)

;ESPACO DE DADOS

ORG 22

zero:

DB 0

menos1:

DB -1

dois:

DB 2

loopindex:

DB 0

valor1:

DB 2 ;exemplo

valor2:

DB 3 ;exemplo

resultado:

DB 0

fim:

HLT

;PROGRAMA PRINCIPAL

ORG 0

NOP

NOP

LDA valor1

STA loopindex

somaloop:

LDA loopindex

JZ fim

ADD menos1

STA loopindex

LDA resultado

ADD valor2

STA resultado

JMP somaloop

Arquivo .coe gerado:

memory\_initialization\_radix=10;

memory\_initialization\_vector=0,0,32,26,16,25,32,25,160,29,48,23,16,25,32,28,48,27,16,28,128,6,0,255,2,0,2,3,0,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;

------------------------------------------------------------------------------------------------------------------

;Aplicação 3 - Divisão de dois valores A por B, resultado armazenado em R, e resto armazenado em C

;Nesse exemplo, o valor A é inicializado como 10,

;o valor B é inicializado como 5,

;então o resultado armazenado em R é 2

;e resto armazenado segue zero.

;ESPACO DE DADOS

ORG 40

zero:

DB 0

cte\_1:

DB 1

menos1:

DB -1

dois:

DB 2

temp:

DB 0

A:

DB 10

B:

DB 2

R:

DB 0

C:

DB 0

resto:

DB 0

fim:

HLT

;PROGRAMA PRINCIPAL

ORG 0

NOP

NOP

LDA A

STA temp

loop\_div:

LDA temp

JZ fim

SUB B; instrução SUB criada

JN negativo

JZ terminou\_div

STA temp

LDA R

ADD cte\_1

STA R

JMP loop\_div

terminou\_div:

; resto zero, encerra a divisão

LDA R

ADD cte\_1

STA R

JMP fim

negativo:

;recupera o valor anterior

ADD B

STA C

JMP fim

Arquivo .coe gerado:

memory\_initialization\_radix=10;

memory\_initialization\_vector=0,0,32,45,16,44,32,44,160,50,176,46,144,34,160,26,16,44,32,47,48,41,16,47,128,6,32,47,48,41,16,47,128,50,48,46,16,48,128,50,0,1,255,2,0,10,2,0,0,0,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;

------------------------------------------------------------------------------------------------------------------

;Aplicação 4 - Verificação de validade da instrução XOR com um NANDs dentre os valores A e B, validade armazenada em V

;Nesse exemplo, o valor A é inicializado como 255,

;o valor B é inicializado como 112,

;o resultado obtido pelas operações nand é armazenado em R\_nands

;então o resultado obtido pelo XOR é armazenado em R\_xor, valendo 143.

;O resultado de R\_nands e R\_xor são comparados usando a instrução sub

;caso iguais, V vale 1, caso contrário, V vale zero.

;Nesse caso é 1 (verdadeiro).

;ESPACO DE DADOS

ORG 48

zero:

DB 0

cte\_1:

DB 1

menos1:

DB -1

dois:

DB 2

temp:

DB 0

temp2:

DB 0

A:

DB 255

B:

DB 112

V:

DB 0

R\_nands:

DB 0

R\_xor:

DB 0

C:

DB 0

fim:

HLT

;PROGRAMA PRINCIPAL

ORG 0

NOP

NOP

LDA A

AND B

NOT

STA temp

AND A

NOT

STA temp2

LDA temp

AND B

NOT

STA temp

AND temp2

NOT

STA R\_nands

LDA A

XOR B ; instrucao XOR criada

STA R\_xor

SUB R\_nands ; instrucao sub criada

JZ valido

JMP invalido

valido:

LDA cte\_1

STA V

JMP fim

invalido:

LDA zero

STA V

JMP fim

Arquivo .coe gerado:

memory\_initialization\_radix=10;

memory\_initialization\_vector=0,0,32,54,80,55,96,16,52,80,54,96,16,53,32,52,80,55,96,16,52,80,53,96,16,57,32,54,176,16,58,192,160,36,128,42,32,49,16,56,128,60,32,48,16,56,128,60,0,1,255,2,0,0,255,112,0,0,0,0,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;

1. Simulações sem e com atraso com detalhes e flechas mostrando inicio meio e final do programa e resultados
2. 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 |  |  |  |
| Multiplicação por somas sucessivas |  |  |  |
| Programa com SUB |  |  |  |
| Programa com XOR |  |  |  |

**Dados de Area do Neander**

FPGA device:

Numero de 4-LUTs:

Numero de ffps:

Numero de BRAM:

Numero de DSP

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).