Trabalho 1 – Pedro Company Beck - 324055

1) Descrição do trabalho

Este trabalho visa implementar o processador hipotético Neander em um circuito VHDL, além disso são adicionadas as instruções SUB e XOR para o conjunto de instruções do processador e foram escritos três programas para testar a implementação do processador. Todos os arqiuvos fonte e as imagens incluídas nesse relatório estão em anexo.

2) VHDL completo do Neander

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Pedro Company Beck
--
-- Create Date:    00:46:28 07/20/2023
-- Design Name:
-- Module Name:    neander - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
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 (
		rst: in STD_LOGIC;
		clk: in STD_LOGIC;
		outHalt: out STD_LOGIC;
		outPC: out STD_LOGIC_VECTOR(7 downto 0);
```

```
                outAC: out STD_LOGIC_VECTOR(7 downto 0);
                outRI: out STD_LOGIC_VECTOR(3 downto 0)
        );
end neander;

architecture Behavioral of neander is

COMPONENT mem1
  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;

type states is (t0, t1, t2, t3, t4, t5, t6, t7);
signal state: states;

signal incPC, cargaPC: STD_LOGIC;
signal saidaPC: STD_LOGIC_VECTOR(7 downto 0);

signal s1MPX: STD_LOGIC;
signal saidaMPX: STD_LOGIC_VECTOR(7 downto 0);

signal cargaREM: STD_LOGIC;
signal saidaREM: STD_LOGIC_VECTOR(7 downto 0);
signal entradaMEM: STD_LOGIC_VECTOR(7 downto 0);

signal readMEM: STD_LOGIC;
signal writeMEM: STD_LOGIC_VECTOR(0 downto 0);
signal saidaMEM: STD_LOGIC_VECTOR(7 downto 0);

signal cargaRI: STD_LOGIC;
signal saidaRI: STD_LOGIC_VECTOR(3 downto 0);

signal cargaAC: STD_LOGIC;
signal saidaAC: STD_LOGIC_VECTOR(7 downto 0);

signal cargaNZ: STD_LOGIC;
signal saidaNZ: STD_LOGIC_VECTOR(1 downto 0);

signal selALU: STD_LOGIC_VECTOR(2 downto 0);
signal saidaALU: STD_LOGIC_VECTOR(7 downto 0);
signal nzALU: STD_LOGIC_VECTOR(1 downto 0);

begin
```

```
memoria : mem1
  PORT MAP (
    clka => clk,
    wea => writeMEM,
    addra => saidaREM,
    dina => entradaMEM,
    douta => saidaMEM
  );

outPC <= saidaPC;
outAC <= saidaAC;
outRI <= saidaRI;

process(clk, rst) -- state machine
begin
        if rst = '1' then
                state <= t0;
        elsif clk'event and clk = '1' then
                case state is
                        when t0 =>
                                state <= t1;
                        when t1 =>
                                state <= t2;
                        when t2 =>
                                state <= t3;
                        when t3 =>
                                if (saidaRI = "0000") -- NOP
                                or (saidaRI = "0110") -- NOT
                                or (saidaRI = "1001" and saidaNZ(1) = '0') -- JN, N=0
                                or (saidaRI = "1010" and saidaNZ(0) = '0') -- JZ, Z=0
                                then
                                        state <= t0;
                                elsif (saidaRI = "0001") -- STA
                                  or (saidaRI = "0010") -- LDA
                                        or (saidaRI = "0011") -- ADD
                                        or (saidaRI = "0100") -- OR
                                        or (saidaRI = "0101") -- AND
                                        or (saidaRI = "1000") -- JMP
                                        or (saidaRI = "1001" and saidaNZ(1) = '1') -- JN, N=1
                                        or (saidaRI = "1010" and saidaNZ(0) = '1') -- JZ, Z=1
                                        or (saidaRI = "1011") -- XOR
                                        or (saidaRI = "1100") -- SUB
                                        then
                                        state <= t4;
                                end if;
                        when t4 =>
                                state <= t5;
                        when t5 =>
                                if (saidaRI = "0001") -- STA
```

```
                                or (saidaRI = "0010") -- LDA
                                or (saidaRI = "0011") -- ADD
                                or (saidaRI = "0100") -- OR
                                or (saidaRI = "0101") -- AND
                                or (saidaRI = "1011") -- XOR
                                or (saidaRI = "1100") -- SUB
                                then
                                        state <= t6;

                                elsif (saidaRI = "1000") -- JMP
                                        or (saidaRI = "1001" and saidaNZ(1) = '1') -- JN, N=1
                                        or (saidaRI = "1010" and saidaNZ(0) = '1') -- JZ, Z=1
                                        then
                                        state <= t0;

                                end if;
                        when t6 =>
                                state <= t7;
                        when t7 =>
                                state <= t0;
                        when others =>
                                state <= state;
                end case;
        end if;
end process;

process(state, saidaNZ, saidaRI, saidaAC) -- fsm
begin
        incPC    <= '0';
        s1MPX    <= '0';
        readMEM  <= '0';
        writeMEM <= "0";
        cargaPC  <= '0';
        cargaREM <= '0';
        cargaRI  <= '0';
        cargaAC  <= '0';
        cargaNZ  <= '0';
        selALU   <= "111";
        entradaMEM <= "00000000";
        outHalt <= '0';

        case state is
                when t0 =>
                        s1MPX    <= '0';
                        cargaREM <= '1';

                when t1 =>
                        readMEM <= '1';
                        incPC   <= '1';
```

```
when t2 =>
        cargaRI <= '1';

when t3 =>
        if (saidaRI = "0001") -- STA
        or (saidaRI = "0010") -- LDA
        or (saidaRI = "0011") -- ADD
        or (saidaRI = "0100") -- OR
        or (saidaRI = "0101") -- AND
        or (saidaRI = "1000") -- JMP
        or (saidaRI = "1001" and saidaNZ(1) = '1') -- JN, N=1
        or (saidaRI = "1010" and saidaNZ(0) = '1') -- JZ, Z=1
        or (saidaRI = "1011") -- XOR
        or (saidaRI = "1100") -- SUB
        then
                s1MPX <= '0';
                cargaREM <= '1';

        elsif saidaRI = "0110" then -- NOT
                selALU  <= "011"; -- not X
                cargaAC <= '1';
                cargaNZ <= '1';

        elsif (saidaRI = "1001" and saidaNZ(1) = '0') -- JN, N=0
                or (saidaRI = "1010" and saidaNZ(0) = '0') -- JZ, Z=0
                then
                incPC  <= '1';

        elsif saidaRI = "1111" then -- HLT
                outHalt <= '1';
        else
        end if;

        when t4 =>
                if (saidaRI = "0001") -- STA
                or (saidaRI = "0010") -- LDA
                or (saidaRI = "0011") -- ADD
                or (saidaRI = "0100") -- OR
                or (saidaRI = "0101") -- AND
                or (saidaRI = "1011") -- XOR
                or (saidaRI = "1100") -- SUB
                then
                        readMEM <= '1';
                        incPC   <= '1';

                elsif (saidaRI = "1000") -- JMP
                  or (saidaRI = "1001" and saidaNZ(1) = '1') -- JN, N=1
                        or (saidaRI = "1010" and saidaNZ(0) = '1') -- JZ, Z=1
```

```
                then
                readMEM <= '1';
        else
        end if;

when t5 =>
        if (saidaRI = "0001") -- STA
        or (saidaRI = "0010") -- LDA
        or (saidaRI = "0011") -- ADD
        or (saidaRI = "0100") -- OR
        or (saidaRI = "0101") -- AND
        or (saidaRI = "1011") -- XOR
        or (saidaRI = "1100") -- SUB
        then
                s1MPX    <= '1';
                cargaREM <= '1';

        elsif (saidaRI = "1000") -- JMP
                or (saidaRI = "1001" and saidaNZ(1) = '1') -- JN, N=1
                or (saidaRI = "1010" and saidaNZ(0) = '1') -- JZ, Z=1
                then
                cargaPC <= '1';
        else
        end if;

when t6 =>
        if (saidaRI = "0001") then -- STA
                -- cargaRDM <= 1
        else -- LDA, ADD, OR, AND, XOR, SUB
                readMEM <= '1';
        end if;

when t7 =>
        if (saidaRI = "0001") then -- STA
                writeMEM <= "1";
                entradaMEM <= saidaAC;
        elsif (saidaRI = "0010") then -- LDA
                selALU  <= "100"; -- Y
                cargaAC <= '1';
                cargaNZ <= '1';
        elsif (saidaRI = "0011") then -- ADD
                selALU  <= "000";
                cargaAC <= '1';
                cargaNZ <= '1';
        elsif (saidaRI = "0100") then -- OR
                selALU  <= "010"; -- X or Y
                cargaAC <= '1';
                cargaNZ <= '1';
        elsif (saidaRI = "0101") then -- AND
```

```
                                        selALU  <= "001"; -- X and Y
                                        cargaAC <= '1';
                                        cargaNZ <= '1';
                                elsif (saidaRI = "1011") then -- XOR
                                        selALU <= "110"; -- X xor Y
                                        cargaAC <= '1';
                                        cargaNZ <= '1';
                                elsif (saidaRI = "1100") then -- SUB
                                        selALU <= "101"; -- X - Y
                                        cargaAC <= '1';
                                        cargaNZ <= '1';
                                else
                                end if;
                        when others =>

        end case;
end process;

process(clk, rst) -- PC
begin
        if rst = '1' then
                saidaPC <= "00000000";
        elsif clk'event and clk = '1' then
                if cargaPC = '1' then
                        saidaPC <= saidaMEM;
                else
                        if incPC = '1' then
                                saidaPC <= STD_LOGIC_VECTOR(1 + unsigned(saidaPC));
                        else
                                saidaPC <= saidaPC;
                        end if;
                end if;
        else
        end if;
end process;

process(clk) -- AC
begin
        if clk'event and clk = '1' then
                if cargaAC = '1' then
                        saidaAC <= saidaALU;
                else
                        saidaAC <= saidaAC;
                end if;
        end if;
end process;

process(s1MPX, saidaPC, saidaMEM) -- MPX
begin
```

```
	if s1MPX = '0' then
		saidaMPX <= saidaPC;
	else
		saidaMPX <= saidaMEM;
	end if;
end process;

process(clk) -- REM
begin
	if clk'event and clk = '1' then
		if cargaREM = '1' then
			saidaREM <= saidaMPX;
		else
			saidaREM <= saidaREM;
		end if;
	else
	end if;
end process;

process(selALU, saidaAC, saidaMEM, saidaALU) -- ALU
begin
	case selALU is
		when "000" => -- X + Y
			saidaALU	<=	STD_LOGIC_VECTOR(unsigned(saidaAC)	+
unsigned(saidaMEM));
		when "001" => -- X and Y
			saidaALU <= saidaAC and saidaMEM;
		when "010" => -- X or Y
			saidaALU <= saidaAC or saidaMEM;
		when "011" => -- not X
			saidaALU <= not saidaAC;
		when "100" => -- Y
			saidaALU <= saidaMEM;
		when "101" => -- X - Y
			saidaALU	<=	STD_LOGIC_VECTOR(unsigned(saidaAC)	-
unsigned(saidaMEM));
		when "110" => -- X xor Y
			saidaALU <= saidaAC xor saidaMEM;
		when others => -- 111
			saidaALU <= saidaAC;
	end case;
end process;

process(saidaALU) -- saidaNZ
begin
	if saidaALU = "00000000" then
		nzALU(0) <= '1';
	else
		nzALU(0) <= '0';
```

```
        end if;

        if saidaALU(7) = '1' then
                nzALU(1) <= '1';
        else
                nzALU(1) <= '0';
        end if;
end process;

process(clk) -- NZ
begin
        if clk'event and clk = '1' then
                if cargaNZ = '1' then
                        saidaNZ <= nzALU;
                else
                        saidaNZ <= saidaNZ;
                end if;
        else
        end if;
end process;

process(clk) -- RI
begin
        if clk'event and clk = '1' then
                if cargaRI = '1' then
                        saidaRI(3) <= saidaMEM(7);
                        saidaRI(2) <= saidaMEM(6);
                        saidaRI(1) <= saidaMEM(5);
                        saidaRI(0) <= saidaMEM(4); -- rest = don't care
                else
                        saidaRI <= saidaRI;
                end if;
        else
        end if;
end process;

end Behavioral;
```

3) Testbench VHDL completo

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer: Pedro Company Beck
--
-- Create Date:   10:09:39 07/25/2023
-- Design Name:
-- Project Name:  placa1
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: neander
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
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 testbench1 IS
END testbench1;

ARCHITECTURE behavior OF testbench1 IS

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

   COMPONENT neander
   PORT(
       rst : IN  std_logic;
       clk : IN  std_logic;
       outHalt : OUT  std_logic;
                           outPC : OUT STD_LOGIC_VECTOR(7 downto 0);
```

```
                        outAC : OUT STD_LOGIC_VECTOR(7 downto 0);
                        outRI : OUT STD_LOGIC_VECTOR(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal rst : std_logic := '0';
   signal clk : std_logic := '0';

         --Outputs
   signal outHalt : std_logic;
         signal outPC: STD_LOGIC_VECTOR(7 downto 0);
         signal outAC: STD_LOGIC_VECTOR(7 downto 0);
         signal outRI: STD_LOGIC_VECTOR(3 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

         -- Instantiate the Unit Under Test (UUT)
   duv: neander PORT MAP (
        rst => rst,
        clk => clk,
        outHalt => outHalt,
                        outPC => outPC,
                        outAC => outAC,
                        outRI => outRI
        );

   -- Clock process definitions
         process
   begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
   end process;


--   -- Stimulus process
         process
         begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      -- insert stimulus here
                rst <= '1';
                wait for clk_period;
```

```
                rst <= '0';
                wait for 1000000ns;
  end process;

END;
```

4) Explicação e descrição das aplicações em Assembly

PROGRAMA 1:
Esse programa se automodifica para incrementar os próprios endereços de escrita e leitura e executa 9 vezes as três primeiras instruções para somar os 9 valores de uma matriz 3x3 e guardar no espaço de memória reservado para o output.

Assembly do Neander:

```
0: NOP
1-2: LDA 128
3-4: ADD 137
5-6: STA 146
-- self-modifying code
7-8: LDA 2
9-10: ADD 127
11-12: STA 2

13-14: LDA 4
15-16: ADD 127
17-18: STA 4

19-20: LDA 6
21-22: ADD 127
23-24: STA 6
-- inc counter
25-26: LDA 126
27-28: ADD 127

-- jz to hlt
29-30: JZ 35
-- else
31-32: STA 126
33-34: JMP 1

35: HLT

-- data
126: 247
127: 1

128 - 136: matrix A
137 - 145: matrix B
146 - 154: output
```

PROGRAMA 2:
Esse programa utiliza o primeiro operando como contador e soma o segundo operando sucessivamente na output.
O programa utiliza a instrução SUB implementada para subtrair 1 do valor do contador (endereço do primeiro operando) até o mesmo chegar a 0.
Na execução, foram utilizados os valores A = 3 e B = 5, então a soma sucessiva é realizada 3 vezes.

Assembly do Neander:

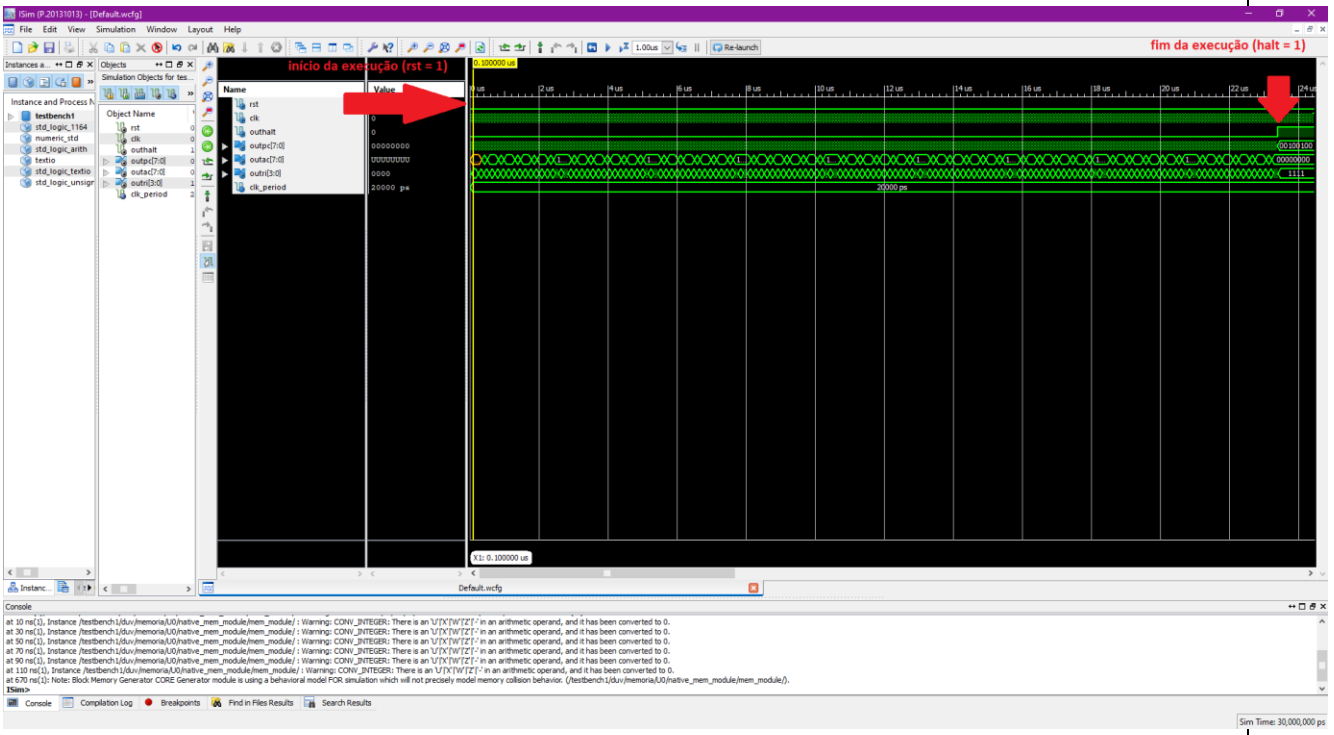
```
0: NOP
1-2: LDA 128
3-4: JZ 17
5-6: SUB 127 -- sub = 11000000 = C0 = 192
7-8: STA 128
9-10: LDA 130
11-12: ADD 129
13-14: STA 130
15-16: JMP 1

17: HLT

-- data
127: 1
128: A
129: B
130: out
```

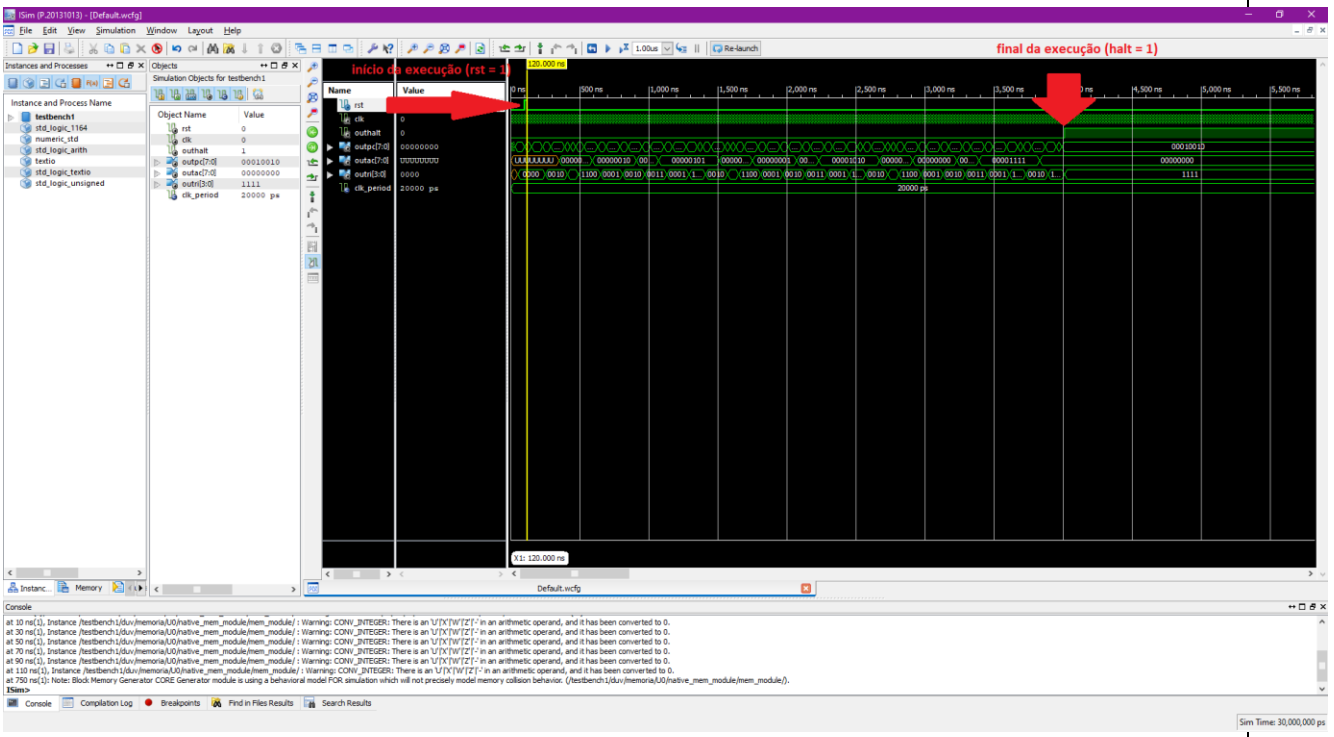
PROGRAMA 3:
Esse programa utiliza a instrução XOR implementada para realizar o swap de dois valores em memória sem utilizar nenhum endereço de memória temporária.

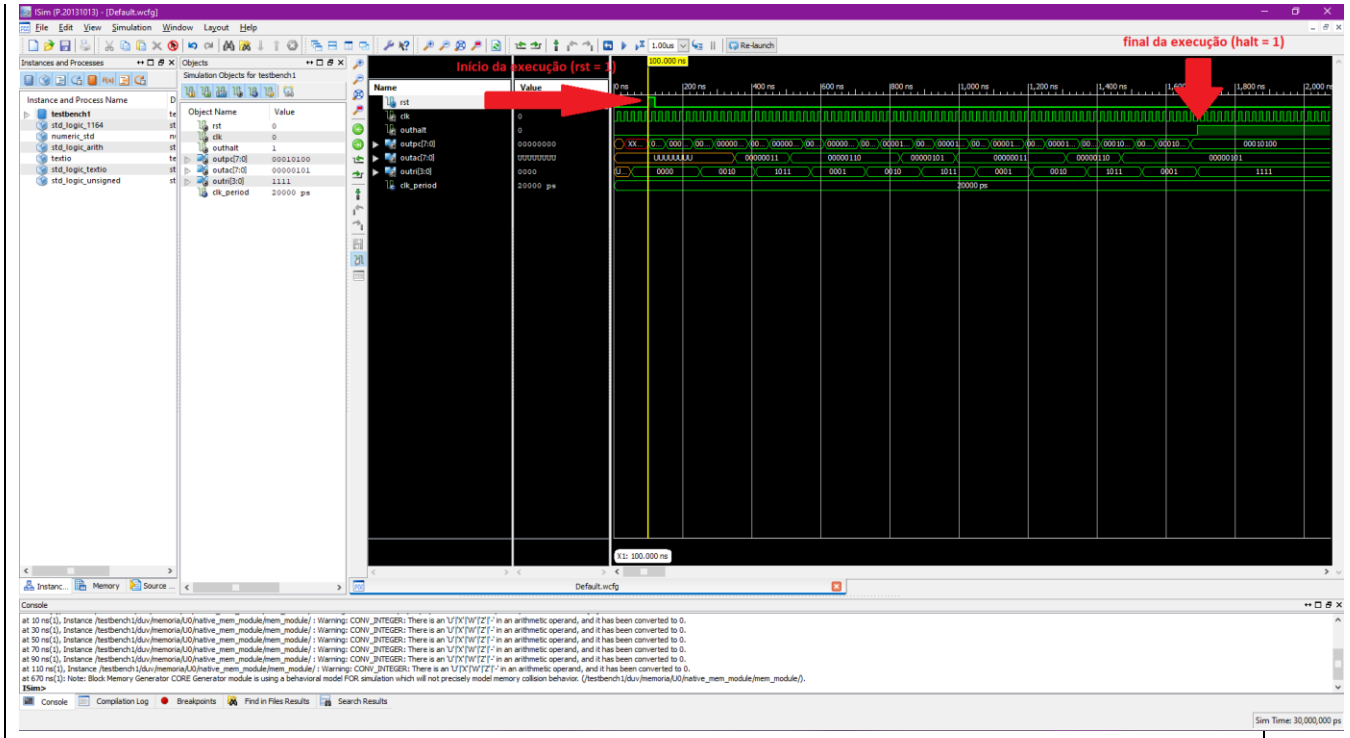
Assembly do Neander:

```
-- swap two values in place
0: NOP
1-2: LDA 128
3-4: XOR 129 -- XOR = 1011 0000 = B0 = 176
5-6: STA 128

7-8: LDA 129
9-10: XOR 128
11-12: STA 129

13-14: LDA 128
15-16: XOR 129
17-18: STA 128

19: HLT

-- data
128: A
129: B
```

5) Simulações sem e com atraso com detalhes e flechas mostrando inicio meio e final do programa e resultados






final da execução (halt = 1)
Início da execução (rst = 1)

6) 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) (T = 20ns)
Soma de matrizes	153	1165	23300ns = 0.0000233s
Multiplicação por somas sucessivas	28	195	3900ns = 0.0000039s
Programa com XOR	11	79	1580ns = 0.00000158s

Dados de Area do Neander
FPGA device: xc3s100e-5vq100
Numero de 4-LUTs ou 6-LUTs (conforme o FPGA): used = 135 / available = 1920
Numero de ffps: used = 39 / available = 1920
Numero de BRAM: used = 1 / available = 4
Numero de DSP:

neander Project Status (07/27/2023 - 21:40:20)			
Project File:	placa1.xise	**Parser Errors:**	No Errors
Module Name:	neander	**Implementation State:**	Synthesized
Target Device:	xc3s100e-5vq100	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	2 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	70	960	7%
Number of Slice Flip Flops	39	1920	2%
Number of 4 input LUTs	135	1920	7%
Number of bonded IOBs	23	66	34%
Number of BRAMs	1	4	25%
Number of GCLKs	1	24	4%

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