UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMATICA

Trabalho 1 - Sistemas Digitais

Prof. Fernanda Kastensmidt

TRABALHO Processador Neander

Pontuação: 10 pontos (vale 1/4 da nota do semestre)

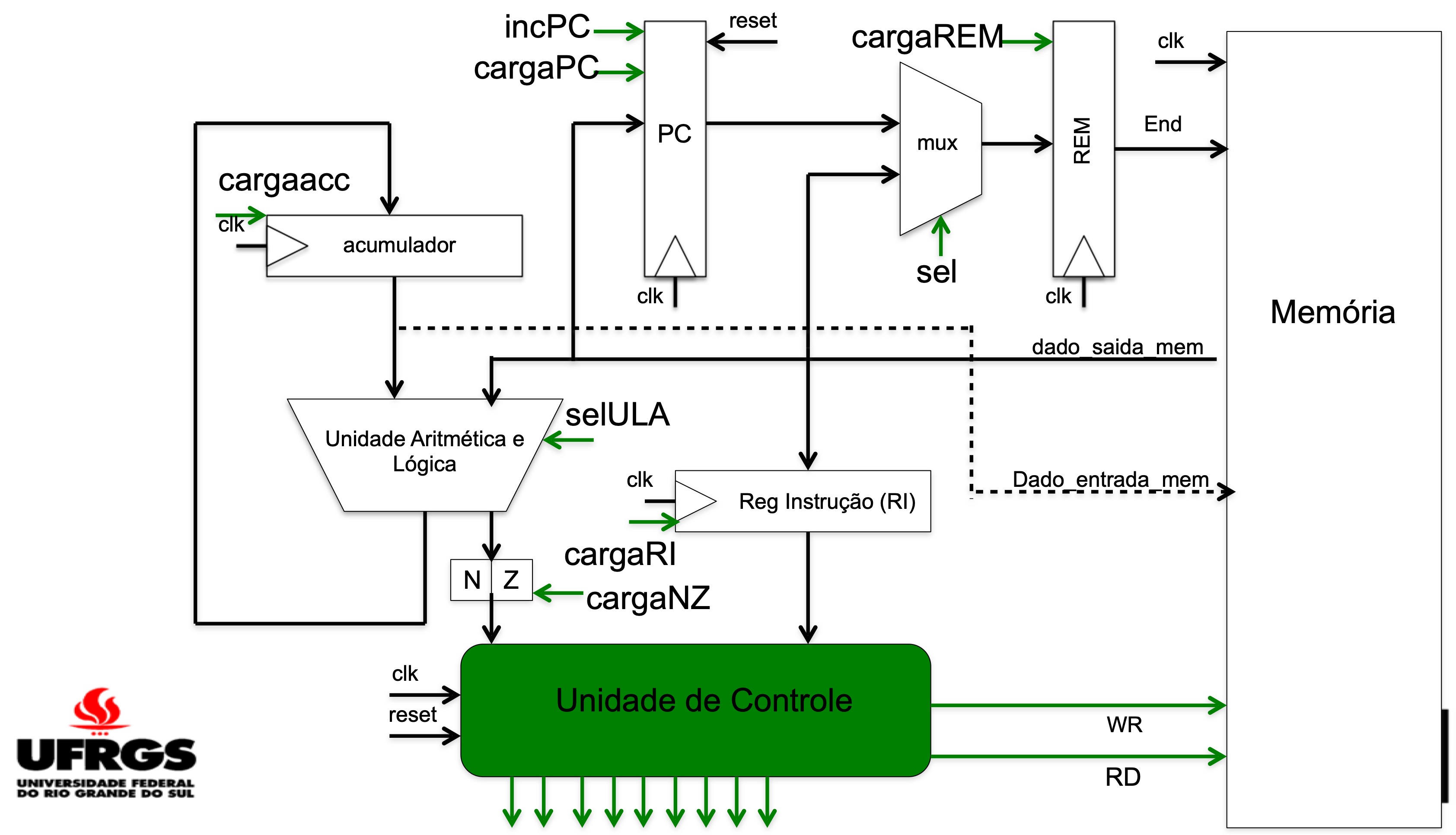
**Nome:** Juliana Rodrigues de Vargas  **matricula:** 00337553

**Objetivo:** projetar e descrever em VHDL o processador Neander, implementar 2 programas em sua memória e mostrar através de simulação lógica sem e com atraso o funcionamento.

Pontução extra: programar na placa de prototipação o Neander e mostrar funcionando com os displays 7 segmentos.

**PASSO 1:** 3 pontos

Descrever o DATAPATH do processador Neander em VHDL em uma entidade apenas chamada de datapath\_neander.



***Cole aqui o código completo em 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;

use IEEE.std\_logic\_unsigned.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity datapath is

Port (clk, rst: in std\_logic;

loadREM: in std\_logic;

incPC: in std\_logic;

loadRI: in std\_logic;

sel: in std\_logic;

loadAC: in std\_logic;

loadNZ: in std\_logic;

loadPC: in std\_logic;

selULA: in std\_logic\_vector (2 downto 0);

menOut: in std\_logic\_vector (7 downto 0);

menAddress:out std\_logic\_vector(7 downto 0);

menInput: out std\_logic\_vector(7 downto 0);

instruct: out std\_logic\_vector(3 downto 0);

negFlag: out std\_logic;

zeroFlag: out std\_logic);

end datapath;

architecture Behavioral of datapath is

-- declarations ==========================================================================================================================================

-- signals

-- reg

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

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

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

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

-- wires

signal ULAout: std\_logic\_vector (8 downto 0);

-- components

begin

-- instantiations ========================================================================================================================================

-- (port maps)

-- Statements ============================================================================================================================================

-- (combinational and sequential)

ULAout <= (('0' & ac) + ('0' & menOut)) when selULA = "000" else

(('0' & ac) and ('0' & menOut)) when selULA = "001" else

(('0' & ac) or ('0' & menOut)) when selULA = "010" else

('0' & (not(ac))) when selULA = "011" else

('0' & menOut) when selULA = "100" else

ULAout when selUla = "111";

-- + 000

-- and 001

-- or 010

-- not x 011

-- y 100

menAddress <= REMem;

menInput <= ac;

instruct <= RI;

process(clk, rst)

begin

if rst = '1' then

ac <= "00000000";

PC <= "00000000";

elsif clk'event and clk='1' then

-- PC

if loadPC = '1' then

PC <= menOut;

elsif incPC = '1' then

PC <= PC + 1;

end if;

-- REMem

if loadREM = '1' then

case sel is

when '0' =>

REMem <= PC;

when '1' =>

REMem <= menOut;

when others =>

REMem <= REMem;

end case;

end if;

-- RI

if loadRI = '1' then

RI <= menOut(7 downto 4);

end if;

-- Acumulador

if loadAC = '1' then

ac <= ULAout(7 downto 0);

end if;

--NZ

if loadNZ = '1' then

negFlag <= ULAout(7) ;

if ULAout(7 downto 0) = "00000000" then

zeroFlag <= '1';

else

zeroFlag <= '0';

end if;

end if;

end if;

end process;

end Behavioral;

**Qual componente FPGA escolheste para a síntese? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Quantos registradores tem o datapath do Neander? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

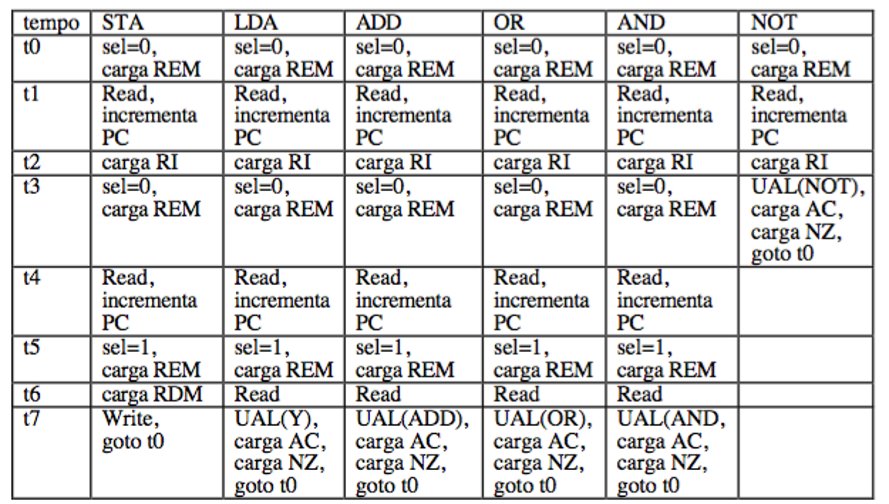
**Quantas operações diferentes tem a ULA? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

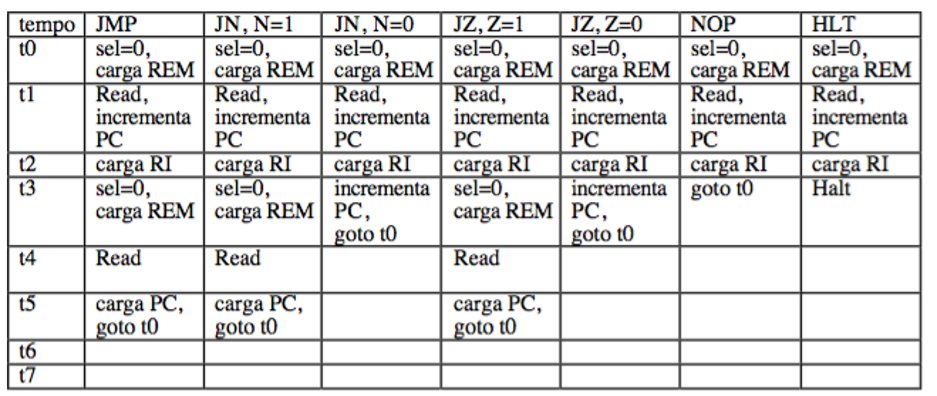
**A área do DATAPTH em # LUTs: \_\_\_\_\_\_\_\_\_\_\_ e #ffps: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**PASSO 2:** 3 pontos

Descrever a parte de controle do Neander em VHDL como uma maquina de estados usando 2 PROCESS, um process(clk, rst) e outro process(estado, entradas).

Dada as tabelas com as instruções do Neander por estado da máquina de estrados





Cole aqui o VHDL da parte de controle usando FSM com dois process.

*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 leaf cells in this code.*

*--library UNISIM;*

*--use UNISIM.VComponents.all;*

*entity control is*

*Port ( clk, rst: in std\_logic;*

*instruction: in std\_logic\_vector(3 downto 0);*

*negFlag, zeroFlag: in std\_logic;*

*loadREM: out std\_logic;*

*incPC: out std\_logic;*

*loadRI: out std\_logic;*

*sel: out std\_logic;*

*loadAC: out std\_logic;*

*loadNZ: out std\_logic;*

*loadPC: out std\_logic;*

*write: out std\_logic;*

*read: out std\_logic;*

*selULA: out std\_logic\_vector (2 downto 0));*

*end control;*

*architecture Behavioral of control is*

*TYPE state IS (t0, t1, t2, t3, t4, t5, t6, t7);*

*signal states: state;*

*signal next\_state: state;*

*constant ins\_STA : std\_logic\_vector (3 downto 0) := "0001"; -- 1*

*constant ins\_LDA : std\_logic\_vector (3 downto 0) := "0010"; -- 2*

*constant ins\_ADD : std\_logic\_vector (3 downto 0) := "0011"; -- 3*

*constant ins\_OR : std\_logic\_vector (3 downto 0) := "0100"; -- 4*

*constant ins\_AND : std\_logic\_vector (3 downto 0) := "0101"; -- 5*

*constant ins\_NOT : std\_logic\_vector (3 downto 0) := "0110"; -- 6*

*constant ins\_JMP : std\_logic\_vector (3 downto 0) := "1000"; -- 8*

*constant ins\_JN : std\_logic\_vector (3 downto 0) := "1001"; -- 9*

*constant ins\_JZ : std\_logic\_vector (3 downto 0) := "1010"; -- a*

*constant ins\_NOP : std\_logic\_vector (3 downto 0) := "0000"; -- 0*

*constant ins\_HLT : std\_logic\_vector (3 downto 0) := "1111"; -- f*

*-- + 000*

*-- and 001*

*-- or 010*

*-- not x 011*

*-- y 100*

*begin*

*process(clk, rst)*

*begin*

*if (rst = '1') then*

*states <= t0;*

*elsif clk'event and clk='1' then*

*states <= next\_state;*

*end if;*

*end process;*

*process(states, instruction)*

*begin*

*case states is*

*when t0 =>*

*loadREM <= '1';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t1;*

*when t1 =>*

*loadREM <= '0';*

*incPC <= '1';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '1';*

*write <= '0';*

*next\_state <= t2;*

*when t2 =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '1';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t3;*

*when t3 =>*

*case instruction is*

*when ins\_NOT =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '1';*

*loadNZ <= '1';*

*loadPC <= '0';*

*selULA <= "011";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*when ins\_STA | ins\_LDA | ins\_ADD | ins\_OR | ins\_AND |ins\_JMP =>*

*loadREM <= '1';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t4;*

*when ins\_JN =>*

*if (negFlag = '1') then*

*loadREM <= '1';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t4;*

*elsif (negFlag = '0') then*

*loadREM <= '0';*

*incPC <= '1';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*end if;*

*when ins\_JZ =>*

*if (zeroFlag = '1') then*

*loadREM <= '1';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t4;*

*elsif (zeroFlag = '0') then*

*loadREM <= '0';*

*incPC <= '1';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*end if;*

*when ins\_NOP =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*when ins\_HLT =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t3;*

*when others =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*end case;*

*when t4 =>*

*case instruction is*

*when ins\_STA | ins\_LDA | ins\_ADD | ins\_OR | ins\_AND =>*

*loadREM <= '0';*

*incPC <= '1';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '1';*

*write <= '0';*

*when ins\_JMP =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '1';*

*write <= '0';*

*when ins\_JN =>*

*if negFlag = '1' then*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '1';*

*write <= '0';*

*end if;*

*when ins\_JZ =>*

*if zeroFlag = '1' then*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '1';*

*write <= '0';*

*end if;*

*when others =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*end case;*

*next\_state <= t5;*

*when t5 =>*

*case instruction is*

*when ins\_STA | ins\_LDA | ins\_ADD | ins\_OR | ins\_AND =>*

*loadREM <= '1';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '1';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t6;*

*when ins\_JMP =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '1';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*when ins\_JN =>*

*if negFlag = '1' then*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '1';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*end if;*

*when ins\_JZ =>*

*if zeroFlag = '1' then*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '1';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*end if;*

*when others =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*next\_state <= t0;*

*end case;*

*when t6 =>*

*case instruction is*

*when ins\_LDA | ins\_ADD | ins\_OR | ins\_AND =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '1';*

*write <= '0';*

*when others =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*end case;*

*next\_state <= t7;*

*when t7 =>*

*case instruction is*

*when ins\_STA =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '1';*

*when ins\_LDA =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '1';*

*loadNZ <= '1';*

*loadPC <= '0';*

*selULA <= "100";*

*read <= '0';*

*write <= '0';*

*when ins\_ADD =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '1';*

*loadNZ <= '1';*

*loadPC <= '0';*

*selULA <= "000";*

*read <= '0';*

*write <= '0';*

*when ins\_OR =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '1';*

*loadNZ <= '1';*

*loadPC <= '0';*

*selULA <= "010";*

*read <= '0';*

*write <= '0';*

*when ins\_AND =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '1';*

*loadNZ <= '1';*

*loadPC <= '0';*

*selULA <= "001";*

*read <= '0';*

*write <= '0';*

*when others =>*

*loadREM <= '0';*

*incPC <= '0';*

*loadRI <= '0';*

*sel <= '0';*

*loadAC <= '0';*

*loadNZ <= '0';*

*loadPC <= '0';*

*selULA <= "111";*

*read <= '0';*

*write <= '0';*

*end case;*

*next\_state <= t0;*

*end case;*

*end process;*

*end Behavioral;*

**PASSO 3:** 1 ponto

Descrever o programa em Assembly do Neander que realize a multiplicação de dois números inteiros positivos de 8 bits por soma sucessiva e colocar no arquivo .COE na memória BRAM.

Inserir aqui o programa em Assembly com explicação.

LDA 252

NOT

STA 252

ADD 251

STA 252

LDA 255

STA 254

LDA 253

JZ 29

ADD 252

STA 253

LDA 254

ADD 255

STA 254

JMP 13

LDA 251

STA 252

LDA 254

JZ 46

LDA 255

NOT

ADD 251

ADD 254

STA 254

HLT

////////////////////////////////////////////////

//explicacao

////////////////////////////////////////////////

**endereco** **variavel**

255 variavel B, que sera somada consigo mesma ate se obter a multiplicação

254 onde fica guardada cada nova soma parcial, ao final sera o resultado

253 variavel A, que será decrementada a cada soma sucessiva de B, até chegar em zero e o programa parar

252 valor -1, para decrementar A

251 valor 1, para reiniciar a variavel ao final do programa

Inserir aqui o .coe

memory\_initialization\_radix=16; modes: 10 (decimal) / 16 (hexadecimal)

memory\_initialization\_vector=20, FC, 60, 10, FC, 30, FB, 10, FC, 20, FF, 10, FE, 20, FD, A0, 1D, 30, FC, 10, FD, 20, FE, 30, FF, 10, FE, 80, 0D, 20, FB, 10, FC, 20, FE, A0, 2E, 20, FF, 60, 30, FB, 30, FE, 10, FE, F0, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 05, 04, 00, 09, 01, 02, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 01, 01, 08, 00, 02;

**PASSO 4:** 3 pontos

Simular sem atraso o Neander com o programa teste a ser feito pelo aluno e depois que testado e funcionando, simular com o programa do passo 3. Depois de tudo funcionando, simular também com atraso.

Lembrem-se que deve ser feito um testbench para a simulação.

Colar aqui o programa teste e simulações (.JPG)

LDA 100

ADD 101

STA 102

LDA 103

NOT

ADD 104

ADD 102

JZ 19

LDA 104

JMP 21

LDA 105

HLT

////////////////////////////////////////////////

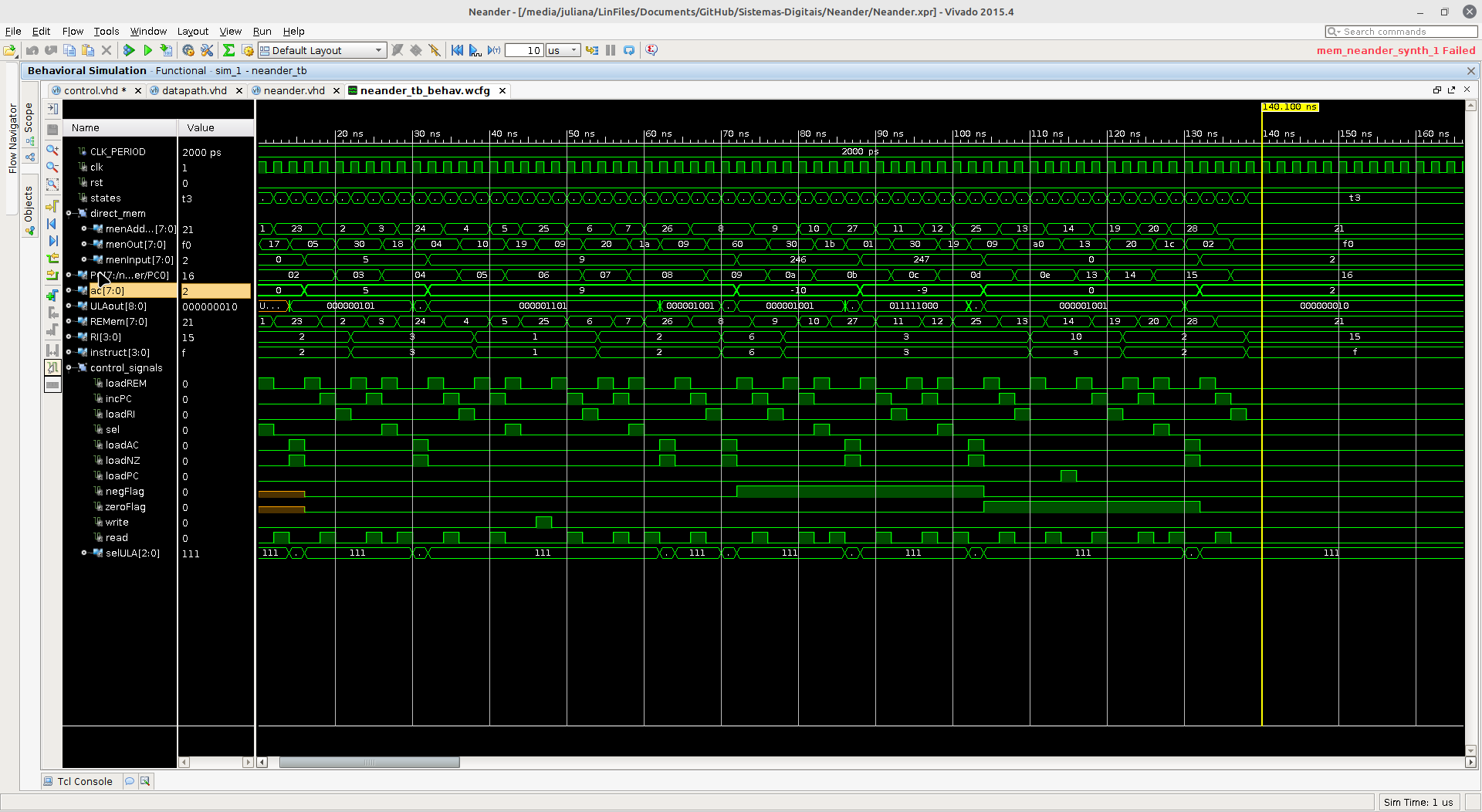
//explicacao

////////////////////////////////////////////////

testa se A(end 100) + B (end 101) é igual a c (102) fazendo a soma de c com -(a+b)

se for igual, ac recebe 2, se não, ac recebe 1

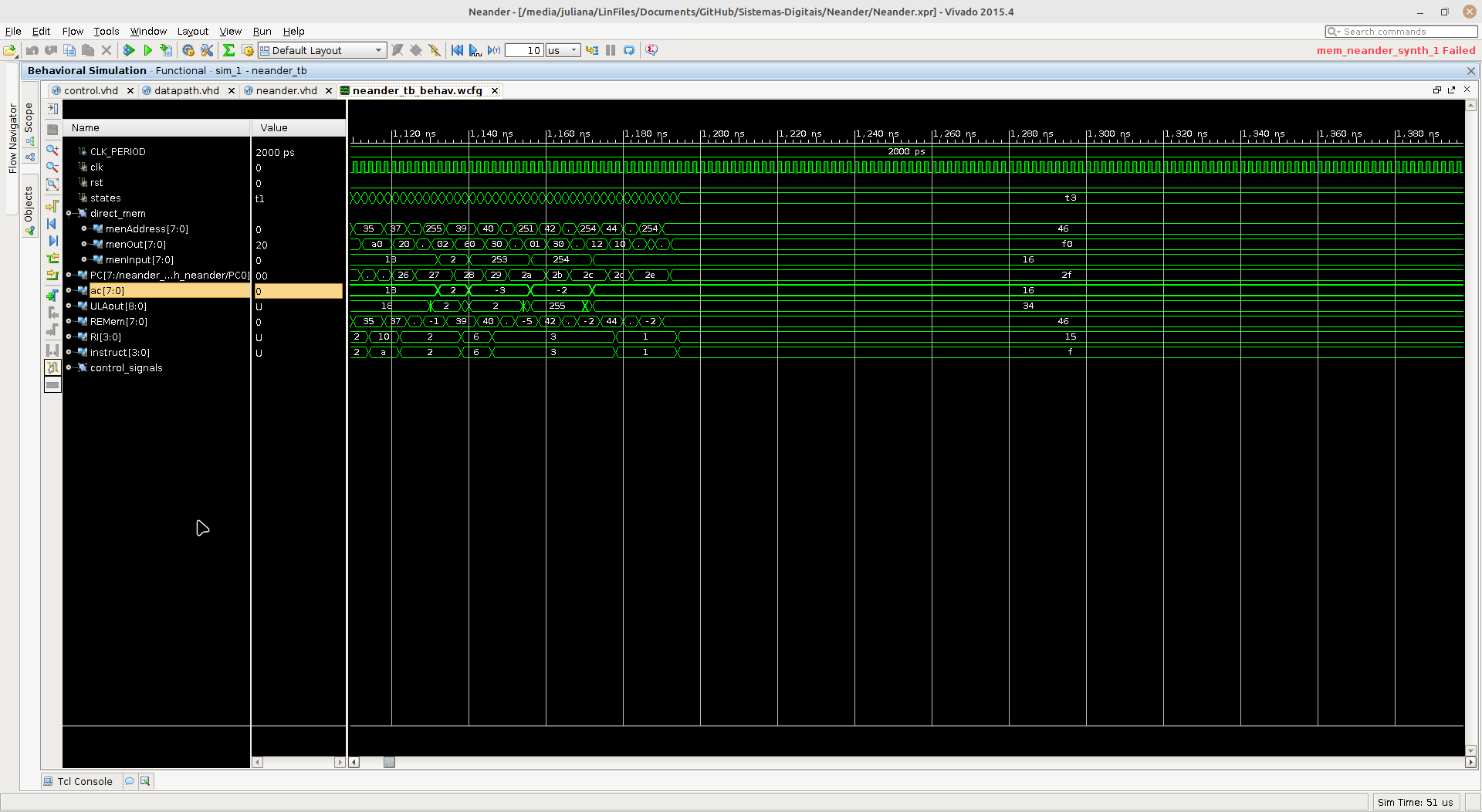
**Simulação do programa TESTE sem atraso:**



**Simulação do programa TESTE com atraso:**

Colar aqui as simulações do programa do passo 3:

**Simulação do programa PASSO 3 sem atraso:**



**Simulação do programa PASSO 3 com atraso:**

Dados do Neander completo:

**A área do DATAPTH em # LUTs: \_\_\_\_\_\_\_\_\_\_\_ e #ffps: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ e # BRAM \_\_\_\_\_**

**Quantos ciclos de relógio foram necessários para a execução do programa de multiplicação no Neander? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = # instrucoes x # cc por instrucao**

**Qual frequência de operação o Neander atingiu? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Como tu fizeste o teste para saber que ele não consegue rodar mais rápido que essa frequência? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.**

PONTO EXTRA:

Link para video mostrando o funcionamento da placa de prototipação.