UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMATICA

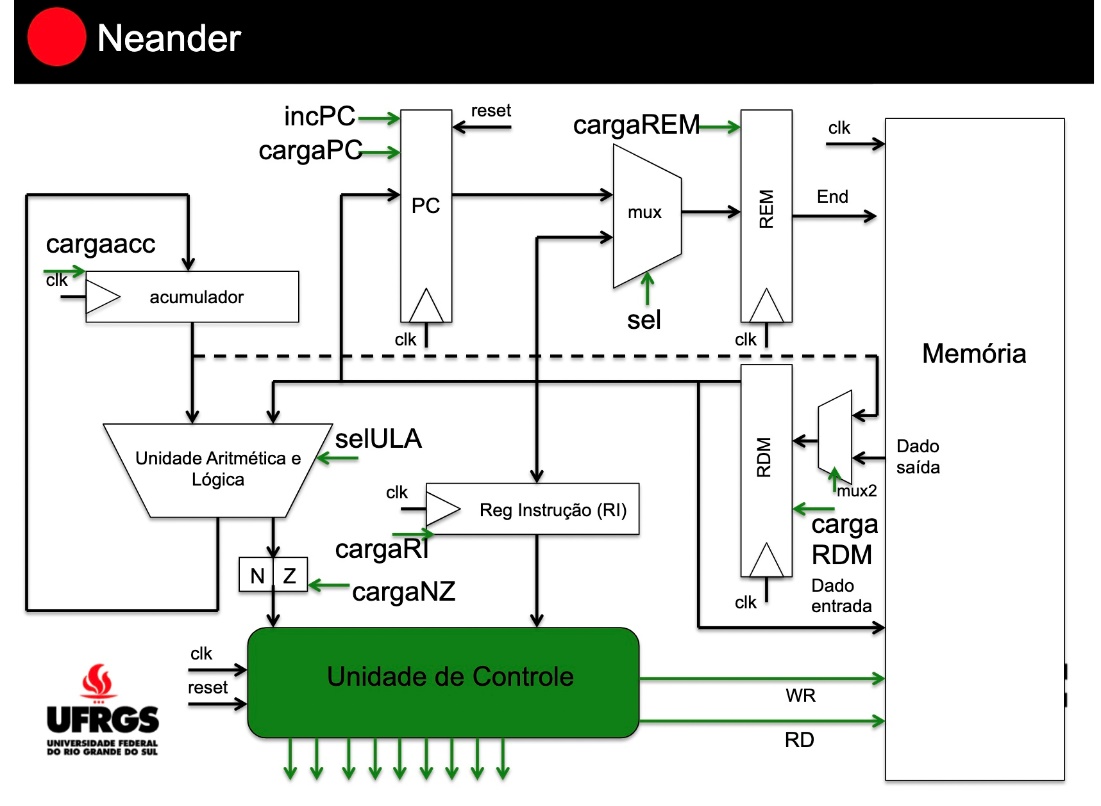
Trabalho 1 - Sistemas Digitais – Parte 1

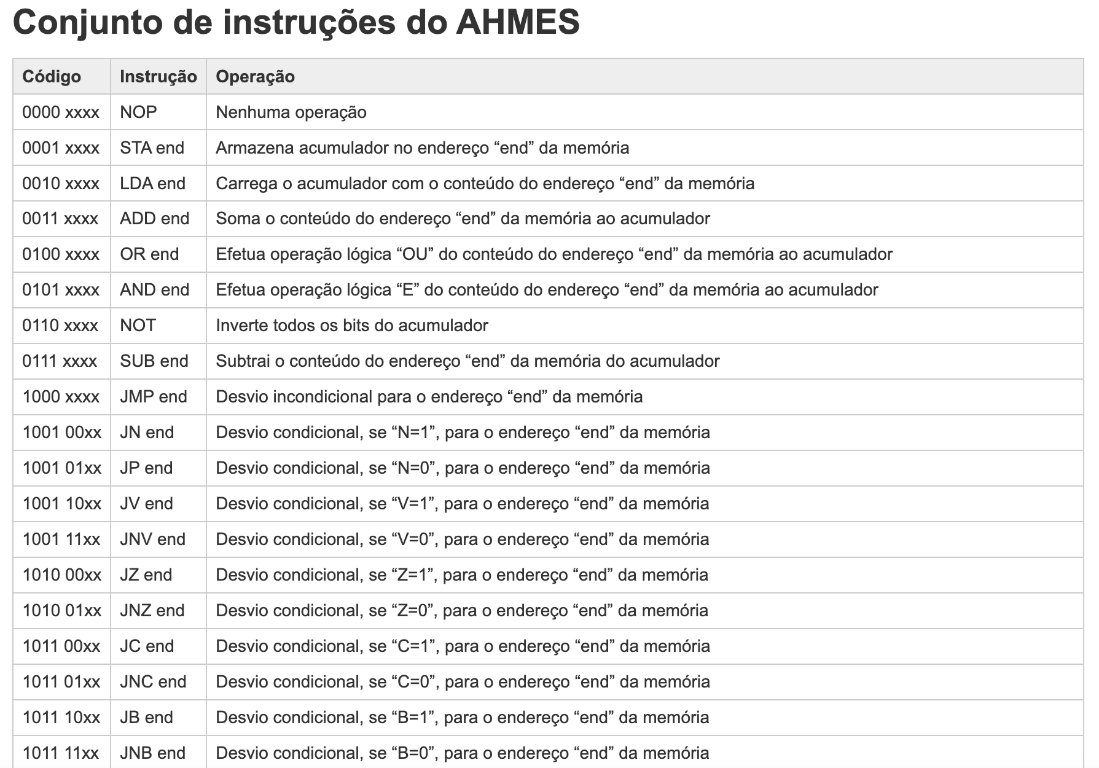
Prof. Fernanda Kastensmidt

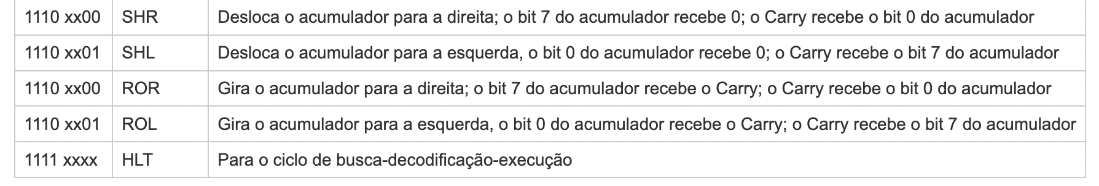
**Objetivo do Trabalho Completo a ser enviado em Fevereiro 10/2:** projetar e descrever em VHDL o processador AHMES (https://www.inf.ufrgs.br/arq/wiki/doku.php?id=insahmes) implementar 2 programas em sua memória e mostrar através de simulação lógica sem e com atraso o funcionamento.

**Parte 1 – a ser realizado na aula 24/1/23**

O DATAPATH do AHMES é o mesmo do NEANDER mas a ULA e o registrador ACC muda.







**AULA 1 (24/1/2023)**

Descrever o DATAPATH do processador RAMSES em VHDL em uma entidade apenas chamada de datapath\_ramses.

1. ***Cole aqui o código completo em VHDL do datapath***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity datapath\_ahmes is

Port

(

CLOCK : in STD\_LOGIC;

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

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

N : out STD\_LOGIC;

Z : out STD\_LOGIC;

V : out STD\_LOGIC;

C : out STD\_LOGIC;

B : out STD\_LOGIC

);

end datapath\_ahmes;

architecture Behavioral of datapath\_ahmes is

-- PC

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

signal load\_PC: std\_logic := '0';

signal inc\_PC: std\_logic := '0';

-- AC

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

signal load\_AC: std\_logic := '0';

-- ULA

signal ULA\_X: std\_logic\_vector (7 downto 0); -- é o reg\_AC

signal ULA\_Y: std\_logic\_vector (7 downto 0); -- é o reg\_RDM;

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

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

-- FLAGS

signal reg\_N: std\_logic := '0';

signal reg\_Z: std\_logic := '0';

signal reg\_V: std\_logic := '0';

signal reg\_C: std\_logic := '0';

signal reg\_B: std\_logic := '0';

signal load\_flag: std\_logic := '0';

-- MUX\_REM

signal MUX\_REM\_out: std\_logic\_vector (7 downto 0);

signal sel\_REM\_MUX: std\_logic :='0';

-- REM

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

signal load\_REM: std\_logic := '0';

-- MUX\_RDM

signal MUX\_RDM\_out: std\_logic\_vector (7 downto 0);

signal sel\_RDM\_MUX: std\_logic :='0';

-- RDM

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

signal load\_RDM: std\_logic := '0';

-- RI

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

signal load\_RI: std\_logic := '0';

-- DECOD

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

signal DECOD\_out: std\_logic\_vector(23 downto 0);

signal DECOD\_sel: std\_logic\_vector(4 downto 0);

-- MEM

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

signal read\_mem: std\_logic := '0';

signal write\_mem: std\_logic := '0';

-- variables

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 => write\_mem,

-- addra => reg\_mem,

-- dina => reg\_AC,

-- douta => reg\_mem

-- );

process (CLOCK, RESET) -- PC

begin

if (RESET ='1') then

reg\_PC <= "00000000";

elsif (rising\_edge(CLOCK)) then -- na subida do clock

if (load\_PC='1') then

reg\_PC<= reg\_RDM;

elsif(inc\_PC='1') then

reg\_PC <= std\_logic\_vector(unsigned(reg\_PC) + 1);

else

reg\_PC <= reg\_PC;

end if;

end if;

end process;

process (CLOCK, RESET) -- AC

begin

if (RESET='1') then

reg\_AC <= "00000000";

elsif (rising\_edge(CLOCK)) then -- na subida do clock

if (load\_AC='1') then

reg\_AC <= ULA\_out;

else

reg\_AC <= reg\_AC;

end if;

end if;

end process;

-- ULA

ULA\_X <= reg\_AC; -- recupera valor X do acumulador

ULA\_Y <= reg\_MEM; -- recupera valor Y de uma posicao da memória

process(sel\_ULA, ULA\_X, ULA\_Y, reg\_C)

variable C\_temp: std\_logic := '0';

begin

case sel\_ULA is

when "0000" =>

if((ULA\_X + ULA\_Y) > 127) then

reg\_V <= '1';

else

reg\_V <= '0';

end if;

reg\_B <= '0';

reg\_C <= '0';

ULA\_out <= (ULA\_X + ULA\_Y); -- operacao ADD

when "0001" =>

reg\_V <= '0';

reg\_B <= '0';

reg\_C <= '0';

ULA\_out <= (ULA\_X OR ULA\_Y); -- operacao OR

when "0010" =>

reg\_V <= '0';

reg\_B <= '0';

reg\_C <= '0';

ULA\_out <= (ULA\_X AND ULA\_Y); -- operacao AND

when "0011" =>

reg\_V <= '0';

reg\_B <= '0';

reg\_C <= '0';

ULA\_out <= (NOT ULA\_X); -- operacao NOT

when "0100" =>

if((ULA\_X - ULA\_Y) < 0) then

reg\_C <= '1';

reg\_B <= '1';

else

reg\_V <= '0';

reg\_B <= '0';

reg\_C <= '0';

end if;

reg\_V <= '0';

ULA\_out <= (ULA\_X - ULA\_Y); -- operacao SUB

when "0101" => -- operacao SHR

reg\_V <= '0';

reg\_B <= '0';

reg\_C <= ULA\_X(0);

ULA\_out(7) <= '0';

ULA\_OUT(6) <= ULA\_X(7);

ULA\_OUT(5) <= ULA\_X(6);

ULA\_OUT(4) <= ULA\_X(5);

ULA\_OUT(3) <= ULA\_X(4);

ULA\_OUT(2) <= ULA\_X(3);

ULA\_OUT(1) <= ULA\_X(2);

ULA\_OUT(0) <= ULA\_X(1);

when "0110" => -- operacao SHL

reg\_V <= '0';

reg\_B <= '0';

reg\_C <= ULA\_X(7);

ULA\_out <= (ULA\_X + ULA\_X);

when "0111" => -- operacao ROR

reg\_V <= '0';

reg\_B <= '0';

C\_temp := ULA\_X(0);

ULA\_out(7) <= reg\_C;

ULA\_OUT(6) <= ULA\_X(7);

ULA\_OUT(5) <= ULA\_X(6);

ULA\_OUT(4) <= ULA\_X(5);

ULA\_OUT(3) <= ULA\_X(4);

ULA\_OUT(2) <= ULA\_X(3);

ULA\_OUT(1) <= ULA\_X(2);

ULA\_OUT(0) <= ULA\_X(1);

reg\_C <= C\_temp;

when "1000" => -- operacao ROL

reg\_V <= '0';

reg\_B <= '0';

C\_temp := ULA\_X(7);

ULA\_out <= (ULA\_X + ULA\_X);

ULA\_out(0) <= reg\_C;

reg\_C <= C\_temp;

when others => ULA\_out <= "XXXXXXXX";

end case;

end process;

process (CLOCK, RESET) -- FLAGS

begin

if (RESET = '1') then

reg\_N <= '0';

reg\_Z <= '0';

reg\_V <= '0';

reg\_B <= '0';

--reg\_C <= '0';

elsif(rising\_edge(CLOCK)) then -- na subida do clock

if reg\_AC = "00000000" then

reg\_Z <= '1';

else

reg\_Z <= '0';

end if;

reg\_N <= reg\_AC(7); -- msb do registrador AC

end if;

end process;

process (sel\_REM\_MUX, reg\_PC, MUX\_RDM\_out) -- MUX\_REM

begin

if (sel\_REM\_MUX = '0') then

MUX\_REM\_out <= reg\_PC;

else

MUX\_REM\_out <= MUX\_RDM\_out;

end if;

end process;

process(CLOCK, RESET) -- REM

begin

if (RESET='1') then

reg\_REM <= "00000000";

elsif (rising\_edge(CLOCK)) then -- na subida do clock

if (load\_REM ='1') then

reg\_REM <= MUX\_REM\_out;

else

reg\_REM <= reg\_REM;

end if;

end if;

end process;

process (sel\_RDM\_MUX, reg\_mem, reg\_AC) -- MUX\_RDM

begin

if(sel\_REM\_MUX = '0') then

MUX\_RDM\_out <= reg\_MEM;

else

MUX\_RDM\_out <= reg\_AC;

end if;

end process;

process (CLOCK, RESET) -- RDM

begin

if (RESET = '1') then

reg\_RDM <= "00000000";

elsif (rising\_edge(CLOCK)) then

if (load\_RDM = '1') then

reg\_RDM <= reg\_AC;

else

reg\_RDM <= MUX\_RDM\_out;

end if;

end if;

end process;

process (CLOCK, RESET) -- RI

begin

if (RESET='1') then

reg\_RI <= "00000000";

elsif (rising\_edge(CLOCK)) then

if (load\_RI = '1') then

reg\_RI <= reg\_RDM;

else

reg\_RI <= reg\_RI;

end if;

end if;

end process;

-- DECOD

reg\_DECOD <= reg\_RI;

process(reg\_decod)

begin

DECOD\_out <= "000000000000000000000000";

case reg\_DECOD is

when "00000000" => DECOD\_out(0) <= '1'; -- 00 NOP

when "00010000" => DECOD\_out(1) <= '1'; -- 16 STA

when "00100000" => DECOD\_out(2) <= '1'; -- 32 LDA

when "00110000" => DECOD\_out(3) <= '1'; -- 48 ADD

when "01000000" => DECOD\_out(4) <= '1'; -- 64 OR

when "01010000" => DECOD\_out(5) <= '1'; -- 80 AND

when "01100000" => DECOD\_out(6) <= '1'; -- 96 NOT

when "01110000" => DECOD\_out(7) <= '1'; -- 112 SUB

when "10000000" => DECOD\_out(8) <= '1'; -- 128 JMP

when "10010000" => DECOD\_out(9) <= '1'; -- 144 JN

when "10010100" => DECOD\_out(10) <= '1'; -- 148 JP

when "10011000" => DECOD\_out(11) <= '1'; -- 152 JV

when "10011100" => DECOD\_out(12) <= '1'; -- 156 JNV

when "10100000" => DECOD\_out(13) <= '1'; -- 160 JZ

when "10100100" => DECOD\_out(14) <= '1'; -- 164 JNZ

when "10110000" => DECOD\_out(15) <= '1'; -- 176 JC

when "10110100" => DECOD\_out(16) <= '1'; -- 180 JNC

when "10111000" => DECOD\_out(17) <= '1'; -- 184 JB

when "10111100" => DECOD\_out(18) <= '1'; -- 188 JNB

when "11100000" => DECOD\_out(19) <= '1'; -- 224 SHR

when "11100001" => DECOD\_out(20) <= '1'; -- 225 SHL

when "11100010" => DECOD\_out(21) <= '1'; -- 226 ROR

when "11100011" => DECOD\_out(22) <= '1'; -- 227 ROL

when "11110000" => DECOD\_out(23) <= '1'; -- 240 HLT

when others => DECOD\_out <= "000000000000000000000000";

end case;

end process;

end Behavioral;

1. Qual componente FPGA escolheste para a síntese? Artix 7 – xc7a100t-3csg324
2. Quantos registradores tem o datapath do AHMES? 13
3. Quantas operações diferentes tem a ULA? 9
4. A área do DATAPTH em # LUTs: \_\_\_\_\_\_\_\_\_\_\_ e #ffps: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\*ENTREGAR DIA da PARTE 1 29/1/2022\*\*\*\* Editando esse DOC e submetendo no MS-TEAMS**