Trabalho: Projeto e Simulação de uma ULA em VHDL

Aluno: Adriano Ulrich do Prado Wiedmann
Matrícula: 202014824

Objetivo:

Projetar, simular e sintetizar uma versão da Unidade lógica e Aritmética (ULA) do RISC-V de 32 bits utilizando o *ModelSim*.

Versão ModelSim:

O trabalho foi realizado utilizando o *ModelSim* na versão 20.1.1.

Código:

Este trabalho consiste em dois arquivos vhd. O arquivo *ulaRiscv.vhd* e *ulaTestbench.vhd* (*testbench*).

1. ***Arquivo ulaRiscv.vhd***

   É definido uma entidade chamada "*ulaRiscv*". Nela é criada um parâmetro genérico chamado *WSIZE* de tamanho 32 bits.

   As portas da entidade definidas são:

   - *A* e *B* – Duas entradas de dados de 32 bits;
   - *opcode* – Entrada de 4 bits que representa as operações;
   - *Z* – Uma saída de dados de 32 bits;
   - *zero* – Uma porta de saída que indica se o resultado da operação é 0 ou não, representado por um único bit.

   As operações que este código realiza são as seguintes:

Operação	Significado	OpCode
ADD A, B	Z recebe a soma das entradas A, B	0000
SUB A, B	Z recebe A - B	0001
AND A, B	Z recebe a operação lógica A and B, bit a bit	0010
OR A, B	Z recebe a operação lógica A or B, bit a bit	0011
XOR A, B	Z recebe a operação lógica A xor B, bit a bit	0100
SLL A, B	Z recebe a entrada A deslocada B bits à esquerda	0101
SRL A, B	Z recebe a entrada A deslocada B bits à direita sem sinal	0110
SRA A, B	Z recebe a entrada A deslocada B bits à direita com sinal	0111
SLT A, B	Z = 1 se A < B, com sinal	1000
SLTU A, B	Z = 1 se A < B, sem sinal	1001
SGE A, B	Z = 1 se A ≥ B, com sinal	1010
SGEU A, B	Z = 1 se A ≥ B, sem sinal	1011
SEQ A, B	Z = 1 se A == B	1100
SNE A, B	Z = 1 se A != B	1101

Para a realização dessas operações é utilizado *Case-When statement* e através do *opcode* é realizado a operação. Conforme as Figuras 1 e 2.

```
case opcode is

        -- ADD A, B
        when "0000" => a32 <= std_logic_vector(signed(A) + signed(B));

        -- SUB A, B
        when "0001" => a32 <= std_logic_vector(signed(A) - signed(B));

        -- AND A, B
        when "0010" => a32 <= A AND B;

        -- OR A, B
        when "0011" => a32 <= A OR B;

        -- XOR A, B
        when "0100" => a32 <= A XOR B;

        -- SLL A, B
        when "0101" => a32 <= std_logic_vector(shift_left(unsigned(A), to_integer(unsigned(B)))); -- A SLL B

        -- SRA A, B sem sinal
        when "0110" => a32 <= std_logic_vector(shift_right(unsigned(A), to_integer(unsigned(B)))); -- A SRL B

        -- SRA A, B com sinal
        when "0111" => a32 <= std_logic_vector(shift_right(signed(A), to_integer(signed(B))));
```

Figura 1 – Através do *opcode* é identificado a operação a ser realizada.

```
        -- SLT A, B
        when "1000" => a32 <= (others => '0');  -- inicializa com zero
                if signed(A) < signed(B) then
                        a32(0) <= '1';
                end if;

        -- SLTU A, B
        when "1001" => a32 <= (others => '0');  -- inicializa com zero
                if unsigned(A) < unsigned(B) then
                        a32(0) <= '1';
                end if;

        -- SGE A, B
        when "1010" => a32 <= (others => '0');  -- inicializa com zero
                if signed(A) >= signed(B) then
                        a32(0) <= '1';
                end if;

        -- SGEU A, B
        when "1011" => a32 <= (others => '0');  -- inicializa com zero
                if unsigned(A) >= unsigned(B) then
                        a32(0) <= '1';
                end if;

        -- SEQ A, B
        when "1100" => a32 <= (others => '0');  -- inicializa com zero
                if A = B then
                        a32(0) <= '1';
                end if;

        -- SNE A, B
        when "1101" => a32 <= (others => '0');  -- inicializa com zero
                if A /= B then
                        a32(0) <= '1';
                end if;

        when others => a32 <= X"00000000";
end case;
```

Figura 2 – Através do *opcode* é identificado a operação a ser realizada.

2. ***Arquivo ulaTestbench.vhd***

No *testbench* é realizado a associação (*port mapping*) dos sinais entre a unidade de teste (*ulaTestbench*) e a unidade de *design (ulaRiscv)* da seguinte forma:

```
uut: ulaRiscv port map (A => A_tb, B => B_tb,
                        opcode => opcode_tb,
                        Z => Z_tb,
                        zero => zero_tb
                        );
```

E para verificar o funcionamento, é atribuído valores às variáveis A_tb e B_tb. E para *opcode_tb* é atribuído o *opcode* da instrução. Por exemplo:

```
-- Teste 1 - ADD com resultado positivo
A_tb <= X"0000000C";
B_tb <= X"00000005";
opcode_tb <= "0000";
wait for 1 ns;
assert Z_tb = X"00000011"
        report "Erro - teste 1 - ADD"
        severity error;
```

A diferença entre as comparações com e sem sinal:

As operações com sinal consideram o bit mais significativo como o bit de sinal. Se o bit for 1, o número é considerado negativo. Se o bit for 0, o número é considerado positivo.

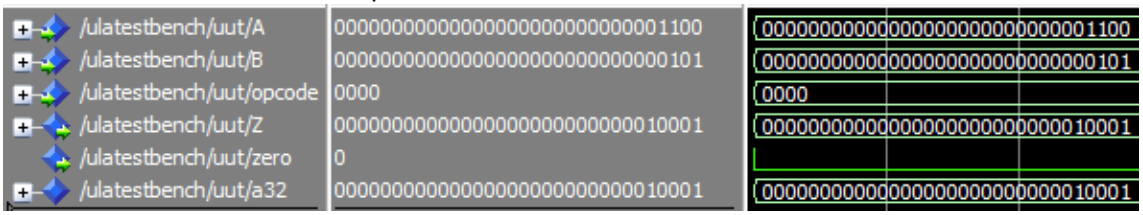
Já as operações sem sinal tratam todos os bits como magnitude, sem distinguir um bit de sinal.

Como se poderia detectar overflow nas operações ADD e SUB?

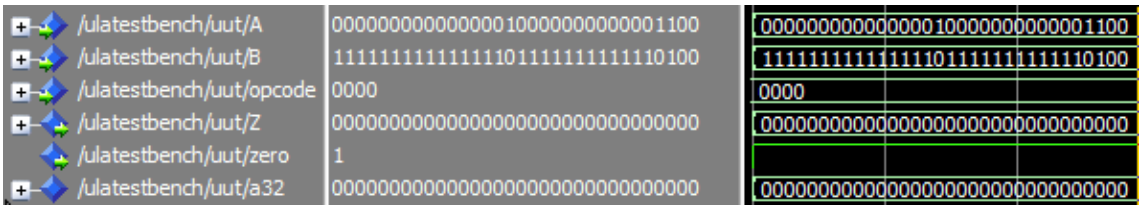
A detecção de *overflow* pode ser realizada observando o *carry-in* e o *carry-out* da posição do bit de sinal. E ocorre um overflow se ambos forem diferentes.

Simulação:

-- Teste 1 - ADD com resultado positivo:



-- Teste 2 - ADD com resultado 0



-- Teste 3 - ADD com resultado negativo

/ulatestbench/uut/A	00000000000000000000000000001111	00000000000000000000000000001111
/ulatestbench/uut/B	11111111111111111111111111110000	11111111111111111111111111110000
/ulatestbench/uut/opcode	0000	0000
/ulatestbench/uut/Z	11111111111111111111111111111111	11111111111111111111111111111111
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	11111111111111111111111111111111	11111111111111111111111111111111

-- Teste 4 - SUB com resultado positivo

/ulatestbench/uut/A	00000000000000000000111100011100	00000000000000000000111100011100
/ulatestbench/uut/B	00000000000000000000000000000101	00000000000000000000000000000101
/ulatestbench/uut/opcode	0001	0001
/ulatestbench/uut/Z	00000000000000000000111100010111	00000000000000000000111100010111
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	00000000000000000000111100010111	00000000000000000000111100010111

-- Teste 5 - SUB com resultado 0

/ulatestbench/uut/A	00000000000000000000000000011000	00000000000000000000000000011000
/ulatestbench/uut/B	00000000000000000000000000011000	00000000000000000000000000011000
/ulatestbench/uut/opcode	0001	0001
/ulatestbench/uut/Z	00000000000000000000000000000000	00000000000000000000000000000000
/ulatestbench/uut/zero	1	
/ulatestbench/uut/a32	00000000000000000000000000000000	00000000000000000000000000000000

-- Teste 6 - SUB com resultado negativo

/ulatestbench/uut/A	00000000000000000000000000001111	00000000000000000000000000001111
/ulatestbench/uut/B	00000000000000000000110100000000	00000000000000000000110100000000
/ulatestbench/uut/opcode	0001	0001
/ulatestbench/uut/Z	11111111111111111111001100001111	11111111111111111111001100001111
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	11111111111111111111001100001111	11111111111111111111001100001111

-- Teste 7 – AND

/ulatestbench/uut/A	00000000000000000100000001010100	00000000000000000100000001010100
/ulatestbench/uut/B	00000000000000110001001000001111	00000000000000110001001000001111
/ulatestbench/uut/opcode	0010	0010
/ulatestbench/uut/Z	00000000000000000000000000000100	00000000000000000000000000000100
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	00000000000000000000000000000100	00000000000000000000000000000100

-- Teste 8 – OR

/ulatestbench/uut/A	00000000000000000100000001010100	00000000000000000100000001010100
/ulatestbench/uut/B	00000000000000110001001000001111	00000000000000110001001000001111
/ulatestbench/uut/opcode	0011	0011
/ulatestbench/uut/Z	00000000000000110101001001011111	00000000000000110101001001011111
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	00000000000000110101001001011111	00000000000000110101001001011111

-- Teste 9 – XOR

/ulatestbench/uut/A	00000000000000000000000000001111	00000000000000000000000000001111
/ulatestbench/uut/B	11111111111111111111111111111111	11111111111111111111111111111111
/ulatestbench/uut/opcode	0100	0100
/ulatestbench/uut/Z	11111111111111111111111111110000	11111111111111111111111111110000
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	11111111111111111111111111110000	11111111111111111111111111110000

-- Teste 10 – SLL

/ulatestbench/uut/A	00000000000000000000111111111111	00000000000000000000111111111111
/ulatestbench/uut/B	00000000000000000000000000001000	00000000000000000000000000001000
/ulatestbench/uut/opcode	0101	0101
/ulatestbench/uut/Z	00000000000011111111111100000000	00000000000011111111111100000000
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	00000000000011111111111100000000	00000000000011111111111100000000

-- Teste 11 – SRL

/ulatestbench/uut/A	11111111111111111111111111111000	11111111111111111111111111111000
/ulatestbench/uut/B	00000000000000000000000000000001	00000000000000000000000000000001
/ulatestbench/uut/opcode	0110	0110
/ulatestbench/uut/Z	01111111111111111111111111111100	01111111111111111111111111111100
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	01111111111111111111111111111100	01111111111111111111111111111100

-- Teste 12 – SRA

/ulatestbench/uut/A	11111111111111111111111111111000	11111111111111111111111111111000
/ulatestbench/uut/B	00000000000000000000000000000001	00000000000000000000000000000001
/ulatestbench/uut/opcode	0111	0111
/ulatestbench/uut/Z	11111111111111111111111111111100	11111111111111111111111111111100
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	11111111111111111111111111111100	11111111111111111111111111111100

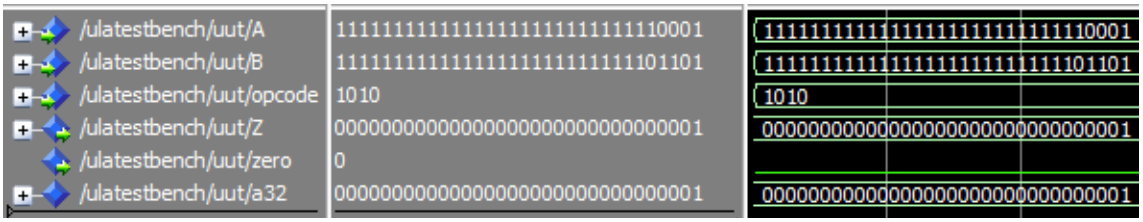
-- Teste 13 – SLT

/ulatestbench/uut/A	11111111111111110010001011101101	11111111111111110010001011101101
/ulatestbench/uut/B	11111111111111110010001101001100	11111111111111110010001101001100
/ulatestbench/uut/opcode	1000	1000
/ulatestbench/uut/Z	00000000000000000000000000000001	00000000000000000000000000000001
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	00000000000000000000000000000001	00000000000000000000000000000001

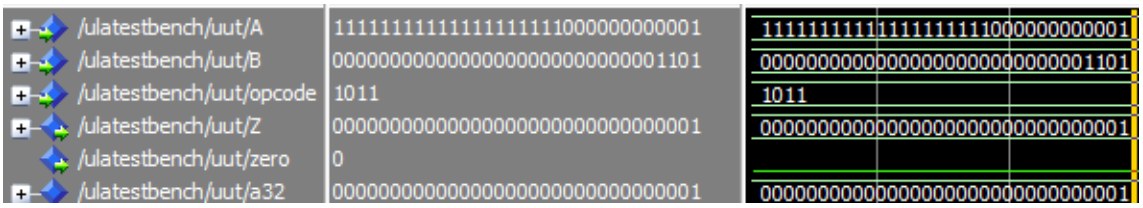
-- Teste 14 – SLTU

/ulatestbench/uut/A	00000000000000000000000011111100	00000000000000000000000011111100
/ulatestbench/uut/B	00000000000000000000111100000000	00000000000000000000111100000000
/ulatestbench/uut/opcode	1001	1001
/ulatestbench/uut/Z	00000000000000000000000000000001	00000000000000000000000000000001
/ulatestbench/uut/zero	0	
/ulatestbench/uut/a32	00000000000000000000000000000001	00000000000000000000000000000001

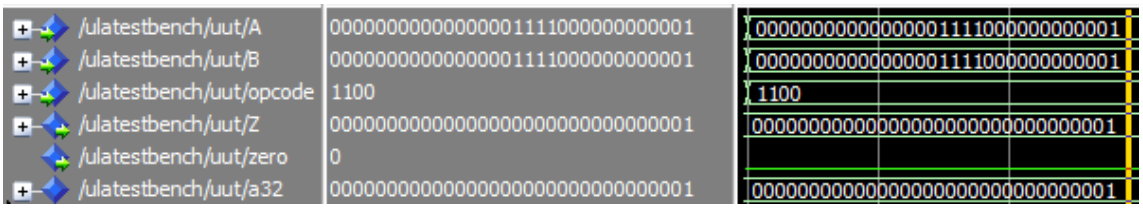
-- Teste 15 – SGE



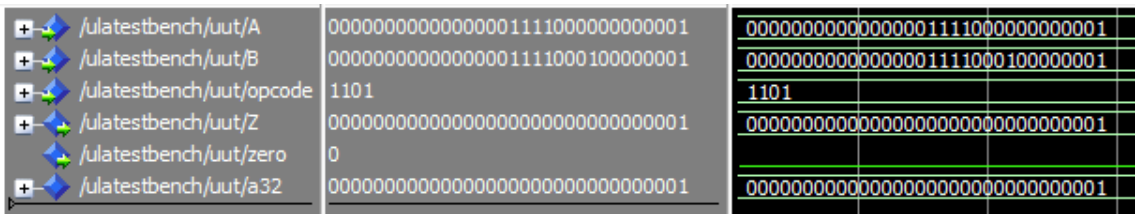
-- Teste 16 – SGEU



-- Teste 17 – SEQ



-- Teste 18 – SNE



Código ulaRiscv.vhd:

```
-- Adriano Ulrich do Prado Wiedmann 202014824
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ulaRiscv is
	generic (WSIZE : natural := 32);
	port (
		A, B		: in std_logic_vector(WSIZE-1 downto 0);
		opcode		: in std_logic_vector(3 downto 0);
		Z		: out std_logic_vector(WSIZE-1 downto 0);
		zero		: out STD_LOGIC);
end ulaRiscv;

architecture behavioral of ulaRiscv is
	signal a32 : std_logic_vector(WSIZE-1 downto 0);
begin
	Z <= a32;
	process (opcode, A, B, a32)
	begin
		if (a32 = X"00000000") then zero <= '1'; else zero <= '0'; end if;
		case opcode is

			-- ADD A, B
			when "0000" => a32 <= std_logic_vector(signed(A) + signed(B));

			-- SUB A, B
			when "0001" => a32 <= std_logic_vector(signed(A) - signed(B));

			-- AND A, B
			when "0010" => a32 <= A AND B;

			-- OR A, B
			when "0011" => a32 <= A OR B;

			-- XOR A, B
			when "0100" => a32 <= A XOR B;

			-- SLL A, B
			when "0101" => a32 <= std_logic_vector(shift_left(unsigned(A), to_integer(unsigned(B)))); -- A SLL B

			-- SRA A, B sem sinal
			when "0110" => a32 <= std_logic_vector(shift_right(unsigned(A), to_integer(unsigned(B)))); -- A SRL B

			-- SRA A, B com sinal
			when "0111" => a32 <= std_logic_vector(shift_right(signed(A), to_integer(signed(B))));

			-- SLT A, B
			when "1000" => a32 <= (others => '0');  -- inicializa com zero
			if signed(A) < signed(B) then
			a32(0) <= '1';
			end if;

			-- SLTU A, B
			when "1001" => a32 <= (others => '0');  -- inicializa com zero
			if unsigned(A) < unsigned(B) then
			a32(0) <= '1';
			end if;

			-- SGE A, B
			when "1010" => a32 <= (others => '0');  -- inicializa com zero
			if signed(A) >= signed(B) then
			a32(0) <= '1';
			end if;

			-- SGEU A, B
			when "1011" => a32 <= (others => '0');  -- inicializa com zero
			if unsigned(A) >= unsigned(B) then
			a32(0) <= '1';
			end if;

			-- SEQ A, B
			when "1100" => a32 <= (others => '0');  -- inicializa com zero
			if A = B then
			a32(0) <= '1';
			end if;

			-- SNE A, B
			when "1101" => a32 <= (others => '0');  -- inicializa com zero
			if A /= B then
			a32(0) <= '1';
			end if;

			when others => a32 <= X"00000000";
		end case;
	end process;
end behavioral;
```

Código ulaTestbench.vhd:

```
-- Adriano Ulrich do Prado Wiedmann 202014824

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ulaTestbench is
end ulaTestbench;

architecture tb_arch of ulaTestbench is
	component ulaRiscv is
		generic (WSIZE : natural := 32);
		port(
			A, B		: in std_logic_vector(WSIZE-1 downto 0);
			opcode		: in std_logic_vector(3 downto 0);
			Z		: out std_logic_vector(WSIZE-1 downto 0);
			zero		: out STD_LOGIC);
	end component;

	signal A_tb, B_tb		: std_logic_vector(31 downto 0);
	signal opcode_tb		: std_logic_vector(3 downto 0);
	signal Z_tb			: std_logic_vector(31 downto 0);
	signal zero_tb			: STD_LOGIC;

begin
	uut: ulaRiscv port map (A => A_tb, B => B_tb,
						opcode => opcode_tb,
						Z => Z_tb,
						zero => zero_tb
						);
	process begin
		-- Teste 1 - ADD com resultado positivo
		A_tb <= X"0000000C";
		B_tb <= X"00000005";
		opcode_tb <= "0000";
		wait for 1 ns;
		assert Z_tb = X"00000011"
			report "Erro - teste 1 - ADD"
			severity error;

		-- Teste 2 - ADD com resultado 0
		A_tb <= X"0001000C";
		B_tb <= X"FFFEFFF4";
		opcode_tb <= "0000";
		wait for 1 ns;
		assert Z_tb = X"00000000"
			report "Erro - teste 2 - ADD"
			severity error;

		-- Teste 3 - ADD com resultado negativo
		A_tb <= X"0000000F";
		B_tb <= X"FFFFFFF0";
		opcode_tb <= "0000";
		wait for 1 ns;
		assert Z_tb = X"FFFFFFFF"
			report "Erro - teste 3 - ADD"
			severity error;

		-- Teste 4 - SUB com resultado positivo
		A_tb <= X"00000F1C";
		B_tb <= X"00000005";
		opcode_tb <= "0001";
		wait for 1 ns;
		assert Z_tb = X"00000F17"
			report "Erro - teste 4 - SUB"
			severity error;

		-- Teste 5 - SUB com resultado 0
		A_tb <= X"00000018";
		B_tb <= X"00000018";
		opcode_tb <= "0001";
		wait for 1 ns;
		assert Z_tb = X"00000000"
			report "Erro - teste 5 - SUB"
			severity error;

		-- Teste 6 - SUB com resultado negativo
		A_tb <= X"0000000F";
		B_tb <= X"00000D00";
		opcode_tb <= "0001";
		wait for 1 ns;
		assert Z_tb = X"FFFFF30F"
			report "Erro - teste 6 - SUB"
			severity error;

		-- Teste 7 - AND
		A_tb <= X"00004054";
		B_tb <= X"0003120F";
		opcode_tb <= "0010";
		wait for 1 ns;
		assert Z_tb = X"00000004"
			report "Erro - teste 7 - AND"
			severity error;

		-- Teste 8 - OR
		A_tb <= X"00004054";
		B_tb <= X"0003120F";
```

```
                opcode_tb <= "0011";
                wait for 1 ns;
                assert Z_tb = X"0003525F"
                        report "Erro - teste 8 - OR"
                        severity error;

                -- Teste 9 - XOR
                A_tb <= X"0000000F";
                B_tb <= X"FFFFFFFF";
                opcode_tb <= "0100";
                wait for 1 ns;
                assert Z_tb = X"FFFFFFF0"
                        report "Erro - teste 9 - XOR"
                        severity error;

                -- Teste 10 - SLL
                A_tb <= X"00000FFF";
                B_tb <= X"00000008";
                opcode_tb <= "0101";
                wait for 1 ns;
                assert Z_tb = X"000FFF00"
                        report "Erro - teste 10 - SLL"
                        severity error;

                -- Teste 11 - SRL
                A_tb <= X"FFFFFFF8";
                B_tb <= X"00000001";
                opcode_tb <= "0110";
                wait for 1 ns;
                assert Z_tb = X"7FFFFFFC"
                        report "Erro - teste 11 - SRL"
                        severity error;

                -- Teste 12 - SRA
                A_tb <= X"FFFFFFF8";
                B_tb <= X"00000001";
                opcode_tb <= "0111";
                wait for 1 ns;
                assert Z_tb = X"FFFFFFFC"
                        report "Erro - teste 12 - SRA"
                        severity error;

                -- Teste 13 - SLT
                A_tb <= X"FFFF22ED";
                B_tb <= X"FFFF234C";
                opcode_tb <= "1000";
                wait for 1 ns;
                assert Z_tb = X"00000001"
                        report "Erro - teste 13 - SLT"
                        severity error;

                -- Teste 14 - SLTU
                A_tb <= X"000000FC";
                B_tb <= X"00000F00";
                opcode_tb <= "1001";
                wait for 1 ns;
                assert Z_tb = X"00000001"
                        report "Erro - teste 14 - SLTU"
                        severity error;

                -- Teste 15 - SGE
                A_tb <= X"FFFFFFF1";
                B_tb <= X"FFFFFFED";
                opcode_tb <= "1010";
                wait for 1 ns;
                assert Z_tb = X"00000001"
                        report "Erro - teste 15 - SGE"
                        severity error;

                -- Teste 16 - SGEU
                A_tb <= X"FFFFF001";
                B_tb <= X"0000000D";
                opcode_tb <= "1011";
                wait for 1 ns;
                assert Z_tb = X"00000001"
                        report "Erro - teste 16 - SGEU"
                        severity error;

                -- Teste 17 - SEQ
                A_tb <= X"0000F001";
                B_tb <= X"0000F001";
                opcode_tb <= "1100";
                wait for 1 ns;
                assert Z_tb = X"00000001"
                        report "Erro - teste 17 - SEQ"
                        severity error;

                -- Teste 18 - SNE
                A_tb <= X"0000F001";
                B_tb <= X"0000F101";
                opcode_tb <= "1101";
                wait for 1 ns;
                assert Z_tb = X"00000001"
                        report "Erro - teste 17 - SNE"
                        severity error;

                wait;
        end process;
end tb_arch;;
```