Projeto e Simulação ULA RISC-V

Gabriel Lucas França do Nascimento
190107111
nascimento.franca@aluno.unb.br

15/01/2025

1 Descrição do Trabalho

O objetivo deste trabalho foi projetar e simular uma Unidade Logica e Aritmética (ULA) de 32 bits para a arquitetura RISC-V, utilizando a linguagem VHDL. A ULA implementada suporta uma variedade de operações aritméticas e logicas, bem como comparações, conforme especificado na documentação. A simulação foi realizada no ambiente EDAPlayground, com o intuito de verificar o correto funcionamento de todas as operaç oes suportadas pela ULA.

2 Diferença entre comparações com e sem sinal

As comparações com sinal consideram os números como valores inteiros com sinal, onde o bit **mais significativo é o bit de sinal** (0 para positivo, 1 para negativo). Por exemplo, a operação `SLT` ***(Set Less Than)*** compara dois operandos como inteiros com sinal e define a saída como 1 se o primeiro operando for menor que o segundo.

Já as comparações sem sinal, tratam os operandos como números inteiros positivos, **independentemente do bit mais significativo.** A operação `SLTU` ***(Set Less Than Unsigned)*** faz a comparação sem levar em conta o sinal dos operandos, **apenas seus valores em binário.**

3 Telas da Simulação

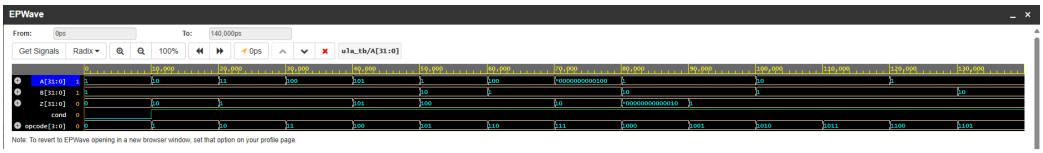

Figure 1: *Waveform* da simulação no EDAPlayground

4 Código da ULA e do Testbench

4.1 Código da ULA (`ulaRV.vhd`)

Listing 1: Código da ULA

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

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

architecture behavior of ulaRV is
signal Z_internal : std_logic_vector(WSIZE-1 downto 0);
begin
process(A, B, opcode)
begin
case opcode is

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

-- SUB
```

```
when "0001" => Z_internal <= std_logic_vector(signed(A) -
signed(B));

-- AND
when "0010" => Z_internal <= A and B;

-- OR
when "0011" => Z_internal <= A or B;

-- XOR
when "0100" => Z_internal <= A xor B;

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

-- SRL
when "0110" => Z_internal <=
std_logic_vector(shift_right(unsigned(A), to_integer(unsigned(B))));

-- SRA
when "0111" => Z_internal <=
std_logic_vector(shift_right(signed(A), to_integer(unsigned(B))));

-- SLT
when "1000" =>
    if signed(A) < signed(B) then
        Z_internal <= (others => '0');
        Z_internal(0) <= '1';
    else
        Z_internal <= (others => '0');
    end if;

-- SLTU
when "1001" =>
    if unsigned(A) < unsigned(B) then
        Z_internal <= (others => '0');
        Z_internal(0) <= '1';
    else
        Z_internal <= (others => '0');
    end if;
```

```
-- SGE
when "1010" =>
    if signed(A) >= signed(B) then
        Z_internal <= (others => '0');
        Z_internal(0) <= '1';
    else
        Z_internal <= (others => '0');
    end if;

-- SGEU
when "1011" =>
    if unsigned(A) >= unsigned(B) then
        Z_internal <= (others => '0');
        Z_internal(0) <= '1';
    else
        Z_internal <= (others => '0');
    end if;

-- SEQ
when "1100" =>
    if A = B then
        Z_internal <= (others => '0');
        Z_internal(0) <= '1';
    else
        Z_internal <= (others => '0');
    end if;

-- SNE
when "1101" =>
    if A /= B then
        Z_internal <= (others => '0');
        Z_internal(0) <= '1';
    else
        Z_internal <= (others => '0');
    end if;

when others => Z_internal <= (others => '0');

end case;
```

```
    Z <= Z_internal;

    cond <= '0';
    for i in Z_internal'range loop
        if Z_internal(i) = '1' then
            cond <= '1';
            exit;
        end if;
    end loop;

end process;

end behavior;
```

4.2 Código do Testbench

Listing 2: Código do Testbench

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ula_tb is
end ula_tb;

architecture tb_arch of ula_tb is
signal opcode : std_logic_vector(3 downto 0);
signal A, B, Z : std_logic_vector(31 downto 0);
signal cond : std_logic;

component ulaRV
    port (
        opcode : in std_logic_vector(3 downto 0);
        A, B   : in std_logic_vector(31 downto 0);
        Z      : out std_logic_vector(31 downto 0);
        cond   : out std_logic
    );
end component;
```

```
begin
uut: ulaRV port map (opcode => opcode,
A => A, B => B, Z => Z, cond => cond);

stimulus: process
begin
    -- ADD
    A <= x"00000001"; B <= x"00000001"; opcode <= "0000";
    wait for 10 ns; -- Saida esperada: Z = 2

    -- SUB
    A <= x"00000002"; B <= x"00000001"; opcode <= "0001";
    wait for 10 ns; -- Saida esperada: Z = 1

    -- AND
    A <= x"00000003"; B <= x"00000001"; opcode <= "0010";
    wait for 10 ns; -- Saida esperada: Z = 1

    -- OR
    A <= x"00000004"; B <= x"00000001"; opcode <= "0011";
    wait for 10 ns; -- Saida esperada: Z = 5

    -- XOR
    A <= x"00000005"; B <= x"00000001"; opcode <= "0100";
    wait for 10 ns; -- Saida esperada: Z = 4

    -- SLL
    A <= x"00000001"; B <= x"00000002"; opcode <= "0101";
    wait for 10 ns; -- Saida esperada: Z = 4

    -- SRL
    A <= x"00000004"; B <= x"00000001"; opcode <= "0110";
    wait for 10 ns; -- Saida esperada: Z = 2

    -- SRA
    A <= x"80000004"; B <= x"00000001"; opcode <= "0111";
    wait for 10 ns; -- Saida esperada: Z = C0000002

    -- SLT
    A <= x"00000001"; B <= x"00000002"; opcode <= "1000";
    wait for 10 ns; -- Saida esperada: Z = 1
```

```
        -- SLTU
        A <= x"00000001"; B <= x"00000002"; opcode <= "1001";
        wait for 10 ns; -- Saida esperada: Z = 1

        -- SGE
        A <= x"00000002"; B <= x"00000001"; opcode <= "1010";
        wait for 10 ns; -- Saida esperada: Z = 1

        -- SGEU
        A <= x"00000002"; B <= x"00000001"; opcode <= "1011";
        wait for 10 ns; -- Saida esperada: Z = 1

        -- SEQ
        A <= x"00000001"; B <= x"00000001"; opcode <= "1100";
        wait for 10 ns; -- Saida esperada: Z = 1

        -- SNE
        A <= x"00000001"; B <= x"00000002"; opcode <= "1101";
        wait for 10 ns; -- Saida esperada: Z = 1

        wait;
    end process;

    end tb_arch;
```

5 Detecção de Overflow

A detecção de *overflow* nas operações `ADD` e `SUB`, é necessário verificar se o bit de sinal dos operandos e o bit de sinal do resultado indicam algum tipo de inconsistência. No caso da soma (`ADD`), o *overflow* ocorre se os dois operandos com o mesmo sinal resultarem em um resultado com sinal oposto. No caso da subtração (`SUB`), o *overflow* ocorre se o minuendo e o subtraendo tiverem sinais diferentes, e o resultado tiver sinal diferente do minuendo. A implementação pode ser realizada utilizando portas XOR para comparar os bits de sinal.