Arquitetura e Organização de Computadores

Projeto e Simulação de uma ULA RISCV em VHDL

Objetivo: projetar, simular e sintetizar uma versão da ULA do RISC-V de 32 bits no ambiente *ModelSim-Altera* ou *EdaPlayground*.

Características:

- Duas entradas de dados: A e B
- Uma Saída de dados: Z
- Sinal *cond*: indica se a condição testada é verdadeira ou falsa - 1/0. Inativo (0) se não for uma operação de comparação (*set equal, set not equal*, etc)
- Operações:

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

Interface:

```
entity ulaRV is
      generics (WSIZE : natural := 32);
      port (
            opcode    : in std_logic(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;
```

onde:

- *opcode* indica a operação a ser realizada
- *A e B*: operandos, 32 bits.
- *Z*: saída, 32 bits
- *cond*: indicação de condição verdadeira

Arquitetura: o funcionamento da ULA pode ser realizado com comandos *with-select* (concorrente, no corpo da arquitetura) ou *case-when* (sequencial, dentro de um processo).

Ex:

```
architecture behavioral of ULA_RV is
    signal a32 : std_logic_vector(31 downto 0);
begin
    Z <= a32;
    proc_ula: process (A, B, opcode, a32) begin
        if (a32 = X"00000000") then zero <= '1'; else zero <= '0'; end if;
        case opcode is
            when ADD_OP  => a32 <= std_logic_vector(signed(A) + signed(B));
            when SUB_OP  => a32 <= std_logic_vector(signed(A) - signed(B));
            when AND_OP  => a32 <= A and B;
            when OR_OP   => a32 <= A or B;
```

Bibliotecas: utilizar a biblioteca *numeric_std* do VHDL para as operações aritméticas sobre os vetores lógicos. As conversões entre vetores e números são ilustradas na imagem anexa.

Simulação e Verificação: simular o funcionamento da ULA de forma a verificar o funcionamento de cada uma das suas operações. Verificar igualmente a geração do sinal *zero*. Utilizar o ModelSim Altera ou EdaPlayground para a simulação, desenvolvendo um *testbench* para acionamento dos sinais.

Testar todas as operações da ULA.

Estrutura 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
```

- declaração de sinais
- declaração do componente ULA

```
begin
```

- instanciação da ULA

- process
    - gera estímulos
    - wait for período (avançar o tempo de simulação)

  end process

end tb_arch;

A verificação deve incluir a execução de ao menos um teste para cada operação da ULA. As operações aritméticas devem ser testadas para resultado zero, negativo e positivo.

Entrega:

- Um relatório contendo:
    - Uma breve descrição do trabalho
    - Explique em suas palavras a diferença entre as comparações com e sem sinal.
    - Telas da simulação com as formas de onda dos sinais
    - Incluir o código da ULA e do *testbench*
    - Como se poderia detectar overflow nas operações ADD e SUB ?