PCS3225 - Sistemas Digitais II
Atividade Formativa 11 - Projeto do Processador PoliLEGv8 em VHDL
Parte 4 - Integração do PoliLEGv8 para Instruções do Tipo "R"

Bruno Abrantes Basseto (2023), revisado por Antonio Vieira da Silva Neto (2024)

Data do Arquivo: 08/11/2024; Prazo da AF11: 23/11/2024

Introdução

O objetivo desta parte da Atividade Formativa 11 é exercitar a integração de componentes VHDL para criar parte do Fluxo de Dados e da Unidade de Controle do processador PoliLEGv8. Os desenvolvimentos realizados nas Partes 1 (Memórias), 2 (Registradores) e 3 (ULA) servirão como referência para a atividade.

Atividades

Atividade Formativa 11 Parte 4, Exercício 1

AF11-P4E1 Implemente um protótipo do Fluxo de Dados do processador PoliLEGv8 em VHDL, interligando o Banco de Registradores e a ULA, para executar as instruções do tipo R (instruções aritméticas e lógicas), como sugerido pela figura 1. Para tanto, crie uma entidade denominada "ULARegs"respeitando a entidade mostrada na Figura 2.

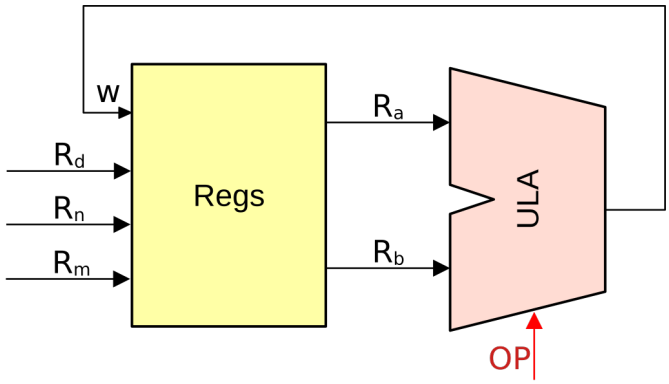

Figura 1: Interligação dos componentes para executar instruções do tipo R

Elabore um *testbench* para fazer a verificação lógica da entidade da Atividade. Defina os **casos de teste** necessários no processo de teste e verificação do projeto.

Atividade Formativa 11 Parte 4 - Exercício 2

AF11-P4E2 Elabore uma bancada de testes (*testbench*) para gerar os sinais "Rd", "Rn" e "Rm" (provenientes da instrução atual), "we" e "clk" (provenientes da Unidade de Controle) para o Banco de Registradores, bem como o sinal "op" para a ULA (proveniente da Unidade de Controle), de forma a simular a execução de algumas instruções do tipo R.

Figura 2: Entidade do Fluxo de Dados "ULARegs"

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

--
-- Agrupamento da ULA e Banco de Registradores
--
entity ULARegs is
port(
        y: out std_logic_vector(63 downto 0);
-- saida da ULA
        op: in std_logic_vector(3 downto 0);
-- operacao a realizar
        zero: out std_logic;
-- indica o resultado zero
        Rd: in std_logic_vector(4 downto 0);
-- indice do registrador a escrever
        Rm: in std_logic_vector(4 downto 0);
-- indice do registrador 1 (ler)
        Rn: in std_logic_vector(4 downto 0);
-- indice do registrador 2 (ler)
        we: in std_logic;
-- habilitacao de escrita
        clk: in std_logic
-- sinal de clock
);
end entity;

-- etc ...
```

Verifique também o correto funcionamento do sinal “zero” e o valor (fixo) de X31=XZR.

Recomenda-se, também, que a bancada de testes tenha integração com a Memória de Instrução desenvolvida na Parte 1 da Atividade Formativa 11. Para tanto, podem ser necessárias alterações no arquivo ".dat"da memória para que ele contenha apenas instruções do tipo "R".

Dica: Na bancada de testes, defina os casos de teste necessários atribuindo valores iniciais aos registradores do banco para poder testá-los, tal como exemplificado na listagem da Figura 3. Também se inspire na descrição comportamental da ULA indicada na Figura 4.

Atividade Formativa 11 Parte 4 - Exercício 3

AF11-P4E3 Submeta, via e-Disciplinas, um arquivo compactado (ZIP) com as produções de todas as quatro partes da Atividade Formativa 11, organizando-as em diretórios por partes e exercícios. Por exemplo:

- Diretório "AF11-P1E1"para o Exercício 1 da Parte 1 da AF11;
- Diretório "AF11-P1E2"para o Exercício 2 da Parte 1 da AF11;

- ...
- Diretório "AF11-P4E2"para o Exercício 2 da Parte 4 da AF11;

Para cada exercício, o diretório deve conter, além dos códigos VHDL, capturas legíveis das simulações realizadas (usando GHDL/GTKWave ou EDAPlayground).

Também deve haver um relatório em PDF com comentários e análises sobre a execução de cada as partes da Atividade Formativa 11. No relatório, inclua comentários sobre a adequação dos resultados de cada parte da Atividade Formativa 11 ao esperado e teça uma conclusão geral sobre a execução dos exercícios (partes cumpridas ou não cumpridas com sucesso; dificuldades enfrentadas pelo grupo, entre outros temas).

Não se esqueçam de incluir a identificação dos nomes de todos os membros do grupo nos arquivos enviados dentro do arquivo ZIP.

Figura 3: Código VHDL para o Banco de Registradores

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

--
-- Banco de 32 Registradores de 64 bits
--
entity banco is
port(
       W: in std_logic_vector(63 downto 0);
-- valor de entrada
        Ra: out std_logic_vector(63 downto 0); -- primeira saida
        Rb: out std_logic_vector(63 downto 0); -- segunda saida
        Rd: in std_logic_vector(4 downto 0);
-- indice do registrador a escrever
       Rm: in std_logic_vector(4 downto 0);
-- indice do registrador 1 (ler)
        Rn: in std_logic_vector(4 downto 0);
-- indice do registrador 2 (ler)
        we: in std_logic;
-- habilitacao de escrita
        clk: in std_logic
-- sinal de clock
);
end entity;

architecture behavior of banco is
        type REGS is array(0 to 31) of std_logic_vector(63 downto 0);
        signal r: REGS := (
                -- alguns valores iniciais para teste:
                0 => x"000000000000AAAA",    -- X0
                1 => x"0000000000005555",    -- X1
                2 => x"0000000000003333",    -- X2
                3 => x"0000000000000001",    -- X3, etc
                others => (others => '0')
        );
begin
        -- saidas
        Ra <= (others => '0') when Rn = "11111" else
              r(to_integer(unsigned(Rn)));
        Rb <= (others => '0') when Rm = "11111" else
              r(to_integer(unsigned(Rm)));

        -- entrada
        r(to_integer(unsigned(Rd))) <= W
           when (we = '1') and rising_edge(clk);
end architecture;
```

Figura 4: Código VHDL para a Unidade Lógico-aritmética

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

--
-- ULA para o LegV8
--
entity ULA is
port(
        y: out std_logic_vector(63 downto 0);
-- saida
        x1, x2: in std_logic_vector(63 downto 0); -- entradas
        op: in std_logic_vector(3 downto 0);
-- operacao a realizar
        zero: out std_logic
-- indica o resultado zero
);
end entity;

architecture behavior of ULA is
        signal op_and, op_or, op_soma,
        op_sub, op_nor: std_logic_vector(63 downto 0);
begin
        -- Operacoes
        op_and <= x1 and x2;
        op_or <= x1 or x2;
        op_soma <= std_logic_vector(unsigned(x1) + unsigned(x2));
        op_sub <= std_logic_vector(unsigned(x1) - unsigned(x2));
        op_nor <= x1 nor x2;

        -- selecao da saida
        with op select y <=
                op_and when "0000",
-- and
                op_or when "0001",                  -- or
                op_soma when "0010",
-- soma
                op_sub when "0110",
-- subtracao
                x2 when "0011",
-- passa x2
                op_nor when "1100",
-- nor
                (others => '0') when others;
-- outros casos

        -- Verifica resultado igual a zero
        zero <= nor y;
        -- outra opcao
        -- zero <= '1' when unsigned(y) = 0 else '0';
end architecture;
```