Documentação Processador VHDL

Grupo: Leonardo Guimarães de Oliveira, Nathan Mateus de Lima, Vitor Oliveira Amorim, Rafael Benfenatti Lima Munhoz

Disciplina: Laboratório de Sistemas Digitais e Computacionais

Professora: Milene Barbosa Carvalho

Data: 19/02/2025

Organização dos Módulos

Módulo	Nome do arquivo VHDL	Aluno(s) responsável(eis) pela implementação
PC	pc	Nathan Lima
Flip flop D com Clear	flipflop_d_c	Nathan Lima
Flip flop D com Preset	flipflop_d_p	Nathan Lima
Banco de registradores	banco_reg	Leonardo Oliveira
Decodificador 5:32	decod_5x32	Nathan Lima
Registrador de 32 bits	flipflop_d_enable_32	Leonardo Oliveira
Registrador de 32 bits com entrada de clear	flipflop_d_clear_enable_32	Leonardo Oliveira
Flip flop D com Enable (e Clear)	flipflop_d_clear_enable	Leonardo Oliveira
Multiplexador de 32 entradas de 32 bits	mux_32_x_32	*código disponibilizado*
Multiplexador de 2 entradas de 32 bits	mux_2_x_32	Vítor Amorim e Rafael Benfenatti
Multiplexador de 2 entradas de 5 bits	mux_2_x_5	Vítor Amorim e Rafael Benfenatti
ULA de 32 bits	Ula_32	Vítor Amorim
ULA de 1 bit	ula_1	Vítor Amorim
OR de 32 entradas	or_32	Vítor Amorim e Nathan Lima
Multiplexador de 2 entradas de 1 bit	mux_2	*código disponibilizado*

Multiplexador de 4 entradas de 1 bit	mux_4	Nathan Lima
Somador completo	somador_completo	Vítor Amorim, Rafael Benfenatti, Nathan Lima e Leonardo Oliveira
Somador de 32 bits	Somador32bits	Vítor Amorim, Rafael Benfenatti, Nathan Lima e Leonardo Oliveira
Extensor de sinal	extensor_sinal	Nathan Lima
Deslocador 2 bits à esquerda	deslocador_2b_esq	Nathan Lima
Unidade de controle	control	Nathan Lima, Rafael Benfenatti
Unidade de controle da ULA	alu_control	Nathan Lima, Rafael Benfenatti
Caminho de dados do MIPS	caminho_de_dados	Vitor Amorim, Leonardo Oliveira, Nathan Lima, Rafael Benfenatti
Flip flop D com Enable	flipflop_d_enable	Leonardo Oliveira
Tipo	tipo	*código disponibilizado*

Documentação dos Módulos

Módulo ULA de 1 bit

O módulo "*ula_1*" implementa uma unidade lógica e aritmética (ULA) de 1 bit com as operações de AND, OR, soma e SLT.

Entradas:

- 'A' (*std_logic*): Primeiro operando da ULA;
- 'B' (*std_logic*): Segundo operando da ULA;
- 'Ainvert' (*std_logic*): Determina se o sinal do primeiro operando deverá ser negado (1) ou não (0);
- 'Binvert' (*std_logic*): Determina se o sinal do segundo operando deverá ser negado (1) ou não (0);
- 'VemUm' (*std_logic*): Determina o CarryIn da ULA anterior;
- 'Operacao' (*std_logic_vector(1 downto 0)*): Determina o resultado que será escolhido para a saída;
- 'Less' (*std_logic*): Utilizado para determinar se o primeiro operando é menor que o segundo.

Saídas:

- 'Resultado' (*std_logic*): Resultado da operação realizada entre os dois operandos;
- 'VaiUm' (*std_logic*): Determina o CarryOut da operação, que será redirecionado para a próxima ULA;
- 'Set' (*std_logic*): Saída para determinar se o primeiro operando é menor que o segundo.

Módulo Deslocador de 2 bits para a Esquerda (32x32)

O módulo "*deslocador_2b_esq*" realiza um deslocamento de 2 bits à esquerda em um vetor de 32 bits de entrada. Após o deslocamento, os 2 bits menos significativos da saída são preenchidos com 0.

Entradas:

- 'entrada' (*std_logic_vector(31 downto 0)*): Vetor de 32 bits de entrada que será deslocado em 2 bits para a esquerda.

Saídas:

- 'saida' (*std_logic_vector(31 downto 0)*): O vetor de 32 bits da entrada deslocado em 2 bits para a esquerda.

Módulo Deslocador de 2 bits para a Esquerda (26x28)

O módulo "*deslocador_2b_esq_26x28*" realiza um deslocamento de 2 bits à esquerda em um vetor de 26 bits de entrada. Após o deslocamento, os 2 bits menos significativos da saída são preenchidos com 0.

Entradas:

- 'entrada' (*std_logic_vector(25 downto 0)*): Vetor de 26 bits de entrada que será deslocado em 2 bits para a esquerda.

Saídas:

- 'saida' (*std_logic_vector(27 downto 0)*): O vetor de 28 bits da entrada deslocado em 2 bits para a esquerda.

Módulo: Somador de 32 bits

O módulo "*Somador32bits*" realiza a soma de dois vetores de 32 bits da entrada e oferece a soma entre os dois na saída. Nesta implementação, não existe verificação de overflow.

Entradas:

- 'A' (*std_logic_vector(31 downto 0)*): Vetor de 32 bits com o primeiro número da operação.
- 'B' (*std_logic_vector(31 downto 0)*): Vetor de 32 bits com o segundo número da operação.

Saídas:

- 'Soma' (*std_logic_vector(31 downto 0)*): Vetor de 32 bits resultante da soma entre as duas entradas A e B.

Módulo ULA de 32 bits

O módulo "*Ula_32*" implementa uma Unidade Lógica Aritmética (ULA) de 32 bits, que pode realizar as operações simples de AND, OR, soma e set on less than (SLT), porém pode realizar as demais operações com a combinação desses procedimentos.

São realizadas todas as operações da ULA em um mesmo ciclo de clock, porém somente o resultado de uma delas é selecionada, de acordo com a operação especificada na entrada.

Entradas:

- 'A' (*std_logic_vector(31 downto 0)*): Vetor de 32 bits com o primeiro operando;

- ‘B’ (*std_logic_vector(31 downto 0)*): Vetor de 32 bits com o segundo operando;
- ‘Anegate’ (*std_logic*): Determina se o sinal do primeiro operando deve ser negado (1) ou não (0);
- ‘Bnegate’ (*std_logic*): Determina se o sinal do segundo operando deve ser negado (1) ou não (0);
- Operacao (*std_logic_vector(1 downto 0)*): Determina o tipo de operação a ser feita.

Saídas:

- ‘Resultado’ (*std_logic_vector(31 downto 0)*);
- Zero(std_logic): Se é zero fica em 1, do contrário fica em 0.

Módulo Unidade de Controle da ULA

O módulo “*alu_control*” implementa uma Unidade de Controle da ULA, a partir do alu_op gerado pela Unidade de Controle e do código de instrução de 6 bits, ele determina os valores de Anegate, Bnegate e Operação, valores essenciais para determinar a operação a ser feita e se o valor deve ser positivo ou negativo.

Entradas:

- ‘funct’ (*std_logic_vector(5 downto 0)*): Código que determina o tipo de função MIPS;
- ‘alu_op’ (*std_logic_vector(1 downto 0)*): Código que auxilia para determinar o tipo de operação;

Saídas:

- ‘a_inverte’ (*std_logic*): Determina se o sinal do primeiro operando vai ser negado (1) ou não (0);
- ‘b_inverte’ (*std_logic*): Determina se o sinal do segundo operando vai ser negado (1) ou não (0);
- ‘operacao’ (*std_logic_vector(1 downto 0)*): Determina o tipo de operação a ser feita na ULA.

Módulo Banco de Registradores

O módulo “*banco_reg*” implementa o Banco de Registradores, ele recebe dois endereços, endereço que vai escrever, dados que vai escrever, e a saída dos dois que leu.

Entradas:

- ‘reg1_addr’ (*std_logic_vector(4 downto 0)*): Endereço de leitura do primeiro registrador;
- ‘reg2_addr’ (*std_logic_vector(4 downto 0)*): Endereço de leitura do segundo registrador;
- ‘write_data’ (*std_logic_vector(31 downto 0)*): Dados recebidos para escrita;
- ‘clock’ (*std_logic*): clock.
- ‘reg_write’ (*std_logic*): Determina se o dado será ou não escrito no registrador de destino

Saídas:

- ‘read_reg1’ (*std_logic_vector(31 downto 0)*): Primeiro dado lido da memória;
- ‘read_reg2’ (*std_logic_vector(31 downto 0)*): Segundo dado lido da memória.

Módulo Unidade de Controle

O módulo “*control*” implementa uma Unidade de Controle, a qual gera os sinais usados nos muxes para determinar o que deve ser escolhido.

Entradas:

- ‘opcode’ (*std_logic_vector(5 downto 0)*): Código que determina o tipo de instrução recebida;

Saídas:

- ‘alu_op’ (*std_logic_vector(5 downto 0)*): Código que auxilia a alu_control na decisão do tipo de operação ;
- ‘reg_write’, ‘reg_dst’, ‘alu_src’, ‘branch’, ‘mem_write’, ‘mem_to_reg’, ‘jump’, ‘mem_read’ (*std_logic*): Sinais mandados para escolher comportamentos de módulos.

Módulo Flip flop D com Clear

O módulo “*flipflop_d_c*” implementa um Flip flop D ativo em borda de subida com Clear.

Entradas:

- ‘D’ (*std_logic*): Entrada D do flip flop;
- ‘Clear’ (*std_logic*): Entrada de clear, caso ativa coloca Q como 0;
- ‘clk’ (*std_logic*): Clock;

Saída:

- ‘Q’ (*std_logic*): Valor de saída, pode ser o valor de D ou 0;

Módulo Flip flop D com Preset

O módulo "*flipflop_d_p*" implementa um Flip flop D ativo em borda de subida com Preset.

Entradas:

- 'D' (std_logic): Entrada D do flip flop;
- 'Preset' (std_logic): Entrada de clear, caso ativa coloca Q como 1;
- 'clk' (std_logic): Clock;

Saída:

- 'Q' (std_logic): Valor de saída, pode ser o valor de D ou 1;

Módulo Extensor de sinal

O módulo "*extensor_sinal*" implementa um extensor de sinal de uma entrada de 16 bits para 32 bits.

Entradas:

- 'input_16' (*std_logic_vector(15 downto 0)*): Um número de 16 bits.

Saídas:

- 'output_32' (*std_logic_vector(31 downto 0)*): Saída do extensor de sinal, representa o número inserido na entrada extendido para 32 bits.

Módulo Decodificador 5:32

O módulo "*decod_5x32*" implementa um decodificador com 5 bits de entrada e 32 de saída.

Entradas:

- 'Entrada' (*std_logic_vector(4 downto 0)*): Número de 5 bits a ser decodificado pelo decodifcador.

Saídas:

- 'Saída' (*std_logic_vector(31 downto 0)*): Saída decodifcada pelo decodificador.

Flip-flop D com Clear e Enable, ativo em borda de descida

O módulo "*flipflop_d_clear_enable*" implementa um flip flop D com clear e enable ativo em borda de descida.

Entradas:

- ‘D’ (std_logic): Entrada D do flipflop.
- ‘Enable’ (std_logic): Define se o flipflop está ativo ou não.
- ‘clk’ (std_logic): Entrada de clock do flipflop.

Saídas:

- ‘Q’ (std_logic): Saída Q do flipflop.

Módulo Registrador de 32 bits com clear

O módulo “*flipflop_d_clear_enable_32*” implementa um flip flop D com clear e enable ativo em borda de descida de 32 bits a fim de atuar como um registrador de 32 bits com clear.

Entradas:

- ‘D’ (*std_logic_vector(31 downto 0)*): Entrada D de 32 bits do flipflop.
- ‘Enable’ (*std_logic*): Define se o flipflop está ativo ou não.
- ‘Clear’ (*std_logic*): Entrada clear do flipflop
- ‘clk’ (*std_logic*): Entrada de clock do flipflop.

Saídas:

- ‘Q’ (*std_logic_vector(31 downto 0)*): Saída Q de 32 bits do flipflop.

Módulo Registrador de 32 bits

O módulo “*flipflop_d_enable_32*” implementa um flip flop D com clear e enable ativo em borda de descida de 32 bits a fim de atuar como um registrador de 32 bits com clear.

Entradas:

- ‘D’ (*std_logic_vector(31 downto 0)*): Entrada D de 32 bits do flipflop.
- ‘Enable’ (*std_logic*): Define se o flipflop está ativo ou não.
- ‘clk’ (*std_logic*): Entrada de clock do flipflop.

Saídas:

- ‘Q’ (*std_logic_vector(31 downto 0)*): Saída Q de 32 bits do flipflop.

Módulo OR de 32 entradas

O módulo “*or_32*” implementa a operação OR de 32 entradas para uma única saída. O módulo faz uso de múltiplas operações OR de 2 bits em árvore para implementar a operação com 32 entradas e saída de 1 bit.

Entradas:

- E (*std_logic_vector(31 downto 0)*): As 32 entradas da operação.

Saídas:

- Saída (*std_logic*): Resultado de 1 bit da operação OR entre todas as entradas.

Módulo Somador Completo

O módulo "*somador_completo*" implementa a soma de dois operandos de 1 bit da entrada. Além dos dois operandos, o somador completo suporta uma entrada de CarryIn. Na saída, são passados um sinal de CarryOut e o resultado.

Entradas:

- A (*std_logic*): Primeiro operando de 1 bit;
- B (*std_logic*): Segundo operando de 1 bit;
- VemUm (*std_logic*): CarryIn da operação.

Saídas:

- Soma (*std_logic*): Resultado da operação do somador;
- VaiUm (*std_logic*): CarryOut da operação de soma.

Módulo Mux de 4 entradas de 1 bit

O módulo "*mux_4*" implementa um mux composto por 4 entradas de 1 bit.

Entradas:

- E (*std_logic_vector(3 downto 0)*): Entradas do mux que poderão ser selecionadas;
- Sel (*std_logic_vector(1 downto 0)*): Entrada de seleção do mux.

Saídas:

- Saida (*std_logic*): Saída de 1 bit selecionada de acordo com os bits de entrada do seletor.

Módulo Mux de 32 entradas de 32 bits

O módulo "*mux_32_x_32*" implementa um mux composto por 32 entradas de 32 bits.

Entradas:

- entradas (*std_logic_vector(0 to 31)*): As 32 entradas de 32 bits do mux;
- Sel (*std_logic_vector(1 downto 0)*): Entrada de seleção do mux, composta por 5 bits.

Saídas:

- saida (*std_logic_vector(31 downto 0)*): Saída de 32 bit selecionada de acordo com os bits de entrada do seletor.

Módulo Mux de 2 entradas de 5 bits

O módulo "*mux_2_x_5*" implementa um mux composto por 2 entradas de 5 bits cada utilizando o generate.

Entradas:

- A (*std_logic_vector(4 downto 0)*): Primeira entrada de 5 bits do mux;
- B (*std_logic_vector(4 downto 0)*): Segunda entrada de 5 bits do mux;
- Sel (*std_logic*): Bit de seleção do mux.

Saídas:

- Saida (*std_logic_vector(4 downto 0)*): Saída de 5 bits selecionada a partir da entrada do seletor.

Módulo Mux de 2 entradas de 32 bits

O módulo "*mux_2_x_32*" implementa um mux composto por 2 entradas de 32 bits cada, utilizando o generate.

Entradas:

- A (*std_logic_vector(31 downto 0)*): Primeira entrada de 32 bits do mux;
- B (*std_logic_vector(31 downto 0)*): Segunda entrada de 32 bits do mux;
- Sel (*std_logic*): Bit de seleção do mux.

Saídas:

- Saida (*std_logic_vector(31 downto 0)*): Saída de 32 bits selecionada a partir da entrada do seletor.

Módulo Mux de 2 entradas de 1 bit

O módulo "*mux_2*" implementa um mux composto por 2 entradas de 1 bit.

Entradas:

- E0 (*std_logic*): Primeira entrada de 1 bit do mux;

- E1 (*std_logic*): Segunda entrada de 1 bit do mux;
- Sel (*std_logic*): Bit de seleção do mux.

Saídas:

- Saida (*std_logic*): Saída de 1 bit selecionada a partir da entrada do seletor.

Módulo Registrador PC

O módulo "*pc*" implementa um Registrador PC, o qual quando com valor reset, reseta o registrador para o valor 0x0000300. O registrador possui entradas de enable, clear e preset também.

Entradas:

- clk (*std_logic*): Clock;
- reset (*std_logic*): Reseta o registrador para o valor 0x0000300.
- d_in (*std_logic_vector(31 downto 0)*): Entrada de dados de 32 bits.

Saídas:

- q_out (*std_logic_vector(31 downto 0)*): Saída de 32 bits do registrador.

Módulo Memória de Instruções

O módulo "*memInstruções*" implementa uma memória de instruções

Entradas:

- Endereço (std_logic_vector(31 downto 0)): Endereço a ser buscado na memória de instruções;

Saídas:

- Palavra (std_logic_vector(31 downto 0)): Instrução relativa ao endereço de entrada;

Módulo Memória de Dados

O módulo "*memDados*" implementa uma memória de dados com 256 palavras de 32 bits.

Entradas:

- DadoEscrita (std_logic_vector(31 downto 0)): Dado a ser escrito na memória de dados;
- Endereço (std_logic_vector(31 downto 0)): Endereço a escrever o dado;

- EscreverMem (std_logic): Sinal para definir se é para ser escrito o valor na memória;
- Clock (std_logic): Sinal de clock;
- LerMem (std_logic): Sinal para determinar se o dado no endereço deve ser lido da memória;
- DebugEndereço (std_logic_vector(31 downto 0)): Vetor de debug do endereço de entrada da memória de dados.

Saídas:

- DebugPalavra (std_logic_vector(31 downto 0)): Vetor de debug da palavra da memória de dados;
- DadoLido (std_logic_vector(31 downto 0)): Vetor com o dado lido da memória.

Módulo Caminho de Dados

O módulo "*caminho_de_dados*" implementa o caminho de dadas do processador MIPS em VHDL.

Entradas:

- CLK (*std_logic*): Clock do processador;
- pc_reset (*std_logic*): Reseta o endereço do PC, retornando ao endereço inicial;
- debugEndereco (*std_logic_vector(31 downto 0)*): Entrada de debug de endereço de entrada da memória de dados.

Saídas:

- debugPalavra (*std_logic_vector(31 downto 0)*): Debug da palavra de saída da memória de dados.

Principais dificuldades e observações

As principais dificuldades do grupo estiveram relacionadas à escolha de qual maneira de implementar cada módulo do processador, uma vez que a linguagem descritiva VHDL possui várias maneiras de implementar um mesmo circuito. Também tivemos dificuldades na construção do caminho de dados, uma vez que a identificação de erros em sua implementação é complicada.

No caso do módulo Program Counter (PC) tivemos dificuldades em assimilar a dica dada para a sua implementação e em relação ao mapeamento dos flip-flops.

O mais interessante do projeto foi a possibilidade de implementar circuitos grandes com menos trabalho, se comparado com o simulador Digital, embora o simulador forneça uma experiência visual mais satisfatória ao engenheiro do projeto.