Universidade Federal de Ouro Preto - UFOP

Projeto prático CSI509 - Implementação ULA e a sua Unidade de Controle

Guilherme Ferreira de Souza Sobrinho n° 19.2.8981

Graziele de Cássia Rodrigues n° 21.1.8120

Julho de 2023

Sumário

1 Resumo

A unidade lógica e aritmética (ULA) é um bloco que realiza operações lógica e artiméticas. Tipicamente, uma ULA recebe dois operandos como entrada, e uma entrada auxiliar de controle que permite especificar qual operação deverá ser realizada. Com o objeitvo de aplicar conceitos aprendidos na disciplina de CSI509 foi implementado, por meio da linguaguem de descrição de máquina - Verilog, uma ULA capaz de executar as instruções load word (lw), store word (sw), branch equal (beq) e as instruções lógicas e aritméticas add, sub, AND, OR e set on less than.

2 Apresentação

O projeto foi implementado com base na tabela abaixo.

Opcode da instrução	OpALU	Operação da instrução	Campo funct	Ação da ALU desejada	Entrada do controle da ALU
LW	00	load word		add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
tipo R	10	add	100000	add	0010
tipo R	10	subtract	100010	subtract	0110
tipo R	10	AND	100100	AND	0000
tipo R	10	OR	100101	OR	0001
tipo R	10	set on less than	101010	set on less than	0111

Figura 1: Tabela com orientação para implementação

A figura acima específica a forma como os bits de controle da ALU são definidos, dependendo dos bits de controle de ALUOp e dos diferentes códigos de função para as instruções tipo R. O opcode, que aparece na primeira coluna, determina a definição dos bits de ALUOp. Todas as codificações são mostradas em binário. Quando ALUOp é 00 ou 01, a ação da ALU desejada não depende do campo de código de função, ou seja, 'don't care' e, portanto, o valor do código de função e o campo funct aparece como XXXXXX. Quando o valor de ALUOp é 10, então o código de função é usado para definir a entrada do controle da ALU.

3 Descrição de atividades

3.1 Ferramentas utilizadas

1. Icarus Verilog: ferramenta open source de simulação que funciona como um compilador de código fonte escrito em Verilog (IEEE-1364).

2. ChipVerify: site com documentação para auxiliar o aprendizado sobre verilog, nele encontra-se diversos tutoriais.

3. Materias de aula e Livro Organização e Projeto de Computadores de John Hennessy

4. Visual Studio Code: editor de texto escolhido.

3.2 Implementação

Tendo como base a figura 1, e sabendo que a ULA está sendo desenvolvida para a arquitetura MIPS, deve-se receber 2 palavras de 32 bits e retornar uma palavra de 32 bits.

Foi disponibilizado pelo professor os arquivos bases contendo os módulos e seus testbench. Dessa forma, a dupla precisou realizar a descrição comportalmental para o funcionamento dos módulos contidos nos arquivos:

- ULA.v
- ULAControl.v
- MIPS.v

3.2.1 ULA.v

O módulo recebe como entradas um clock, duas palavras de 32 bits e um sinal de controle de 4 bits e retorna como saída uma palavra de 32 bits.

Seu comportamento foi descrito utilizando um bloco que é sempre acionado nas bordas de subidas do clock. Dentro desse block, utiliza-se uma comparação que seleciona, a depender do controle recebido, qual operação deve ser executada pela ULA, sendo os seguintes casos:

- 4'b0010: executa uma operação adição com os valores de "a"e "b"e atribui a saída da ULA. Lembrando que essa operação também é usada no LW E SW.
- 4'b0110: executa uma operação subtração com os valores de "a"e "b"e atribui a saída da ULA. Lembrando que essa operação também é usada no BEQ.
- 4'b0000: executa uma operação AND com os valores de "a"e "b"e atribui a saída da ULA.
- 4'b0001: executa uma operação OR com os valores de "a"e "b"e atribui a saída da ULA.
- 4'b0111: executa uma operação "set on less than"com os valores de "a"e "b"e atribui a saída da ULA. Retorna 1 se "a"for menor que "b".
- default: atribui 0 a saída da ULA.

```
module ULA (input         clk,
            input  [0:3]  inputULA,
            input  [0:31] a,
            input  [0:31] b,
            output reg [0:31] outputULA);

    always @(posedge clk) begin
        case(inputULA) // se o input da ULA for 0010 (4'b0010), segundo a tabela, a operação a ser feita é add
            4'b0010: outputULA <= a + b; //add
            4'b0110: outputULA <= a - b; //subtract
            4'b0000: outputULA <= a & b; //AND
            4'b0001: outputULA <= a | b; //OR
            4'b0111: outputULA <= (a < b) ? 1 : 0;
            default: outputULA <= 0;
        endcase
    end

endmodule
```

Figura 2: Implementação ULA.v

```
clk=0, inputULA=0010, a=00000000000000000000000000000011, b=00000000000000000000000000000011, outputULA=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxx
clk=1, inputULA=0010, a=00000000000000000000000000000011, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000000
000110
clk=0, inputULA=0110, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000000
000110
clk=1, inputULA=0110, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000000
000010
clk=0, inputULA=0000, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000000
000010
clk=1, inputULA=0000, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000001

clk=0, inputULA=0001, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000001

clk=1, inputULA=0001, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000011

clk=0, inputULA=0111, a=00000000000000000000000000000001, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000011

clk=1, inputULA=0111, a=00000000000000000000000000000001, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000001
```

Figura 3: Resposta ao testbench para ULA.v

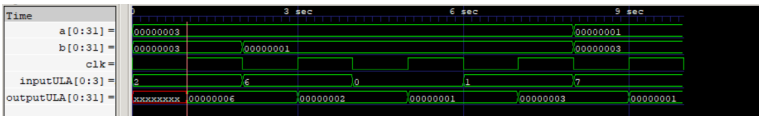

Figura 4: Ondas observardas no gtkwave para ULA.v

3.2.2 ULAControl.v

O controle da ULA é responsável por selecionar a operação que deve ser executada pela ULA. Seu módulo tem 3 entradas e 1 saída: clk, OpALU, funct e inputALU. O OpALU possuí 2 bits que indicam o tipo de instrução (LW ou SW, tipo R ou Branch Equal), o campo funct possuí 6 bits e ele representa a operação da instrução, não sendo usado para os tipos LW, SW e Branch equal, já a saída da ULAControl tem 4 bits e é uma entrada para a ULA.

Seu comportamento foi descrito de forma semelhante ao da ULA.v, usando um bloco always que contém os seguintes casos:

- 2'b00: o tipo de instrução é LW ou SW, logo inputALU é definido como 4'b0010.
- 2'b01: o tipo de instrução é Branch equal, logo inputALU é definido como 4'b0110.
- 2'b10: o tipo de instrução é R, portanto é preciso testar os campos

functs para definir o inputALU. Foram testados os seguintes:

- 6'b100000: o tipo de instrução é add, logo inputALU é definido como 4'b0010.
- 6'b100010: o tipo de instrução é subtract, logo inputALU é definido 4'b0110.
- 6'b100100: o tipo de instrução é AND, logo inputALU é definido como 4'b0000.
- 6'b100101: o tipo de instrução é OR, logo inputALU é definido como 4'b0001.
- 6'b101010: o tipo de instrução é STL, logo inputALU é definido como 4'b0111.
- default: inputALU é definido como 4'b0000.

- default: inputALU é definido como 4'b0000.

```
module ULAControl(input           clk,
                  input      [0:1] OpALU,
                  input      [0:5] funct,
                  output reg [0:3] inputALU);


    always @(negedge clk) begin
        case (OpALU)
            2'b00: inputALU <= 4'b0010; //4 bits, add, porque é LW e/ou SW
            2'b01: inputALU <= 4'b0110; //4 bits, branch equal
            2'b10: case (funct)
                6'b100000: inputALU <= 4'b0010; //add
                6'b100010: inputALU <= 4'b0110; //subtract
                6'b100100: inputALU <= 4'b0000; //AND
                6'b100101: inputALU <= 4'b0001; //OR
                6'b101010: inputALU <= 4'b0111; //slt
                default: inputALU <= 4'b0000;
            endcase
            default: inputALU <= 4'b0000;
        endcase
    end
endmodule
```

Figura 5: Implementação ControlULA.v

```
clk=0, OpALU=00, funct=000000, inputALU=0010
clk=1, OpALU=00, funct=000000, inputALU=0010
clk=0, OpALU=01, funct=000000, inputALU=0110
clk=1, OpALU=01, funct=000000, inputALU=0110
clk=0, OpALU=10, funct=100000, inputALU=0010
clk=1, OpALU=10, funct=100000, inputALU=0010
clk=0, OpALU=10, funct=100010, inputALU=0110
clk=1, OpALU=10, funct=100010, inputALU=0110
clk=0, OpALU=10, funct=100100, inputALU=0000
clk=1, OpALU=10, funct=100100, inputALU=0000
clk=0, OpALU=10, funct=100101, inputALU=0001
clk=1, OpALU=10, funct=100101, inputALU=0001
clk=0, OpALU=10, funct=101010, inputALU=0111
clk=1, OpALU=10, funct=101010, inputALU=0111
ULAControl_tb.v:37: $finish called at 14 (1s)
clk=0, OpALU=10, funct=101010, inputALU=0111
```

Figura 6: Resposta ao testbench para ControlULA.v

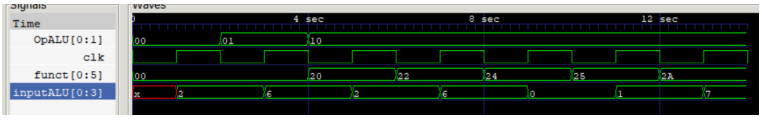

Figura 7: Ondas observardas no gtkwave para ControlULA.v

3.2.3 MIPS.v

Por fim, o módulo MIPS.v foi implementado. Esse módulo é responsável por fazer a integração de ULA.v e ControlULA.v. Ele possuí 5 portas de entradas, sendo o clock, as palavras "a"e "b"com 32 bits, o funct com 6 bits, a entrada o sinal de controle, o qual será conectado por um jumper. E possuí a saída com 32 bits com a operação lógia ou aritmética executada.

Sua descrição comportamental foi feito apenas por meio da instanciação dos módulos. Veja o código abaixo:

```
module MIPS(input              clk,
            input      [0:1]   OpALU,
            input      [0:5]   funct,
            input      [0:31]  a,
            input      [0:31]  b,
            output     [0:31]  outputULA);


    wire [0:3] inputALU;

    //Conectando Jumper de ControlULA
    ULAControl ULAControl_MIPS(
        .clk(clk),
        .OpALU(OpALU),
        .funct(funct),
        .inputALU(inputALU)
    );

    //Conectando Jumper de ULA
    ULA ULA_MIPS(
        .clk(clk),
        .inputULA(inputALU),
        .a(a),
        .b(b),
        .outputULA(outputULA)
    );

endmodule
```

Figura 8: Implementação Mips.v

4 Análise dos Resultados

Abaixo temos os resultados finais obtidos.

```
clk=0, OpALU=00, funct=000000, a=00000000000000000000000000000011, b=00000000000000000000000000000011, outputULA=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxx
clk=1, OpALU=00, funct=000000, a=00000000000000000000000000000011, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000000
000110
clk=0, OpALU=10, funct=100000, a=00000000000000000000000000000001, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000110
clk=1, OpALU=10, funct=100000, a=00000000000000000000000000000001, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000100
clk=0, OpALU=10, funct=100010, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000100
clk=1, OpALU=10, funct=100010, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000010
clk=0, OpALU=10, funct=100100, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000010
clk=1, OpALU=10, funct=100100, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000001
clk=0, OpALU=10, funct=100101, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000001
clk=1, OpALU=10, funct=100101, a=00000000000000000000000000000011, b=00000000000000000000000000000001, outputULA=00000000000000000000000000000011
clk=0, OpALU=10, funct=101010, a=00000000000000000000000000000001, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000011
clk=1, OpALU=10, funct=101010, a=00000000000000000000000000000001, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000001
MIPS_tb.v:66: $finish called at 12 (1s)
clk=0, OpALU=10, funct=101010, a=00000000000000000000000000000001, b=00000000000000000000000000000011, outputULA=00000000000000000000000000000001
```

Figura 9: Saída no terminal

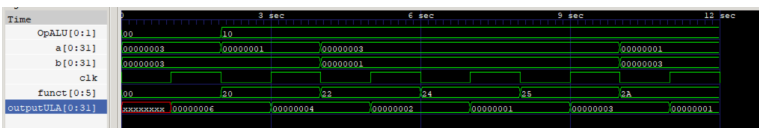

Figura 10: Visualização GTKWAVE

Percebe-se que funcionou como o esperado, a depender do controle recebido a operação é realizada corretamente.

5 Conclusão

O projeto implementa uma ULA para a arquitetura MIPS usando a linguagem de descrição de hardware Verilog. Foram feitos a construção de 3 módulos, sendo ULA.v capaz de executar 5 operações (AND, OR, add, subtract e set on less), ControlULA.v que a partir dos sinais de controle seleciona o tipo de operação que a ULA deve executar e MIPS.v que realiza a integração.

O funcionamento da ULA MIPS foi verificado utilizando os testbench disponibilizados pelo professor e concluiu que a execução funcionou como esperado, ou seja, o projeto foi bem sucedido, atendendo todas as funcionalidades propostas.

Desenvolver esse trabalho possibilitou com que a dupla aprofundasse e aplicasse os conhecimentos adquiridos em sala de aula.

Bibliografia

David A. Patterson, John L. Hennessy. Organização e Projeto de Computadores. 5° edição. GEN LTC. 2017.