

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DCC819 - Arquitetura de Computadores

Relatório I - Unidade Lógica Aritmética

Guilherme Batista Santos
Iuri Silva Castro
João Mateus de Freitas Veneroso
Ricardo Pagoto Marinho

BELO HORIZONTE - MG
10 DE OUTUBRO DE 2017

1 Introdução

O presente trabalho foca na descrição da implementação de uma Unidade Lógica e Aritmética (ULA) em Verilog HDL, uma Linguagem de descrição de hardware, *Hardware Description Language* - HDL - em inglês. A ULA é um circuito digital responsável por realizar operações lógicas e aritméticas no caminho de dados de uma CPU. É a ULA que realiza operações como adição, subtração e operações lógicas como *and* e *or*. Além disso, ela também é responsável por calcular o endereço de memória para escrita ou leitura quando as instruções requisitam. Para implementar a ALU, utilizamos a IDE *Quartus II 13* junto com *ModelSim* para simular uma FPGA e fazer os testes necessários. Os testes em dispositivo físico foi feito no módulo de prototipação DE2.

2 ULA

A Figura 1 mostra o desenho esquemático de uma ULA.

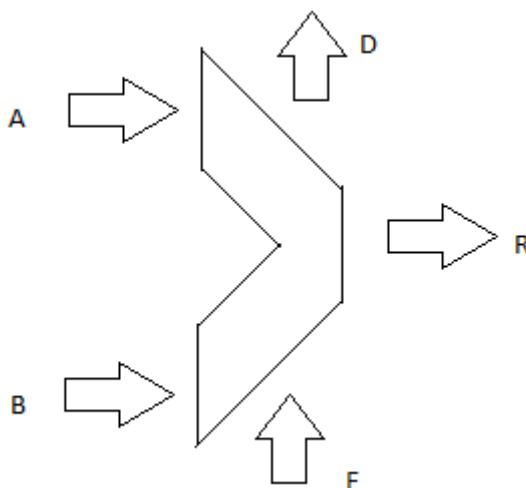


Figura 1: Desenho esquemático de uma ULA.

Nela, é possível ver que a ULA possui duas entradas e uma saída de dados, *A*, *B* e *R* respectivamente, além de uma entrada e uma saída de sinais de controle, *F* e *D*. As entradas *A* e *B* são os valores que a ULA recebe para fazer os cálculos necessários. Esses valores podem vir de registradores ou podem ser imediatos: números absolutos, como nas instruções a seguir:

1	Addi R1,R2,10
2	Sub R3,R4,R5

A instrução 1 soma o valor imediato 10 ao valor armazenado no registrador *R2* e armazena no registrador *R1*. No caso dessa instrução, a entrada *A* na Figura 1 recebe o valor do registrador *R2*, a entrada *B* o valor do imediato (10) e a saída *R* terá o valor da soma, sendo que este valor será armazenado no registrador *R1*. Já a instrução 2 subtrai o valor armazenado no registrador *R5* do valor armazenado em *R4* e armazena em *R3*, fazendo a operação

$$R3 = R4 - R5 \quad (1)$$

A entrada de sinais *F* é a entrada de controles da ALU. Dentre esses sinais de controle está o que sinaliza se o valor na entrada de dados *B* é o valor vindo de um registrador ou um imediato. Esse

seu sinal controla um multiplexador para selecionar qual entrada irá passar para a ALU, a entrada vinda do banco de registradores ou a entrada vinda direto da instrução no caso de um imediato.

Já a saída de sinais *D*, são os sinais que a ALU indica, sendo eles se o resultado é zero, negativo e se houve *overflow* na operação.

3 Implementação

Esta seção fala da implementação da ULA em *Verilog*. Nela, mostramos como cada parte da ULA foi implementada além de decisões de projeto tomadas.

O módulo desenvolvido possui 5 entradas e duas saídas: *OpA*, *OpB*, *Op*, *RST*, *CLK*, *Res* e *FlagReg* respectivamente. O Apêndice 3 mostra o código.

As entradas *OpA* e *OpB* representam as entradas *A* e *B* da Figura 1. A entrada *Op* indica qual operação a ALU vai fazer (Add, Sub, etc.) e faz parte da entrada *F* na Figura 1. As saídas *Res* e *FlagReg* indicam, respectivamente o resultado da operação e o sinal de saída da ALU, na Figura 1, as saídas *R* e *D* respectivamente.

Na implementação, as entradas *OpA* e *OpB* são de 16 bits, que, de acordo com a especificação do trabalho, é o tamanho dos registradores. Aqui, apenas o *OpB* pode ser um imediato. Neste caso, o imediato possui apenas 4 bits, sendo necessário estender mais 12 bits. Mais a frente será mostrado como e onde essa operação é feita. A entrada *Op* indica a operação a ser realizada e possui 4 bits, ou seja, a ALU implementada possui um máximo de 16 operações diferentes. Pela especificação, apenas 11 operações foram utilizadas como mostrado na Tabela 1.

Código	Operação	Descrição
0	Add	Adição com registradores
1	Sub	Subtração com registradores
2	Slti	Salto condicional com imediato
3	And	<i>And</i> binário
4	Or	<i>Or</i> binário
5	Xor	<i>Xor</i> binário
6	Andi	<i>And</i> binário com imediato
7	Ori	<i>Or</i> binário com imediato
8	Xori	<i>Xor</i> binário com imediato
9	Addi	Adição com imediato
10	Subi	Subtração com imediato

Tabela 1: Descrição das operações implementadas.

A saída *Res* possui 16 bits, assim como as entradas *OpA* e *OpB* já que é o resultado da operação e a saída *FlagReg* possui 3 bits, um para cada sinal de saída como mostra a Tabela 2.

Índice	Sinal
0	<i>Overflow</i>
1	Negativo
2	Zero

Tabela 2. Sinais de saída da ULA.

Apêndices

Módulo ULA

```

1 module ULA (OpA, OpB, Res, Op, FlagReg, CLK, RST);
2
3     input CLK, RST;
4     input [3:0] Op;
5     input [15:0] OpA, OpB;
6     output reg [15:0] Res;
7     output reg [2:0] FlagReg;           // [Z N C]: Z=Zero; N=Neg;
        C=Carry/Overflow
8
9     wire [15:0] invOpB;
10    assign invOpB = ~OpB + 16'd1;
11
12    parameter InsADD = 4'b0000;        // ADD  Res = OpA + OpB
13    parameter InsSUB = 4'b0001;        // SUB  Res = OpA - OpB
14    parameter InsSLT = 4'b0010;        // SLTI Res = (OpA > OpB) ? 1 : 0
15    parameter InsAND = 4'b0011;        // AND  Res = OpA & OpB
16    parameter InsOR  = 4'b0100;        // OR   Res = OpA | OpB
17    parameter InsXOR = 4'b0101;        // XOR  Res = OpA ^ OpB
18
19    parameter OverflowFlag = 0;
20    parameter NegFlag      = 1;
21    parameter ZeroFlag     = 2;
22
23    always @(posedge CLK) begin
24        if (RST) begin
25            FlagReg = 3'b000;
26        end
27        else begin
28            case (Op)
29                InsADD: begin
30                    Res = OpA + OpB;
31
32                    /* Zero check */
33                    if (Res == 0)
34                        FlagReg[ZeroFlag] = 1'b1;
35                    else
36                        FlagReg[ZeroFlag] = 1'b0;
37                    /* Overflow check */
38                    FlagReg[OverflowFlag] = (OpA[15] & OpB[15] & ~Res[15]) | (~OpA[
39                        15] & ~OpB[15] & Res[15]);
40                    /* Negative check */
41                    FlagReg[NegFlag] = Res[15];
42                end
43                InsSUB: begin
44                    Res = OpA + invOpB;
45                    /* Zero check */
46                    if (Res == 0)
47                        FlagReg[ZeroFlag] = 1'b1;
48                    else
49                        FlagReg[ZeroFlag] = 1'b0;
50                    /* Overflow check */
51                    FlagReg[OverflowFlag] = (OpA[15] & invOpB[15] & ~Res[15]) | (~
                    OpA[15] & ~invOpB[15] & Res[15]);
                    /* Negative check */

```

```

52     FlagReg[NegFlag] = Res[15];
53 end
54 InsSLT: begin
55     if (OpA > OpB)
56         Res = 16'd1;
57     else
58         Res = 16'd0;
59     /* Zero check */
60     if (Res == 0)
61         FlagReg[ZeroFlag] = 1'b1;
62     else
63         FlagReg[ZeroFlag] = 1'b0;
64     /* Overflow check */
65     FlagReg[OverflowFlag] = (OpA[15] & OpB[15] & ~Res[15]) | (~OpA[
66         15] & ~OpB[15] & Res[15]);
67     /* Negative check */
68     FlagReg[NegFlag] = Res[15];
69 end
70 InsAND: begin
71     Res = OpA & OpB;
72     /* Zero check */
73     if (Res == 0)
74         FlagReg[ZeroFlag] = 1'b1;
75     else
76         FlagReg[ZeroFlag] = 1'b0;
77     /* Overflow check */
78     FlagReg[OverflowFlag] = (OpA[15] & OpB[15] & ~Res[15]) | (~OpA[
79         15] & ~OpB[15] & Res[15]);
80     /* Negative check */
81     FlagReg[NegFlag] = Res[15];
82 end
83 InsOR: begin
84     Res = OpA | OpB;
85     /* Zero check */
86     if (Res == 0)
87         FlagReg[ZeroFlag] = 1'b1;
88     else
89         FlagReg[ZeroFlag] = 1'b0;
90     /* Overflow check */
91     FlagReg[OverflowFlag] = (OpA[15] & OpB[15] & ~Res[15]) | (~OpA[
92         15] & ~OpB[15] & Res[15]);
93     /* Negative check */
94     FlagReg[NegFlag] = Res[15];
95 end
96 InsXOR: begin
97     Res = OpA ^ OpB;
98     /* Zero check */
99     if (Res == 0)
100         FlagReg[ZeroFlag] = 1'b1;
101     else
102         FlagReg[ZeroFlag] = 1'b0;
103     /* Overflow check */
104     FlagReg[OverflowFlag] = (OpA[15] & OpB[15] & ~Res[15]) | (~OpA[
105         15] & ~OpB[15] & Res[15]);
106     /* Negative check */
107     FlagReg[NegFlag] = Res[15];
108 end
109 endcase

```

```
106         end
107     end
108 endmodule
```

Referências