Unidade Lógica e Aritmética (ULA)

Autores	
Gabriel A. F. Souza, Gustavo D. Colletta, Leonardo B. Zoccal, Odilon O. Dutra	Unifei

Histórico de Revisões		
21 de janeiro de 2025	1.0	Primeira versão do documento.

Tópicos

- Introdução
- Implementação em Verilog
- Exercícios

Introdução

O que é uma ULA?

A **Unidade Lógica e Aritmética (ULA)**, ou **ALU (Arithmetic Logic Unit)** em inglês, é um dos principais componentes de uma CPU (Unidade Central de Processamento). Ela é responsável por realizar as operações lógicas e aritméticas básicas necessárias para o funcionamento de programas e sistemas computacionais. A ULA é uma combinação de circuitos digitais que processa operações binárias em alta velocidade.

Funções da ULA

A ULA desempenha duas categorias principais de operações:

1. Operações Aritméticas
2. Operações Lógicas

1. Operações Aritméticas

Essas operações envolvem cálculos matemáticos, como:

- **Adição** $(+)$
- **Subtração** $(-)$
- **Multiplicação** (em algumas arquiteturas mais avançadas)
- **Divisão** (também disponível em arquiteturas avançadas)
- **Incremento** (adição de 1)
- **Decremento** (subtração de 1)

2. Operações Lógicas

Essas operações são baseadas na lógica booleana e incluem:

- **AND** ($\cdot$): Operação lógica "E".
- **OR** ($+$): Operação lógica "OU".
- **XOR**: Operação lógica "OU exclusivo".
- **NOT**: Negação lógica.
- **NAND**: Negação do "E".
- **NOR**: Negação do "OU".

Componentes Internos da ULA

1. **Somadores/Subtratores**: Circuitos como somadores completos (full adders) e complementos de 2 são usados para realizar operações de adição e subtração.
2. **Multiplexadores**: Selecionam a operação a ser executada com base nos sinais de controle.
3. **Circuitos Lógicos**: Implementam as operações lógicas (AND, OR, XOR, etc.).
4. **Sinais de Controle**: Determinam qual operação a ULA deve executar. Por exemplo, um código binário de controle pode indicar se a ULA deve realizar uma adição ou uma operação lógica AND.

Sinais da ULA

A ULA possui três tipos principais de sinais:

- **Entradas**:
  - Dois operandos (A e B), que são os números a serem processados.
  - Sinais de controle (C), que especificam a operação a ser realizada.
- **Saídas**:
  - O resultado da operação (R).
  - Flags ou indicadores de condição, como:
    - **Carry (C)**: Indica um transporte gerado ou emprestado em operações aritméticas.
    - **Zero (Z)**: Indica se o resultado da operação é zero.
    - **Overflow (V)**: Indica se ocorreu um estouro em operações aritméticas.
    - **Negative (N)**: Indica se o resultado é negativo.

Aplicações da ULA

- **Processamento de dados**: A ULA executa operações aritméticas e lógicas em dados recebidos de registradores ou da memória.
- **Tomada de decisões**: A ULA pode auxiliar na execução de instruções condicionais em programas, avaliando comparações entre operandos.
- **Componentes em arquiteturas avançadas**: Além da CPU, as GPUs (Unidades de Processamento Gráfico) também possuem ULAs dedicadas para operações em larga escala, especialmente para computação paralela.

Implementação em Verilog

Especificações da ULA

- Quantidade de bits dos operandos: 4
- Operações Lógicas: AND, OR, NOT, NAND
- Operações Aritméticas: Soma e subtração
- Quantidade de bits de seleção: 3
- Entradas de operação: 2 (A e B)
- Entradas de controle: 1 (seleção)
- Saídas: 1 (resultado da operação)

Especificações da ULA

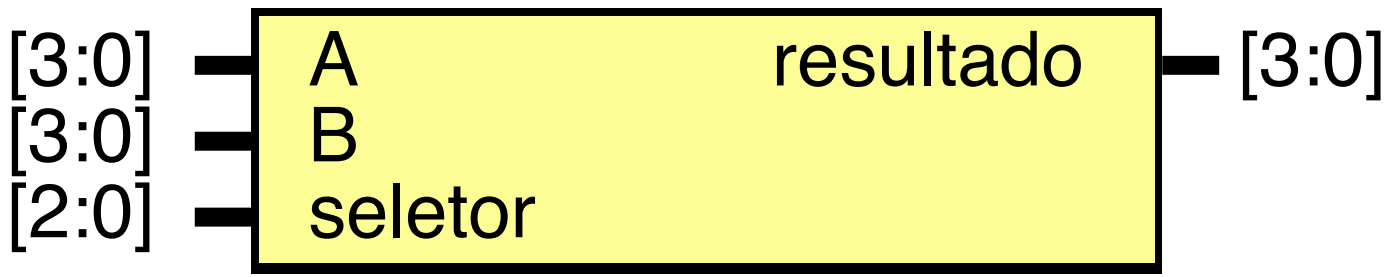

Figura 1: Símbolo da ULA especificada.

Descrição por fluxo de dados

```
1  module ula (
2      input  [3:0] A,              // Operando A
3      input  [3:0] B,              // Operando B
4      input  [2:0] seletor,        // Sinal de seleção (3 bits)
5      output [3:0] resultado       // Resultado da operação
6  );
7      // Operações
8      wire [3:0] op_and   = A & B;          // Operação AND
9      wire [3:0] op_or    = A | B;          // Operação OR
10     wire [3:0] op_not   = ~A;             // Operação NOT
           (aplicada ao operando A)
11     wire [3:0] op_nand  = ~(A & B);       // Operação NAND
12     wire [3:0] op_soma  = A + B;          // Soma
13     wire [3:0] op_sub   = A - B;          // Subtração
```

Descrição por fluxo de dados

```
14      // Multiplexação para selecionar a saída
15      assign resultado =
16          (seletor == 3'b000) ? op_and   :  // Operação AND
17          (seletor == 3'b001) ? op_or    :  // Operação OR
18          (seletor == 3'b010) ? op_not   :  // Operação NOT
19          (seletor == 3'b011) ? op_nand  :  // Operação NAND
20          (seletor == 3'b100) ? op_soma  :  // Operação Soma
21          (seletor == 3'b101) ? op_sub   :  // Operação Subtração
22                                4'b0000;    // Operação padrão
                                        (zero)
23  endmodule
```

Diagrama esquemático

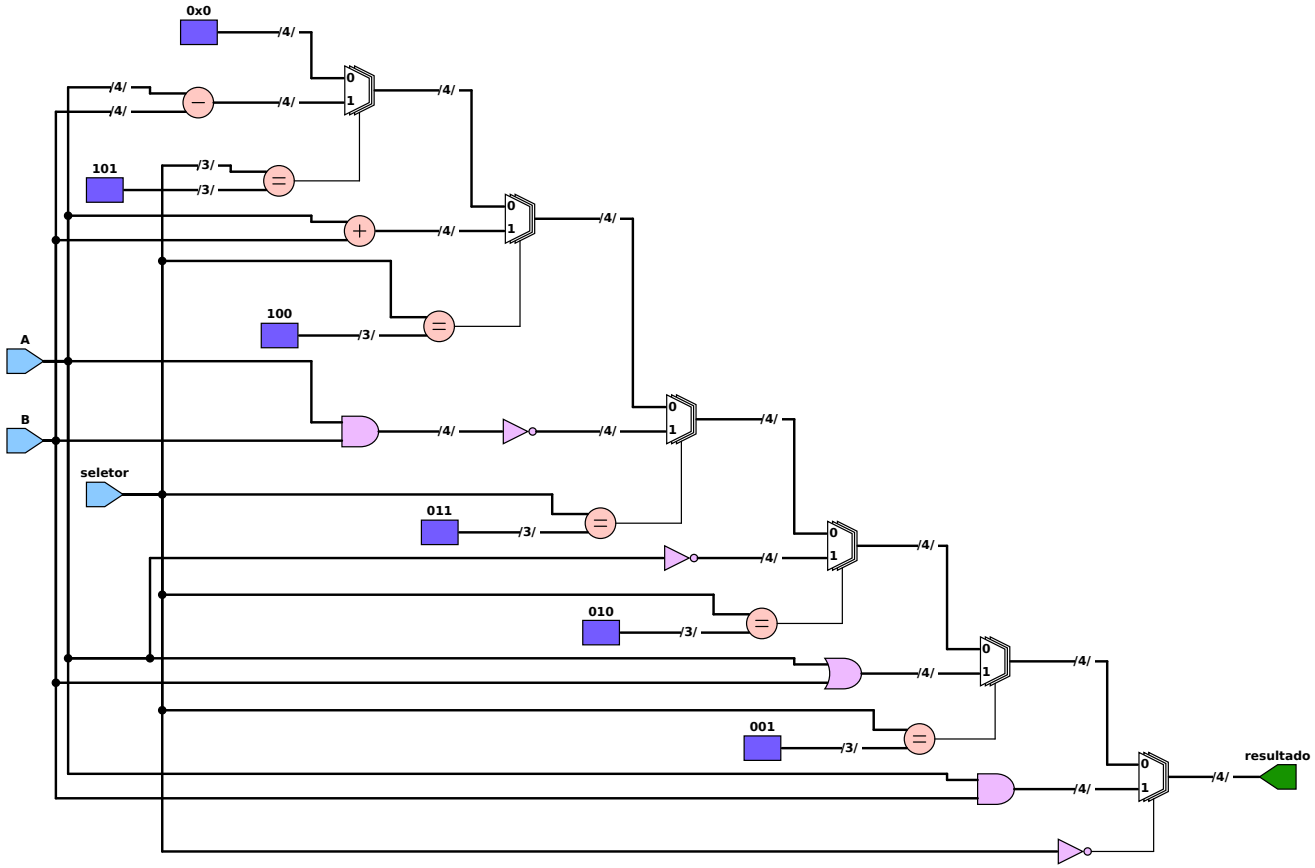

Figura 2: Esquemático da ULA por fluxo da dados.

Descrição estrutural

```
1  module ula (
2      input  [3:0] A,           // Operando A
3      input  [3:0] B,           // Operando B
4      input  [2:0] seletor,     // Controle da operação (3 bits)
5      output [3:0] resultado    // Resultado da operação
6  );
7      wire [3:0] soma_sub;      // Saída do módulo
           somador-subtrator
8      wire       carry_out;     // Transporte do somador-subtrator
9      wire [3:0] and_out;       // Saída da operação AND
10     wire [3:0] or_out;        // Saída da operação OR
11     wire [3:0] not_out;       // Saída da operação NOT
12     wire [3:0] nand_out;      // Saída da operação NAND
```

Descrição estrutural

```
13      // Instância do módulo somador-subtrator
14      somador_subtrator ss (
15          .A(A),
16          .B(B),
17          .op(seletor[0]),          // Seletor[0]: 0 para soma, 1
                para subtração
18          .resultado(soma_sub),
19          .carry_out(carry_out)
20      );
```

Descrição estrutural

```
21      // Operações lógicas com portas primitivas
22      and and0(and_out[0], A[0], B[0]);
23      and and1(and_out[1], A[1], B[1]);
24      and and2(and_out[2], A[2], B[2]);
25      and and3(and_out[3], A[3], B[3]);
26      or or0(or_out[0], A[0], B[0]);
27      or or1(or_out[1], A[1], B[1]);
28      or or2(or_out[2], A[2], B[2]);
29      or or3(or_out[3], A[3], B[3]);
30      not not0(not_out[0], A[0]);
31      not not1(not_out[1], A[1]);
32      not not2(not_out[2], A[2]);
33      not not3(not_out[3], A[3]);
34      nand nand0(nand_out[0], A[0], B[0]);
35      nand nand1(nand_out[1], A[1], B[1]);
36      nand nand2(nand_out[2], A[2], B[2]);
37      nand nand3(nand_out[3], A[3], B[3]);
```

Descrição estrutural

```
38      // Multiplexação para selecionar a operação com base no
            seletor
39      assign resultado =
40          (seletor == 3'b000) ? and_out   :  // Operação AND
41          (seletor == 3'b001) ? or_out    :  // Operação OR
42          (seletor == 3'b010) ? not_out   :  // Operação NOT
43          (seletor == 3'b011) ? nand_out  :  // Operação NAND
44          (seletor == 3'b100) ? soma_sub  :  // Soma
45          (seletor == 3'b101) ? soma_sub  :  // Subtração
46                                4'b0000;     // Operação padrão
                                       (zero)
47  endmodule
```

Diagrama esquemático

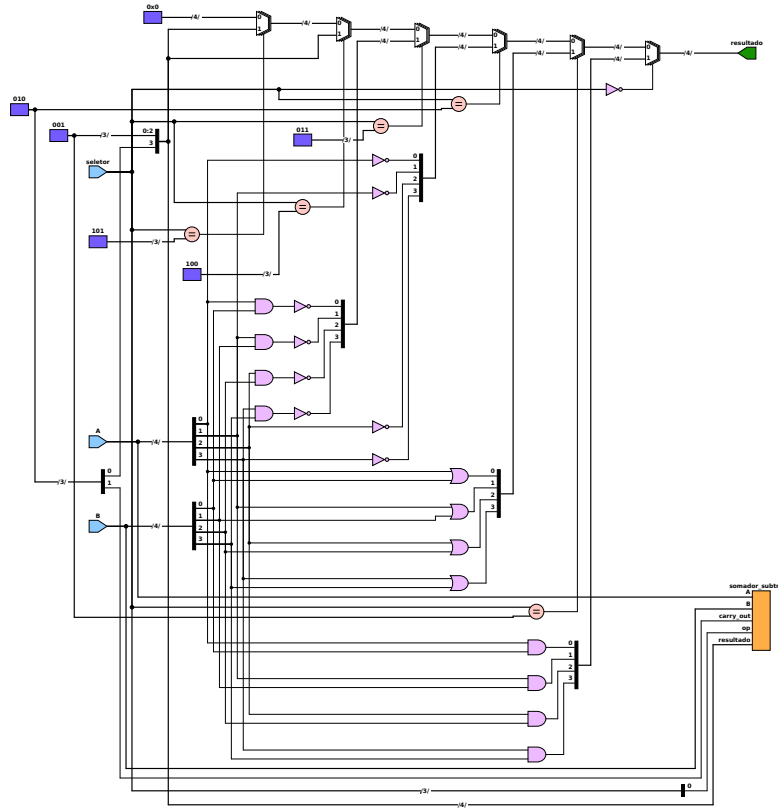

Figura 3: Esquemático da ULA estrutural.

Descrição comportamental

```
1  module ula (
2      input  [3:0] A,          // Operando A
3      input  [3:0] B,          // Operando B
4      input  [2:0] seletor,    // Sinal de seleção (3 bits)
5      output reg [3:0] resultado // Resultado da operação
6  );
```

Descrição comportamental

```
7       always @(*) begin
8           case (seletor)
9               3'b000: resultado = A & B;          // Operação AND
10              3'b001: resultado = A | B;          // Operação OR
11              3'b010: resultado = ~A;             // Operação NOT
                    (aplica-se apenas ao operando A)
12              3'b011: resultado = ~(A & B);       // Operação NAND
13              3'b100: resultado = A + B;          // Soma
14              3'b101: resultado = A - B;          // Subtração
15              default: resultado = 4'b0000;       // Operação padrão
                    (zero)
16          endcase
17      end
18  endmodule
```

Diagrama esquemático

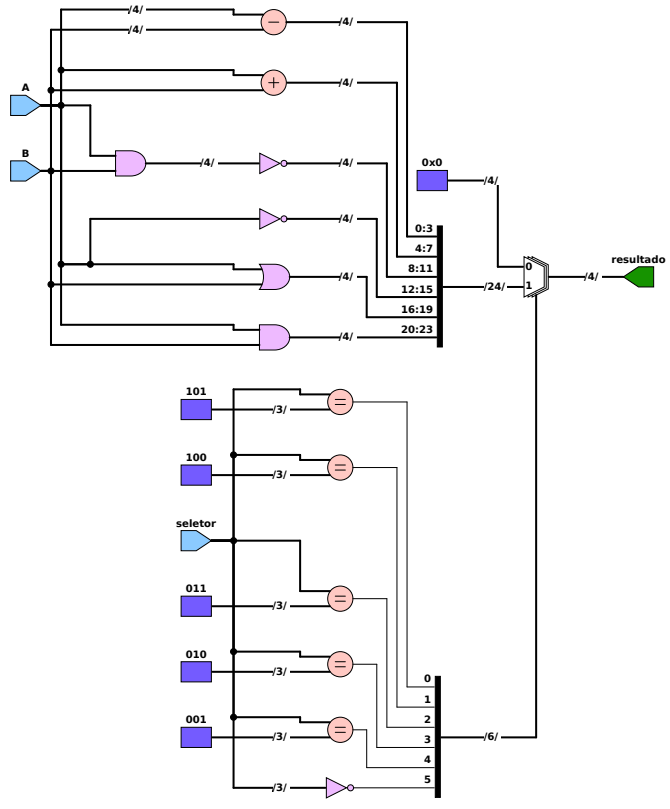

Figura 4: Esquemático da ULA comportamental.

Exercícios

Exercício 1

Crie um arquivo de *testbench* capaz de testar a ULA fornecida.

1. Instancie apenas a descrição comportamental da ULA.
2. Teste as 6 operações que a ULA fornece.

Exercício 2

1. Modifique a ULA fornecida para incluir 2 novas operações: deslocamento lógico para a esquerda (LSL) e deslocamento lógico para a direita (LSR). Nos deslocamentos lógicos, as posições vacantes devem ser preenchidas com valor lógico 0. Dessa forma, na operação LSL o bit mais significativo do operando é descartado enquanto o bit menos significativo é preenchido com 0. Já na operação LSR, o bit menos significativo é descartado enquanto o bit mais significativo é preenchido com 0. Nessa versão, o deslocamento deve ser de apenas 1 posição sobre o operando da entrada A.
2. Modifique também o arquivo de *testbench* do exercício 1 de forma a contemplar o teste das novas operações adicionadas.

Exercício 3

1. Modifique a ULA do exercício 2 para que as operações LSL e LSR utilizem o valor presente no operando B como o número de deslocamentos a ser executado sobre o operando A. Como temos operandos de 4 bits, o valor máximo de deslocamentos deve ser restrito a 4.
2. Modifique o arquivo de *testbench* para testar a modificação realizada.

Exercício 4

1. Modifique a ULA do exercício 3 para incluir 4 novas saídas: C, para indicar estouro de representação não sinalizada (transporte na soma e empréstimo na subtração), V, para indicar estouro de representação sinalizada, Z, para indicar resultado nulo, e N, para indicar resultado negativo. No caso da soma, $C = 1$ indica o transporte. Já na subtração, $C = 0$ indica o empréstimo.
2. Modifique também o arquivo de *testbench* para testar as saídas acrescentadas.

Exercício 5

1. Modifique a ULA do exercício 4 para incluir as operações lógicas NOR e XOR.
2. Modifique o arquivo de *testbench* para testar as novas operações adicionadas.

É importante notar a necessidade de aumentar a quantidade de bits da palavra de seleção para acomodar as operações adicionadas.