

# Laboratório 7 – Unidades Lógico-Aritméticas

## Objetivos:

1. Experimentar a descrição em VHDL de circuitos digitais;
2. Reforçar os conceitos envolvendo o projeto de unidades lógico-aritméticas.
3. Pôr em prática conceitos aprendidos na disciplina teórica.

## Introdução:

Nesta aula iremos implementar uma calculadora multifuncional, utilizando uma unidade lógico-aritmética que realiza operações aritméticas de soma e subtração, além de outras operações lógicas.

## Unidade lógico-aritmética:

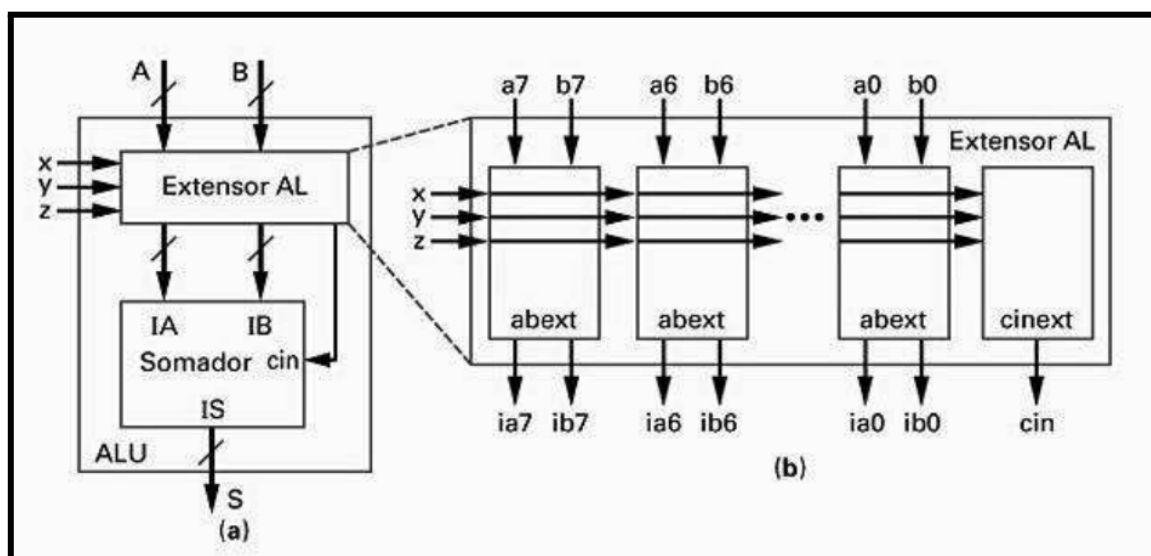
Uma **unidade de aritmética e lógica** - (ALU, do inglês *arithmetic-logic unit*) é um componente de bloco operacional capaz de executar diversas operações aritméticas e lógicas com duas entradas de dados, com  $N$  bits de largura, gerando uma saída de dados de  $N$  bits. A adição e subtração são exemplos de operações aritméticas. Alguns exemplos de operações lógicas incluem AND, OR, XOR, etc. As entradas de controle da ALU indicam qual operação em particular deve ser realizada.

A **Tabela 1** mostra um exemplo de como pode-se atribuir operações lógico-aritméticas em duas entradas A e B, de 8 bits, com a finalidade de obter-se uma saída S, também de 8 bits. Este exemplo pode ser utilizado para a implementação de uma calculadora multifuncional de 8 bits. A tabela inclui diversas operações bit a bit (AND, OR, XOR e complemento). Uma operação bit a bit aplica-se separadamente a cada par de bits correspondentes de A e B.

O projeto desta calculadora multifuncional pode ser feito com o uso de uma ALU, a qual, por sua vez, é projetada a partir de um componente básico, composto por um bloco **somador** e um bloco **extensor** lógico-aritmético (AL). A **Figura 1** ilustra uma ALU projetada a partir de blocos extensor AL e somador.

**Tabela 1** - Operações desejadas de uma calculadora multifuncional.

Entradas			Operação	Exemplo de saída se A=00001111, B=00000101
x	y	z		
0	0	0	$S = A + B$	S=00010100
0	0	1	$S = A - B$	S=00001010
0	1	0	$S = A + 1$	S=00010000
0	1	1	$S = A$	S=00001111
1	0	0	$S = A \text{ AND } B$ (AND bit a bit)	S=00000101
1	0	1	$S = A \text{ OR } B$ (OR bit a bit)	S=00001111
1	1	0	$S = A \text{ XOR } B$ (XOR bit a bit)	S=00001010
1	1	1	$S = \text{NOT } A$ (complemento bit a bit)	S=11110000



**Figura 1** - Unidade de aritmética e lógica: (a) projeto de ALU baseado em um somador simples, com um extensor de aritmética e lógica e (b) detalhe do extensor de aritmética e lógica.

Para projetar a ALU, conforme ilustrado na **Figura 1**, vamos chamar as entradas do somador interno de IA e IB (A "interno" e B "interno"), para distingui-lás das entradas externas A e B da ALU. O objetivo do extensor AL é determinar os valores das entradas do

somador com base nos valores das entradas de controle  $x$ ,  $y$  e  $z$  da ALU, de modo que o resultado aritmético ou lógico desejado apareça na saída do somador. Na realidade, o extensor AL consiste em oito componentes idênticos, chamados de *abext*, um para cada par de bits  $a_i$  e  $b_i$ , como mostrado na **Figura 1**. O extensor AL também possui um componente *cinext*, para computar o bit cin.

Desse modo, precisamos projetar os componentes *abext* e *cinext* para completar o projeto da ALU. Considere as quatro primeiras operações da **Tabela 1**, que são aritméticas:

- Quando  $xyz = 000$ ,  $S = A+B$ . Assim, neste caso, queremos  $IA = A$ ,  $IB = B$  e  $cin = 0$ .
- Quando  $xyz = 001$ ,  $S = A-B$ . Assim, queremos  $IA = A$ ,  $IB = B'$  e  $cin = 1$ .
- Quando  $xyz = 010$ ,  $S = A+1$ . Assim, queremos  $IA = A$ ,  $IB = 0$  e  $cin = 1$ .
- Quando  $xyz = 011$ ,  $S = A$ . Assim, queremos  $IA = A$ ,  $IB = 0$  e  $cin = 0$ . Observe que  $A$  passará através do somador pois  $A+0+0 = A$ .

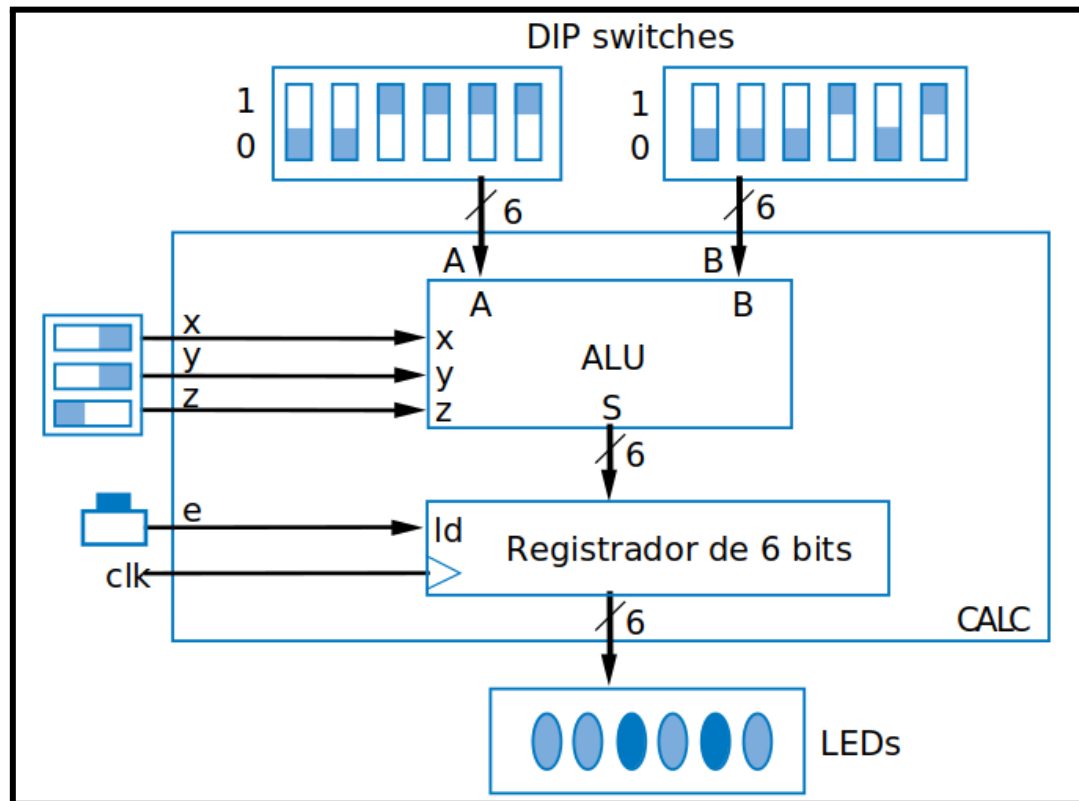
As quatro últimas operações da ALU são todas operações lógicas. Podemos computar a operação desejada no componente *abext* e entrar com esse resultado em  $IA$ . A seguir, fazemos  $IB$  ser 0 e  $cin$  ser 0, de modo que o valor de  $IA$  passará inalterado através do somador.

Uma possível estrutura para os componentes *abext* e *cinext* é feita a partir do projeto de um circuito combinacional customizado para cada uma das saídas dos componentes.

## Projeto de uma calculadora multifuncional de 6 bits:

Vamos projetar uma calculadora multifuncional que possa realizar oito operações, determinadas por uma chave DIP tripla, que fornece três entradas  $x$ ,  $y$  e  $z$  para o nosso sistema, conforme descrição da **Tabela 1**. Para cada combinação das três chaves, queremos executar as operações mostradas na Tabela 1 com as entradas de dados  $A$  e  $B$ , produzindo uma saída  $S$ . Porém, nossa calculadora possuirá entradas e saídas de apenas **6 bits** (ao contrário da calculadora da Tabela 1, cujas entradas e saídas possuem 8 bits).

De forma análoga ao laboratório 5, os números binários de entrada virão de chaves DIP de seis botões (DIP *switches*) e a saída será mostrada através de seis LEDs, como mostrado na **Figura 2**. Uma chave DIP (*Dual Inline Package*) de seis bits é um componente digital simples com botões ou chaves que um usuário pode mover para cima ou para baixo. Em cima, é gerado 1 no bit correspondente e, em baixo, é gerado 0. Um LED (*light-emitting diode*) é apenas uma pequena lâmpada que acende, quando a entrada do LED é 1, e apaga, quando a entrada é 0. O botão *igual a* significa “igual a” e indica quando o novo resultado deve ser exibido. Apertaremos o botão e somente após ter configurado as duas chaves DIP com os novos valores de  $A$  e  $B$  que devem ser levados em consideração para o cômputo da saída  $S$ . A entrada *id* será conectada no pino *Id* de um registrador com carga em paralelo, para evitar que os LEDs fiquem piscando até estabilizar o resultado. Observe que o valor exibido será uma representação binária de 6 bits, com complemento de dois.



**Figura 2** - Calculadora multifuncional de 6 bits.

## Atividades:

1. Implemente a calculadora da Figura 2 em VHDL e simule seu comportamento;
2. Sintetize o código VHDL no kit FPGA disponível e apresente o resultado em sala de aula;
3. Entregue um relatório descrevendo a execução dos itens 1 e 2.

### Observações:

- A implementação dos blocos *abext* e *cinext* que compõem o extensor AL da ALU deve ser feita por meio de circuitos combinacionais projetados a partir da análise de entradas e saídas destes blocos. **Não será aceita** implementação do extensor AL por meio de multiplexadores.
- O relatório deve conter as tabelas verdade e equações booleanas dos circuitos combinacionais que compõem os blocos *abext* e *cinext*.