Anais Eletrônicos

Unidade Lógica Aritmética

Alexandre Moraes Tannus

2019

UniEVANGÉLICA
CENTRO UNIVERSITÁRIO

Centro Universitario de Anápolis - UniEVANGÉLICA

Centro Universitário de Anápolis – UniEVANGÉLICA
Av. Universitária Km. 3,5 – Cidade Universitária - Anápolis-GO CEP:75083-51
Cursos Superiores de Computação

SUMÁRIO

APRESENTAÇÃO

A unidade lógica aritmética é o local onde todos os cálculos são realizados em um sistema computacional. Conhecer sobre o funcionamento deste elemento é essencial para que profissionais de computação não incorram em erros de representação que podem gerar confusões em um sistema.

Números inteiros e reais possuem diferentes formas de representação. Estas formas devem ser compatíveis com a ULA para que os cálculos não sejam realizados de forma incorreta e/ou para que não haja ambiguidades.

OBJETIVOS

Geral

Entender o funcionamento da unidade lógica aritmética (ULA) de um processador

Específicos

- Compreender a função da unidade lógica aritmética
- Apresentar as operações lógicas e aritméticas realizadas pela ULA
- Calcular a representação binária de números negativos

INTRODUÇÃO

Em qualquer sistema computacional (desde uma simples calculadora até o mais complexo supercomputador) os usuários esperam que sejam realizados diversos tipos de cálculos. A responsabilidade por esta tarefa é de um elemento chamado Unidade Lógica Aritmética (ULA). Esta unidade tem como principal função, como já expresso em seu nome, realizar cálculos de funções lógicas e aritméticas.

Para que isso seja realizado da forma correta é essencial que as formas de representação dos valores envolvidos sejam definidas de forma correta. Além disso, circuitos digitais devem ser implementados para realizar cada tipo de operação desejada para o bom funcionamento desta unidade.

Este material aborda as formas de representação dos números inteiros, as operações lógicas e as operações aritméticas que são realizadas em uma ULA

DESENVOLVIMENTO

A unidade lógica aritmética é o componente da Unidade Central de Processamento (CPU – do inglês *Central Processing Unit*) responsável, como o próprio nome sugere, pela realização de operações lógicas e aritméticas. Os outros elementos do sistema computacional (memória, dispositivos de entrada/saída, registradores) levam dados para que a ULA realize as devidas operações e buscam o resultado após a efetuação dos cálculos (Figura 1)

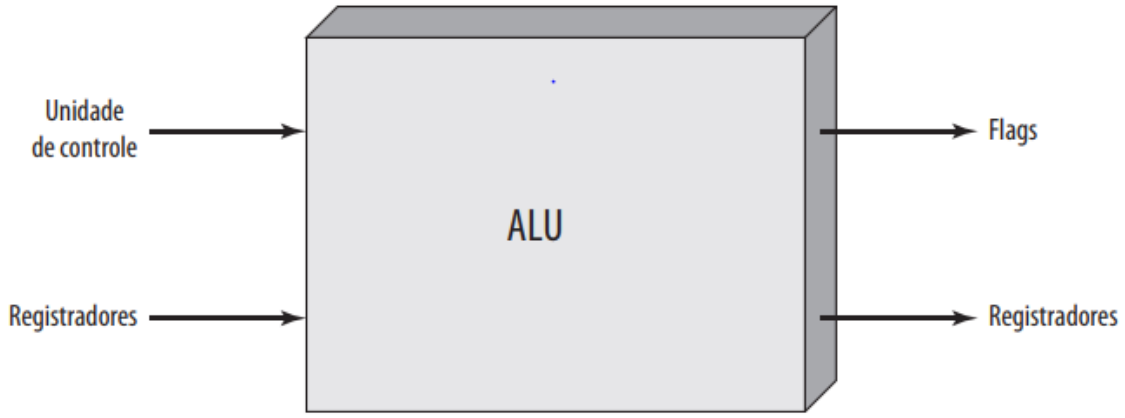

Figura 1 - Entradas e Saídas da ULA (STALLINGS, 2010)

Operações lógicas

Dentre as operações lógicas são notáveis as operações de complemento (*not*), adição (*or*) e multiplicação (*and*). Cada uma destas operações possui um comportamento descrito por uma tabela verdade.

Complemento (not)

A operação de complemento realiza a inversão do valor do bit. Sendo assim, caso um bit igual a 1 seja apresentado ao operador de complemento o resultado será um bit igual a 0, e vice-versa (Tabela 1).

Tabela 1 - Operação Lógica de Complemento

Entrada (A)	Saída ($not\ A$)
0	1
1	0

Esta operação possui diversas formas de representação em forma de expressão, sendo as mais comuns: $not\ A$, $\bar{A}$, A' e $\neg A$. Em linguagens de programação a forma de representação mais comum é $!A$, apesar de algumas linguagens utilizarem a forma $\sim A$.

Adição (or)

A operação de adição lógica, também conhecida como *ou lógico* (*or*), é uma operação que pode envolver dois ou mais *bits*. O resultado da operação é definido pela quantidade de valores iguais a 1 presentes nas entradas. Caso haja pelo menos uma entrada igual a 1 o resultado será igual a 1. O resultado só será igual a 0 caso todas as entradas sejam iguais a 0. A Tabela 2 mostra o comportamento da função *or.*

Tabela 2 - Operação Lógica de Adição

A	B	A *or* B
0	**0**	0
0	**1**	1
1	**0**	1
1	**1**	1

A operação *or* é representada utilizando o símbolo matemático da adição (+). Sendo assim, a operação lógica $A\ or\ B$ pode ser representada na forma $A + B$. Para não haver confusão de operadores na programação, esta operação é realizada utilizando o símbolo || em quase todas as linguagens de programação. Este operador realiza o cálculo utilizando todos os valores de bits do operando. Na prática, todo valor diferente de 0 (para valores numéricos) é considerado como sendo igual a 1. No caso de cadeias de caracteres (*strings*) deve ser verificado como cada linguagem trata a *string* vazia ($''$) e a *string* nula ($null$).

Multiplicação (and)

A operação de multiplicação lógica, também conhecida como *e lógico* (*and*), assim como a operação de adição, pode envolver dois ou mais bits. O resultado desta operação será igual a 1 se todas as entradas forem iguais a 1. Caso haja pelo menos um valor igual a 0 o resultado será 0.

Tabela 3 - Operação Lógica de Adição

A	B	A *and* B
0	**0**	0
0	**1**	0
1	**0**	0
1	**1**	1

A representação da operação *and* é feita através do símbolo matemático da multiplicação (.), podendo ser omitido caso não cause ambiguidades. Desta forma, a operação $A\ and\ B$ pode ser representada como $A.B$ ou AB. Na programação, o operador utilizado é o $\&\&$. Na função *and* o comportamento com valores numéricos e com cadeias de caracteres é semelhante ao descrito para a função *or*.

Detector de diferenças (XOR)

A função lógica *ou exclusivo*, também conhecida como XOR (*Exclusive OR)*, é utilizada como um detector de diferença entre dois bits. Caso os bits sejam diferentes o resultado da operação é igual a 1. Caso haja igualdade entre os operadores o resultado será igual a 0. A Tabela 4 mostra o comportamento da função XOR

Tabela 4 - Operador Lógico XOR

A	B	A *xor* B
0	**0**	0
0	**1**	1
1	**0**	1
1	**1**	0

A representação simbólica desta operação é feita com o símbolo $\oplus$. Sendo assim, $A\ xor\ B$ é representado como $A \oplus B$. Outra forma de representar esta operação é através da fórmula $(not\ A).B + A.(not\ B)$. Em programação, algumas linguagens, tais como C, utilizam o operador ^^ para executar a operação XOR.

Operações aritméticas

As ULAs realizam operações aritméticas simples como adição e subtração. É possível que implementem também circuitos para a realização de cálculos de multiplicação, divisão, potenciação e radiciação, mas normalmente isso é feito através de múltiplas instruções de adição e subtração.

A unidade lógica aritmética, como elemento fundamental do processador, trabalha com instruções de baixo nível, tratando as operações no nível de bits. Sendo assim, é necessário que sejam realizadas conversões para o sistema binário para que a ULA trabalhe. Tais conversões são explicadas em (TANNUS, 2019).

Adição aritmética binária

Os números binários podem ser somados seguindo a convenção exposta na Tabela 5. Nesta tabela, os valores de entrada são representados por A e B. O resultado da operação é mostrado em S. A variável de saída C (*carry*) representa o conceito de 'vai um', também utilizado na adição decimal.

Tabela 5 - Soma binária

A	B	S	C
0	**0**	0	0
0	**1**	1	0
1	**0**	1	0
1	**1**	0	1

Em nível de hardware, a Tabela 5 é representada pelo circuito conhecido como meio somador (*half-adder*). A Figura 2 mostra um diagrama de blocos do meio somador

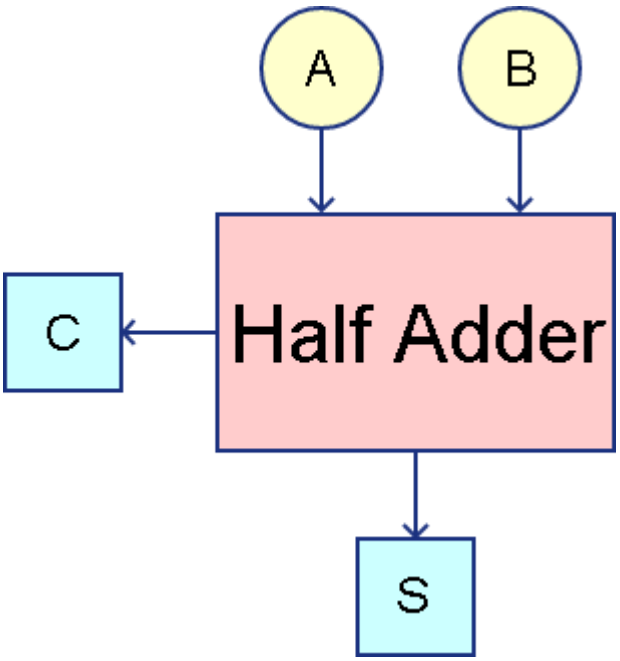

Figura 2 - Half Adder - Diagrama de blocos

Apesar de sua utilidade para cálculos que envolvem apenas dois bits, o meio somador possui limitações quando é necessário somar cadeias de bits, devido ao fato de ele não considerar o elemento de *carry* (C) no cálculo. Para solucionar essa deficiência o circuito somador completo (Figura 3) é utilizado. A Tabela 6 mostra o funcionamento do somador completo. Nesta tabela, a variável C do meio somador foi renomeada para C_{OUT} e continua representando o conceito de 'vai um'. O 'um' que é obtido através do C_{OUT} é utilizado como C_{IN} no bit imediatamente à esquerda.

Tabela 6 - Somador Completo

A	B	C_{IN}	S	C_{OUT}
0	**0**	**0**	0	0
0	**0**	**1**	1	0
0	**1**	**0**	1	0
0	**1**	**1**	0	1
1	**0**	**0**	1	0
1	**0**	**1**	0	1
1	**1**	**0**	0	1
1	**1**	**1**	1	1

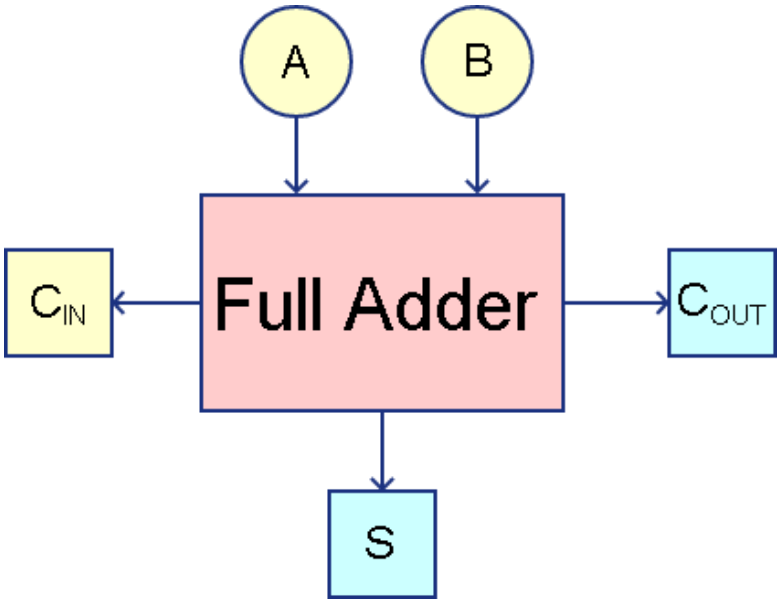

Figura 3 - Somador Completo - Diagrama de Blocos

Exemplo: Calcular $(1001)_2 + (0101)_2$

C		0	0	1
A	1	0	0	1
B	0	1	0	1
S	1	1	1	0

Representação Númerica

Os números decimais podem ser representados em forma de cadeias de bits. A quantidade de bits da cadeia delimita a faixa de valores decimais que pode ser representada. Esta faixa varia de 0 a $2^n - 1$, para números naturais (também chamados de números não sinalizados), sendo n o número de bits da cadeia. Caso os números sejam inteiros (números sinalizados) esta faixa é dividida simetricamente para números positivos e negativos, sendo o zero considerado no grupo dos números positivos. Sendo assim, a faixa varia de -2^{n-1} a $2^{n-1} - 1$

Exemplo: Para uma cadeia de 8 bits qual é a faixa de representação para números sinalizados e não sinalizados.

- *Não sinalizado:* Sabendo que $2^8 - 1 = 255$, temos que a faixa de valores varia de 0 a 255

- *Sinalizado:* Considerando que $2^{8-1} = 2^7 = 128$ e $2^{8-1} - 1 = 2^7 - 1 = 127$, temos que a faixa varia de -128 a 127

Representação de números inteiros

Os números inteiros podem ser representados em dois formatos

- Sinal-magnitude
- Complemento de 2

O formato de sinal-magnitude define um bit, o mais significativo da cadeia (MSB – *most significant bit*) como indicador do sinal do número, sendo o bit igual a 1 para representar números negativos e o bit 0 para números positivos

Exemplo: Qual é o valor de $(\mathbf{10010})_\mathbf{2}$ no sistema decimal

$$\left(\underbrace{\mathbf{1}}_{\boldsymbol{sinal}} \quad \underbrace{\mathbf{0010}}_{\boldsymbol{magnitude}} \right)_\mathbf{2}$$

O valor 1 de sinal representa um número negativo. Já o valor 0010 representa o valor 2 em decimal. Então, podemos concluir que $(\mathbf{10010})_\mathbf{2}$ é igual a -2

O principal problema deste sistema é com relação à representação do número 0, que seria corretamente representado com sinal negativo $(10000)_2$ ou com sinal positivo $(00000)_2$. Esta ambiguidade não é interessante para o trabalho da ULA e outro método foi desenvolvido para superar este problema.

O método mais utilizado atualmente para representação de números negativos é o complemento de 2. Este método envolve quatro etapas

- **Etapa 1:** Definição da quantidade de bits mínima necessária para representar o número negativo
- **Etapa 2:** Obtenção do valor binário do número oposto (o oposto de -3 é $+3$, por exemplo)
- **Etapa 3:** Inversão dos valores de todos os bits (complemento de 1)
- **Etapa 4:** Adição de 1 ao valor (complemento de 2)

Exemplo: Encontrar a representação binária do número -5

- **Etapa 1:** O número $(-5)_{10}$ encontra-se na faixa entre -8 e $+7$. Sabemos que $-8 = -2\text{^}3$ e que o cálculo da faixa representação de números sinalizados é feito através da fórmula -2^{n-1}. Temos então que a quantidade de bits é

$$-2^{n-1} = -2^3$$

$$n - 1 = 3$$

$$n = 4$$

  Então a quantidade de bits mínima necessária para representar $(-5)_{10}$ é de 4 bits
- **Etapa 2:** Representando o número $(= 5)_{10}$ utilizando 4 bits obtemos $(+\mathbf{5})_{\mathbf{10}} = (\mathbf{0101})_{\mathbf{2}}$
- **Etapa 3:** O complemento de $(0101)_2$ é $(1010)_2$
- **Etapa 4:** $(1010)_2 + (1)_2 = (1011)_2$

Conclui-se então que a representação binária de $(-5)_{10}$ é $(1011)_2$

FICA A DICA!

Ponto flutuante

A representação mais comum de números em ponto flutuante é a mantissa-expoente. Esta notação divide o número em 3 conjuntos de bits, sendo que o bit mais significativo representa o sinal do número (sinal-magnitude), seguido por um conjunto de bits que representa o expoente e outro conjunto representando a mantissa. A norma IEEE 754 (IEEE, 2008) explica como proceder para obter um número através desta notação e também as formas de cálculo com ponto flutuante.

CONCLUSÃO

Os objetivos deste material são o entendimento do funcionamento de uma ULA, abordando as formas de representação numéricas e os cálculos com números inteiros.

O entendimento destes tópicos acrescenta uma valiosa informação sobre como funcionam os sistemas computacionais em sua função mais essencial: a de calcular respostas a partir de valores de entrada.

REFERÊNCIAS

IDOETA, Ivan; CAPUANO, Francisco. Elementos de Eletrônica Digital. 41. ed. São Paulo: Érica, 2012. .

IEEE. IEEE Standard for Floating-Point Arithmetic . [S.l.]: IEEE Computer Society. , 2008

STALLINGS, William. Arquitetura e Organização de Computadores. 8. ed. São Paulo: Pearson, 2010. .9788576055648.

TANENBAUM, Andrew S. Organização estruturada de computadores. 2007.

TANNUS, Alexandre Moraes. Sistemas de Numeração . Anápolis: UniEvangélica. , 2019 [não publicado]