

N_HARD COMPA2 - Texto de apoio

Site: [EAD Mackenzie](#)
Tema: HARDWARE PARA COMPUTAÇÃO {TURMA 01B} 2021/2
Livro: N_HARD COMPA2 - Texto de apoio

Impresso por: ANDRE SOUZA OCLECIANO .
Data: terça, 28 set 2021, 21:34

Descrição

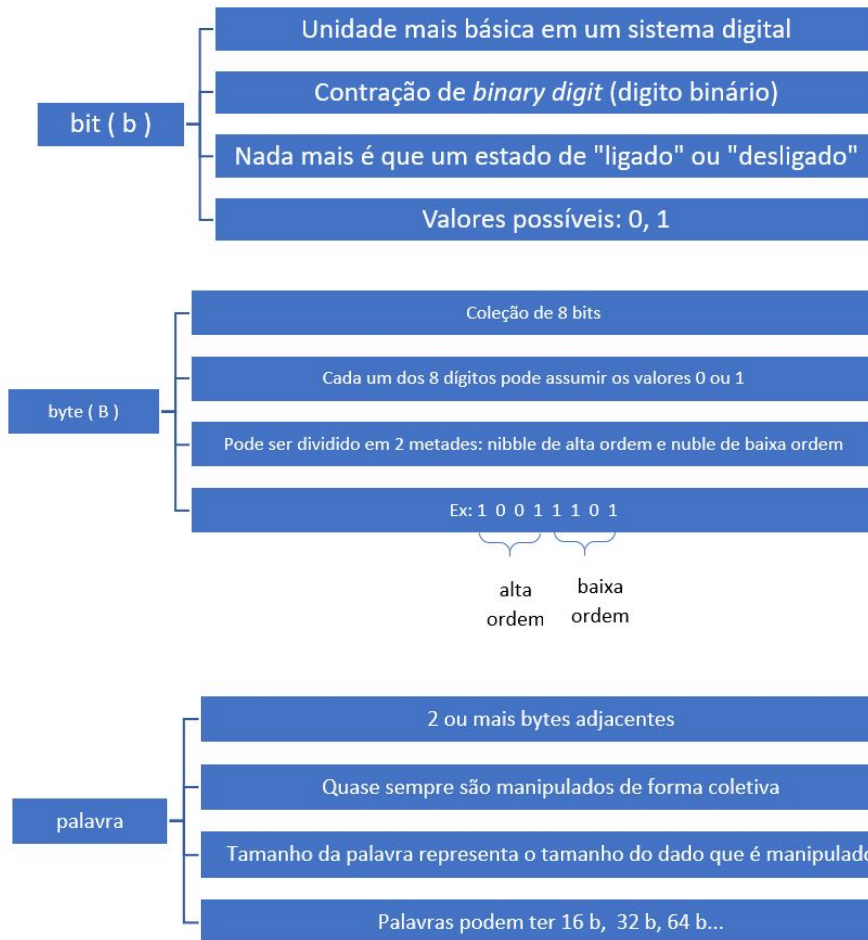
Índice

1. SISTEMAS DE NUMERAÇÃO e REPRESENTAÇÃO DE DADOS

1. SISTEMAS DE NUMERAÇÃO e REPRESENTAÇÃO DE DADOS

A organização de um computador depende de como ele representa números, caracteres e informações de controle.

Unidades básicas de representação de dados em um sistema de computação são:



BASES NUMÉRICAS

Uma base numérica, de uma forma simplificada, é um conjunto de símbolos e/ou algarismos utilizado para representar quantidades, números ou dados.

As bases importantes para um sistema computacional são:

- Base binária
 - Base 2
 - Dígitos: 0, 1
- Base octal
 - Base 8
 - Dígitos: 0, 1, 2, 3, 4, 5, 6, 7
- Base decimal
 - Base 10
 - Dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Base hexadecimal

▫ Base 16

▫ Dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

A tabela a seguir apresenta os números de 0_{10} a 15_{10} representados em diferentes bases numéricas. Note que uma mesma sequência de algarismos pode representar valores diferentes em diferentes bases numéricas.

Decimal	Binário	Octal	Hexadecimal
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

CONVERSÕES DE BASES

Qualquer base para decimal

Utilizar o sistema de numeração posicional.

- § Cada dígito em um número tem um valor diferente, dependendo da posição que ocupa.

Por exemplo: 123 – dígito 1 equivale a 100

321 – dígito 1 equivale a 1

- § Um valor numérico é representado por potências crescentes de uma raiz (ou base).
- § Cada posição é ponderada por uma potência de uma base.

Por exemplo: 243_{10} – um número representado na base decimal (base que usamos no nosso dia a dia).

Então, como representar esse mesmo número usando a base 10? É possível? **SIM**

$$243_{10} = 200 + 40 + 3$$

$$2 * 100 + 4 * 10 + 3$$

$2 * 10^2 + 4 * 10^1 + 3 * 10^0$ à conseguimos representar o 243_{10} em potências crescentes da base 10

Essa representação vale para qualquer base. E como fazer essa representação para outras bases de uma forma mais prática?

1º passo – numerar as posições do número, da direita para esquerda, iniciando em zero.

$$212_3 = 2 \ 1 \ 2_3$$

2 1 0 -- posições

2º passo – para cada dígito, faça: dígito * base^{posição}

$$212_3 = 2 \cdot 1 \cdot 2_3$$

2 1 0 -- posições

$$= 2 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0$$

E se somarmos o resultado desta representação em sistema de numeração posicional?

$$212_3 = 2 \cdot 1 \cdot 2_3$$

2 1 0 -- posições

$$= 2 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0$$

$$= 2 \cdot 9 + 1 \cdot 3 + 2 \cdot 1$$

$$= 18 + 3 + 2$$

$$= 23 \text{ ??}$$

E o que o 23 representa? $212_3 = 23_{10}$

Então a representação de um número, em qualquer base numérica, por meio do sistema de numeração posicional, gera o número correspondente na base decimal.

Por exemplo: $10110_2 = ?_{10}$

$$10110_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

$$= 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$$

$$= 16 + 0 + 4 + 2 + 0$$

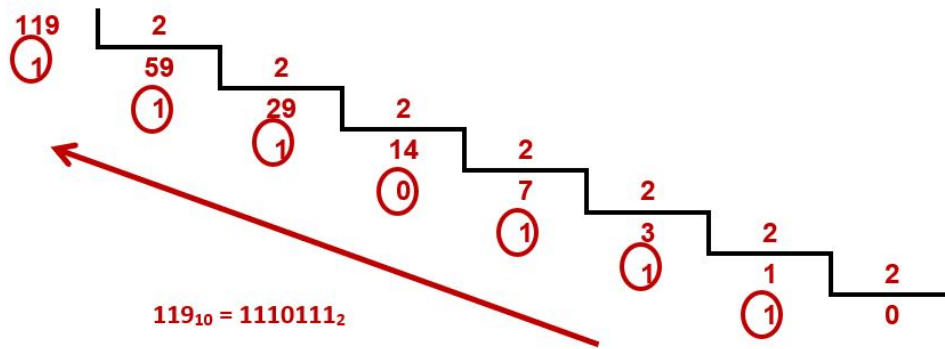
$$= 22_{10}$$

Portanto, $10110_2 = 22_{10}$

Decimal para binário

Divisões inteiras sucessivas do número de origem pela base de destino até que o quociente seja igual a zero. Após finalizar as divisões, devemos pegar o resto das divisões sempre “debaixo para cima”.

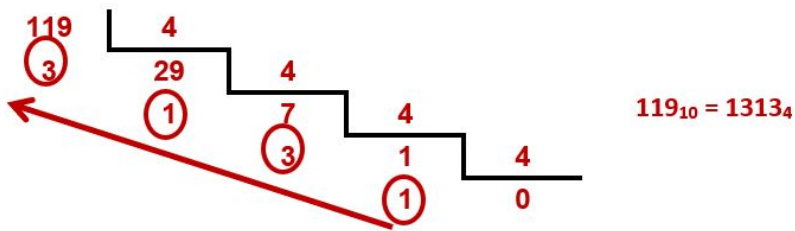
Por exemplo: $119_{10} = ?_2$



DECIMAL PARA QUALQUER BASE

Divisões inteiras sucessivas do número de origem pela base de destino até que o quociente seja igual a zero. Após finalizar as divisões, devemos pegar o resto das divisões sempre “debaixo para cima”.

Por exemplo: $119_{10} = ?_4$



Binário para octal

Agrupamento de dígitos é utilizado na conversão porque as bases de origem e destino são bases potência de 2.

Para conversão da base 2 para base 8, é necessário fazer o agrupamento em conjuntos de três dígitos.

$8 = 2^3$ à agrupar os dígitos em grupos de três, da direita para a esquerda e associar o octal correspondente a cada grupo.

Por exemplo: $110010011101_2 = ?_8$

110 010 011 101
 6 2 3 5 à $110010011101_2 = 6235_8$

Octal para binário

Converter cada dígito do número na base 8 em seu equivalente binário em três dígitos.

Por exemplo: $6235_8 = ?_2$

$\begin{array}{cccc} & 6 & 2 & 3 & 5 \\ 110 & 010 & 011 & 101 & \\ \end{array} \rightarrow 110010011101_2 = 6235_8$

Binário para hexadecimal

Agrupamento de dígitos é utilizado na conversão porque as bases de origem e destino são bases potência de 2.

Para conversão da base 2 para base 16, é necessário fazer o agrupamento em conjuntos de quatro dígitos.

$16 = 2^4$ → no binário, agrupar os dígitos em grupos de quatro, da direita para a esquerda e associar o hexadecimal correspondente a cada grupo.

Por exemplo: $110010011101_2 = ?_{16}$

-
 $\begin{array}{ccc} 1100 & 1001 & 1101 \\ C & 9 & D \end{array} \rightarrow 110010011101_2 = C9D_{16}$

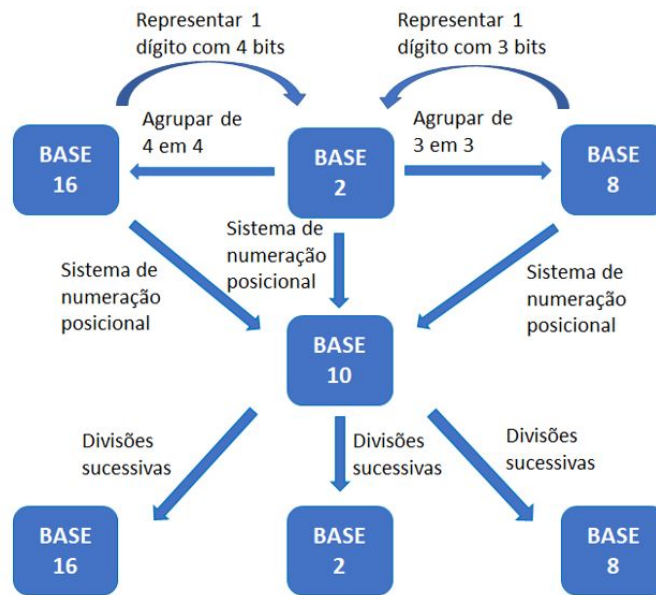
Hexadecimal para binário

Converter cada dígito do número na base 16 em seu equivalente binário em quatro dígitos.

Por exemplo: $C9D_{16} = ?_2$

-
 $\begin{array}{ccc} C & 9 & D \\ 1100 & 1001 & 1101 \end{array} \rightarrow 110010011101_2 = C9D_{16}$

O que aprendemos...



REPRESENTAÇÃO DE NÚMEROS INTEIROS COM SINAL

Como codificar o sinal de um número inteiro?

O sinal é representado pelo bit de mais alta ordem (o bit mais à esquerda).

- 1 – indica um número negativo
- 0 – indica um número positivo

Existem duas representações para o número inteiro com sinal:

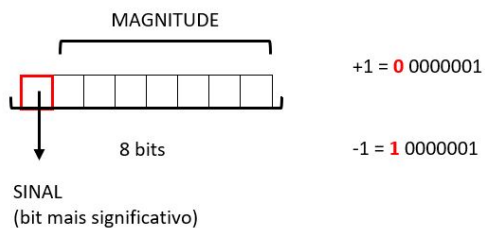
- Sinal-magnitude
- Complemento de dois

Sinal magnitude

- bit mais significativo – representa o sinal
- bits restantes – representam a magnitude (valor absoluto)

Por exemplo:

Representar os valores +1 e -1 em sinal magnitude utilizando 8 bits.



- Operações de adição e subtração:
 - se os sinais são iguais, adicione as magnitudes e use o mesmo sinal para o resultado;
 - se os sinais são diferentes, você deve determinar qual operando tem a magnitude maior. O sinal do resultado é o mesmo sinal do operando de maior magnitude e a magnitude deve ser obtida subtraindo (e não somando) o menor do maior;
 - cuidado ao incluir apenas os bits de magnitude na resposta.

SOMA				
0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	0 (vai 1)

SUBTRAÇÃO				
0	-	0	=	0
0*	-	1	=	1
1	-	0	=	1
1	-	1	=	0

* - pega emprestado

Ex₁: $79_{10} + 35_{10} = ?$

Ex₂: $99_{10} - 79_{10} = ?$

				1	1	1	1		
	0	1	0	0	1	1	1	1	79_{10}
+	0	0	1	0	0	0	1	1	35_{10}
	0	1	1	1	0	0	1	0	114_{10}

			0	1	1				
	0	1	1	1	1	1	1	1	99_{10}
	0	1	0	0	0	1	1	1	79_{10}
	0	0	0	1	0	1	0	0	20_{10}

Complemento de dois (2)

- É a representação usada nas máquinas atuais;
- simplifica a adição e subtração porque não utiliza a quantidade de regras de operações se comparado à representação sinal magnitude;
- a representação em complemento de dois é aplicada nos números inteiros negativos. Os números inteiros positivos são representados normalmente;
- complemento de dois de um número binário = complemento de 1 incrementado de 1

= inverte os bits e adicione 1

Por exemplo: Representação em notação de complemento de dois utilizando 8 bits

$23_{10} = 00010111_2$

$-23_{10} = ?_2$

1º passo: representar o número positivo correspondente em binário.

$23_{10} = 00010111_2$

2º passo: gerar o complemento de 1 do número positivo (inverter os bits).

$00010111_2 \rightarrow 11101000_2$

3º passo: somar um ao complemento de 1.

$11101000_2 + 1 = 11101001_2$

$-23_{10} = 11101001_2$

$-9_{10} = -(00001001_2) = 11110110_2 + 1 = 11110111_2$

- operações de adição e subtração

Com o uso da representação dos números negativos em complemento de dois, a subtração é resolvida por meio de uma adição. Dessa forma, as regras de operação, como explicitado na representação sinal magnitude, não existem na representação em complemento de dois.

Ex₁: $9_{10} + (-23_{10}) = ?$

	0	0	0	1	0	0	1		9_{10}
+	1	1	1	0	1	0	0	1	-23_{10}
	1	1	1	1	0	0	1	0	-14_{10}

Ex₂: $23_{10} - 9_{10} = ?$

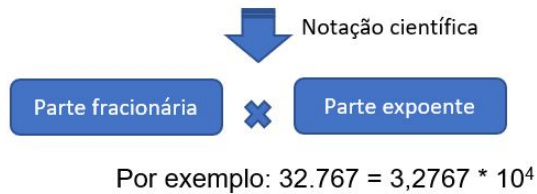
Carry é ignorado

	1	1	1		1	1	1		
	0	0	1	0	1	1	1		23_{10}
	1	1	1	1	0	1	1	1	9_{10}
	0	0	0	0	1	1	1	0	14_{10}

REPRESENTAÇÃO EM PONTO FLUTUANTE

Se uma palavra de 16 bits for utilizada em uma máquina, o maior inteiro que esse sistema poderia armazenar seria 32.767.

Então, como representar a população dos municípios do estado de São Paulo? E como representar números fracionários?



Por exemplo: $32.767 = 3,2767 * 10^4$

A notação científica simplifica os cálculos que envolvem números muito grandes ou muito pequenos e é utilizada na base decimal.

Em computadores digitais, a notação científica é representada por **ponto-flutuante** que consiste em três partes:



Onde:

Bit de sinal — { 0 – número positivo
1 – número negativo

Parte de expoente à representa o expoente de uma potência 2

Parte fracionária à mantissa ou significando (padrão IEEE)

Número de bits para cada parte:

§ Otimização para um intervalo (intervalo do menor valor para o maior valor) – **mais bits para o expoente**

§ Precisão (quantidade de informação que temos sobre um valor e quantidade de informação usada para representar o valor) – **mais bits na parte fracionária**

Por exemplo: representar o número 17_{10} em ponto flutuante

1º -- converter o número a ser representado para a base binária.

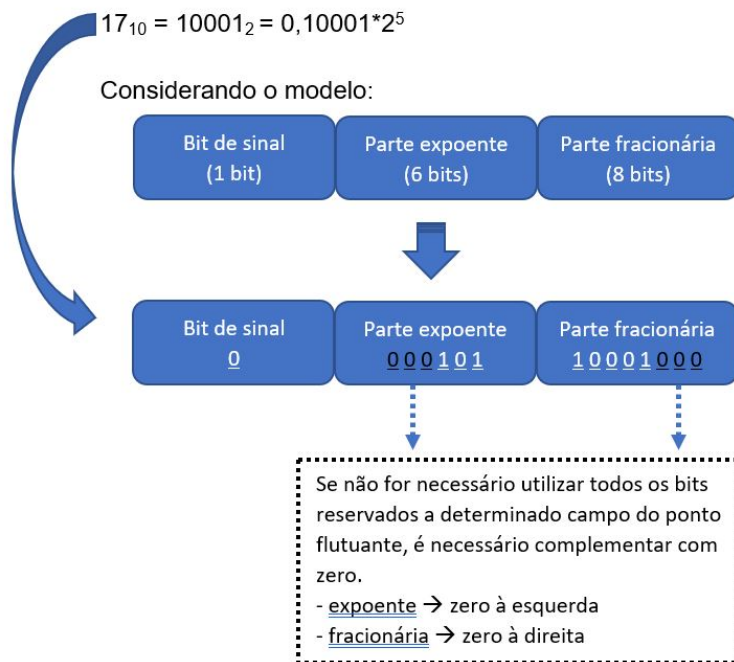
$$17_{10} = 10001_2$$

2º -- representar o número em um valor sem parte inteira (apenas parte fracionária). Manter o significado do número.

$$17_{10} = 10001_2 = 0,10001 * 2^5$$

Obs.:

- Valor inicial foi mantido.
- Não existe parte inteira.
- Cada posição que a vírgula "anda" no número, o expoente é acrescido de uma unidade.



Para refletir:

Não é possível representar expoentes negativos. Como representar o número $0,25_{10}$?

ÁLGEBRA BOOLEANA

Variáveis Booleanas só podem assumir um dentre dois valores possíveis, os quais podem ser denotados por [F,V] (falso ou verdadeiro), [H,L] (*high* e *low*) ou ainda [0,1].

Nos primórdios da eletrônica, todos os problemas eram solucionados por meio de sistemas analógicos. Com o avanço da tecnologia, os problemas passaram a ser solucionados pela eletrônica digital. Na eletrônica digital, os sistemas (computadores, processadores de dados, sistemas de controle, codificadores, decodificadores etc.) empregam um pequeno grupo de circuitos lógicos básicos, que são conhecidos como portas "e", "ou", "não" e flip-flop. Com a utilização adequada dessas portas, é possível implementar todas as expressões geradas pela álgebra de Boole.

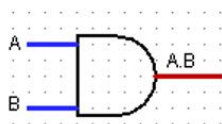
OPERAÇÕES ou FUNÇÕES

Todas as funções booleanas podem ser representadas a partir destas operações básicas.

▪ E (AND)

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

entradas ↓ saída

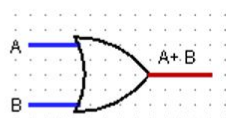


Onde:
0 – falso
1 – verdadeiro

- O resultado da operação **E** é 1 (verdadeiro), se e somente se, todas as entradas valerem 1 (forem verdadeiras).
- O resultado da operação **E** é 0 (falso) se pelo menos uma das entradas for 0 (falso).

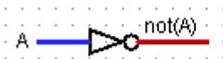
▪ OU (OR)

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



▪ NEGAÇÃO (NOT)

A	not(A)
0	1
1	0



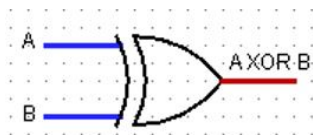
Obs.: a negação de uma entrada também pode ser escrita utilizando a barra de negação.

$$\text{not}(A) = \overline{A}$$

OUTRAS OPERAÇÕES

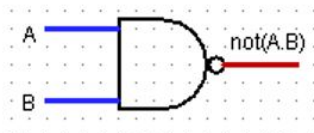
- OU-EXCLUSIVO (XOR)

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0



NE (NAND)

A	B	not(A.B)
0	0	1
0	1	1
1	0	1
1	1	0

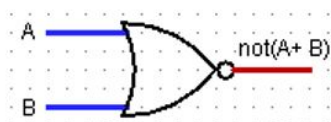


Obs.: a negação de operação pode ser representada usando a barra de negação em toda a operação —

$$\text{not}(A.B) = \overline{A.B}$$

NOU (NOR)

A	B	not(A+B)
0	0	1
0	1	0
1	0	0
1	1	0



PROPRIEDADES

Lei Comutativa

- $A . B = B . A$
- $A + B = B + A$
- $A \oplus B = B \oplus A$

Lei Associativa

- $(A . B) . C = A . (B . C)$
- $(A + B) + C = A + (B + C)$
- $(A \oplus B) \oplus C = A \oplus (B \oplus C)$

Lei Distributiva

- $A . (B + C) = (A . B) + (A . C)$
- $A + (B . C) = (A + B) . (A + C)$

TABELA VERDADE

Tabela verdade é um tipo de tabela matemática usada em lógica para determinar se uma fórmula (expressão) é válida ou não.

Verificaremos se algumas das propriedades apresentadas são válidas ou não.

Ex01: $A + B = B + A$

1º passo – montar a tabela verdade para a expressão do lado esquerdo da igualdade

2º passo – montar a tabela verdade para a expressão do lado direito da igualdade

3º passo – comparar os resultados das colunas. Verificar se todas as linhas são iguais.

A	B	A + B	B + A
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Importante!

Duas variáveis diferentes em toda expressão indicam que a tabela verdade deve ter quatro linhas que representam todas as combinações de 0s e 1s para estas duas variáveis.

$$\text{QtidadeLinhas} = 2^{\text{qtdeVariáveis}}$$

Ex02: $A + (B \cdot C) = (A + B) \cdot (A + C)$

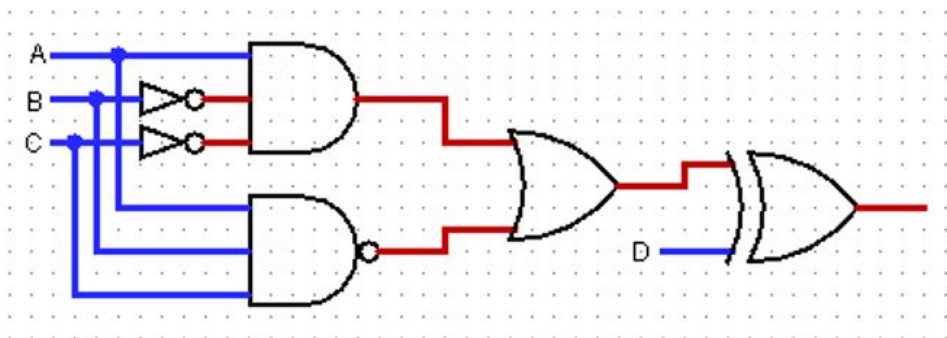
Dica: se a expressão a ser verificada for muito longa, é importante dividir em partes menores para evitar erros.

A	B	C	$B \cdot C$	$A + (B \cdot C)$	$A + B$	$A + C$	$(A+B)(A+C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

DIAGRAMAS ou CIRCUITOS







Forma de representar as expressões booleanas em circuitos.

Ex01: $((A \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot C)) \oplus D$



RESUMO

O que vimos?

Nome	Símbolo Gráfico	Função Algébrica	Tabela Verdade															
E (AND)		$S=A.B$ $S=AB$	<table><tr><th>A</th><th>B</th><th>S=A.B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	S=A.B	0	0	0	0	1	0	1	0	0	1	1	1
A	B	S=A.B																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OU (OR)		$S=A+B$	<table><tr><th>A</th><th>B</th><th>S=A+B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	S=A+B	0	0	0	0	1	1	1	0	1	1	1	1
A	B	S=A+B																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NÃO (NOT) Inversor		$S=\bar{A}$ $S=A'$ $S=\neg A$	<table><tr><th>A</th><th>S=\bar{A}</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	S= \bar{A}	0	1	1	0									
A	S= \bar{A}																	
0	1																	
1	0																	
NE (NAND)		$S=\overline{A.B}$ $S=(A.B)'$ $S=\neg(A.B)$	<table><tr><th>A</th><th>B</th><th>S=$\overline{A.B}$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	S= $\overline{A.B}$	0	0	1	0	1	1	1	0	1	1	1	0
A	B	S= $\overline{A.B}$																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOU (NOR)		$S=\overline{A+B}$ $S=(A+B)'$ $S=\neg(A+B)$	<table><tr><th>A</th><th>B</th><th>S=$\overline{A+B}$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	S= $\overline{A+B}$	0	0	1	0	1	0	1	0	0	1	1	0
A	B	S= $\overline{A+B}$																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$S=A\oplus B$	<table><tr><th>A</th><th>B</th><th>S=A⊕B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	S=A⊕B	0	0	0	0	1	1	1	0	1	1	1	0
A	B	S=A⊕B																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

(Notas de Aula – Prof. Jean Marcos Laine)