AULA 1 – SISTEMA BINÁRIO

OBJETIVO DA AULA

Compreender a importância da base binária para a arquitetura do computador.

APRESENTAÇÃO

Veremos agora como o computador armazena e processa as informações que são tão importantes para nós.

Como uma máquina que trabalha com sinais elétricos, o computador precisa "entender" aquilo que queremos e escrevemos através de programas.

Quando criamos programas, usamos linguagens de programação que se assemelham à linguagem humana. Mas como isso chega aos circuitos eletrônicos? A resposta é um sistema que vai traduzindo (compilando) de um nível mais alto até o nível mais baixo, o do computador.

Vamos agora ver como isso funciona, trabalhando com as bases numéricas.

1. SISTEMA BINÁRIO

Os computadores digitais, tão presentes em nosso dia a dia, trabalham em dois níveis de tensão e a forma mais adequada para representar isso é a notação ou sistema binário.

A primeira descrição conhecida de um sistema numérico binário foi apresentada pelo matemático indiano Pingala no século III a.C, representando os números de 1 a 8 com a sequência 001, 010, 011, 100, 101, 110, 111 e 1000.

FIGURA 1 | **Pingala, o "pai" da numeração binária**

Foto: www.cuemath.com

O termo *bit* é uma abreviação da expressão *binary digit*. O agrupamento de quatro bits é chamado de *nibble* e o agrupamento de oito bits é chamado de *byte*.

Um computador trabalha basicamente executando sequências de instruções, que conhecemos como programas. Essas instruções são buscadas na memória, decodificadas e executadas.

Porém, como elas e os dados ficam guardados na memória? Como o processador sabe o que elas significam?

Vamos ver como isso acontece.

Uma instrução é armazenada no sistema de memórias de uma forma que representamos utilizando zeros e uns, como a Figura 2.

FIGURA 2 | **Sequência de uma instrução em números binários**

10011001101011001111010110010110

Fonte: Autor

No exemplo acima, temos uma instrução utilizando 32 bits que, para nós, pode não significar absolutamente nada a princípio, mas para o processador ela diz muito, na verdade, tudo, já que contém todas as informações de que ele precisa para executá-la.

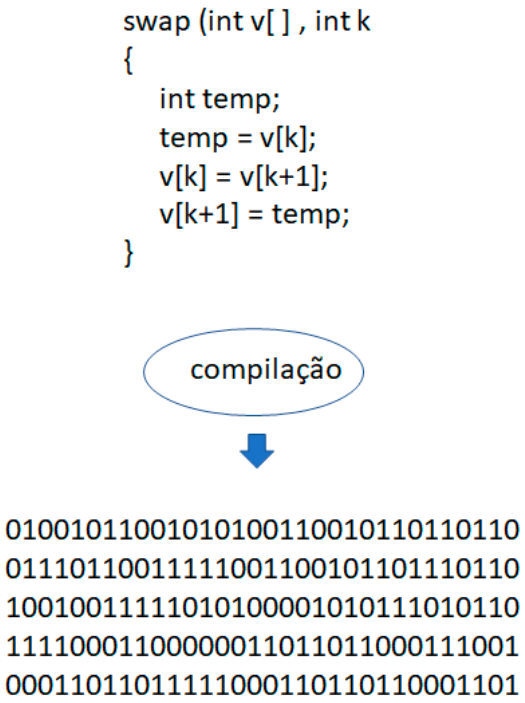
E que informações seriam essas? Por exemplo, a sequência de bits destacada em negrito, em uma instrução de um processador hipotético, pode representar o *opcode* (código de operação), que informa ao processador qual operação ele deverá executar. Esta operação pode ser uma soma, subtração, comparação, ou qualquer operação lógica, ou aritmética.

Além disso, os *bits* de uma instrução têm outras funções, tais como o endereço onde encontrar os dados que a instrução manipulará, o endereço onde o resultado será armazenado, bem como se a instrução trabalha com inteiros ou não inteiros, entre outras.

Todo processador tem uma espécie de vocabulário que ele consegue entender e executar e isso é chamado de **arquitetura do conjunto de instruções** *(ISA – Instruction Set Architecture)*. Essa arquitetura define como as instruções são codificadas em binário e contém todas as instruções que o processador conseguirá executar.

Nós, humanos, temos nossa linguagem própria e as linguagens de programação se aproximam dessa linguagem. Mas os computadores não têm a capacidade de entender a linguagem humana diretamente. Tudo tem que ser “traduzido” para a linguagem que eles entendem. Esse processo é chamado de **compilação** e transforma tudo o que escrevemos em um código representado pela notação binária, conforme mostrado na Figura 3.

FIGURA 3 | **O processo de compilação**



Elaborado pelo autor.

Na Figura 3 temos, na parte de cima, um programa escrito em uma linguagem de programação muito conhecida e usada por programadores. Após o processo de compilação, o programa é "traduzido" para um código em binário que o computador conseguirá interpretar e executar porque está numa linguagem adequada a ele.

Vale lembrar aqui que esses zeros e uns são apenas uma forma de representarmos os dois níveis de tensão em que os computadores trabalham, como mostramos acima.

Portanto, quando nos referimos a um programa em **linguagem de máquina**, estamos nos referindo àquele representado em código binário, ou seja, que o computador consegue entender e executar.

Importante observar que, embora seja muito difícil para um humano entender um código representado em binário, há profissionais que precisam fazer isso, já que o próprio compilador é um programa como outro qualquer e alguém tem que entender a correlação entre o código em alto nível (usado pelos programadores) e o código em baixo nível (interpretado e executado pelos computadores).

A notação binária é, como as notações decimal, octal e hexadecimal, que também estudaremos neste curso, uma notação posicional, o que significa que o valor de cada algarismo depende da sua posição no número.

Quando nos referimos à posição dos bits em um número, consideramos como *bits* mais significativos ou menos significativos e isso depende da posição de cada *bit* nesse número.

Os *bits* mais significativos são aqueles que estão mais à esquerda e, consequentemente, os menos significativos estão mais à direita, conforme mostramos na Figura 4.

FIGURA 4 | **Bit mais significativo e menos significativo**

Elaborado pelo autor.

A posição de cada *bit* terá uma participação fundamental quando tratarmos da conversão entre bases.

Como a notação é binária (só admite dois valores) precisaremos trabalhar com as potências de 2, já que a posição de cada *bit* terá o peso da respectiva posição. É importante lembrar que a contagem das posições começa em zero.

Assim, o *bit* menos significativo será equivalente a 2^0, o segundo *bit* menos significativo será equivalente a 2^1, o terceiro *bit* menos significativo será equivalente a 2^2 e assim por diante.

Observe a Figura 5.

FIGURA 5 | **Os bits e seus valores posicionais**

Elaborado pelo autor.

Na Figura 5 observamos o **peso** de cada bit relacionado à sua posição no conjunto e a respectiva potência de 2. Usaremos isso, na prática, quando estudarmos as conversões de base e aritmética em binário.

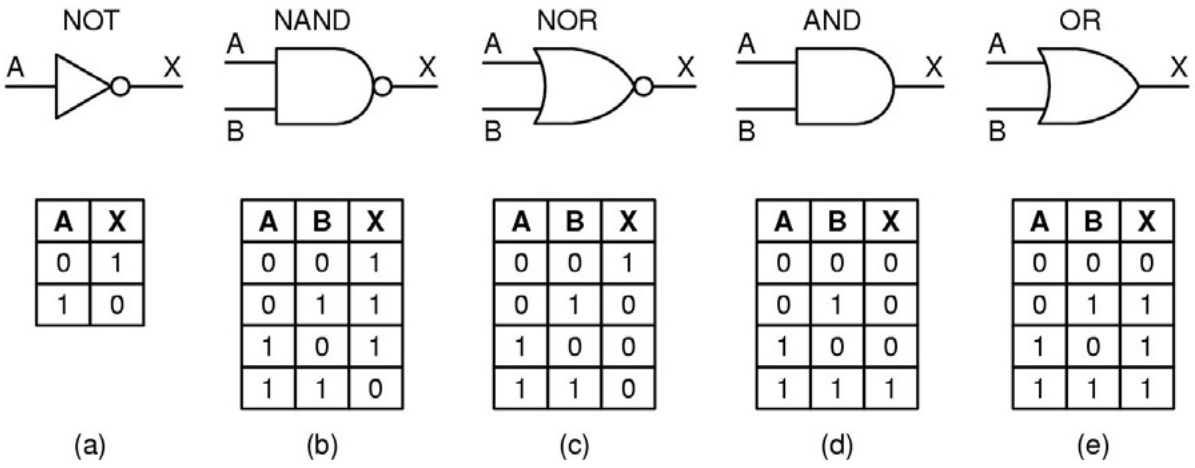
1.1. A ÁLGEBRA DE BOOLE E A LÓGICA BINÁRIA

Na matemática e na ciência da computação, a Álgebra de Boole é composta de estruturas algébricas que representam as operações lógicas E, OU e NÃO, além das operações, envolvendo a Teoria dos Conjuntos. Ela também é a base da matemática computacional, baseada em números binários.

Recebeu o nome de George Boole, matemático inglês, que viveu no século XIX, e a definiu como parte de um sistema de lógica. Hoje, a álgebra booleana tem aplicações na eletrônica e na computação.

Os operadores de álgebra booleana podem ser representados de várias formas e são usados na criação de circuitos lógicos, que veremos mais adiante.

FIGURA 6 | **As portas que compõem os circuitos e sua representação binária**



A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

Fonte: http://www.dpi.inpe.br/~carlos/Academicos/Cursos/ArqComp/aula_5.html

1.2. A CODIFICAÇÃO DE CARACTERES EM BINÁRIO

Há diversos códigos ou sistemas de representação de caracteres utilizando a base binária. Alguns exemplos são:

- EBCDIC (*Extended Binary Coded Decimal Interchange Code* – Código Decimal de Intercâmbio Codificado em Binário Estendido);
- BCD (*Binary Coded Decimal* – Decimal Codificado em Binário);
- ASCII (*American Standard Code for Information Interchange* – Código Padrão Americano para Intercâmbio de Informações), muitas vezes referido como ASC2 por questões de praticidade.

O ASCII utiliza inicialmente 7 bits para a representação de caracteres, o que possibilita 128 (27) combinações possíveis, mas ainda há uma versão estendida, o *Extended* ASCII, que permite outras 128 combinações para a representação de símbolos e caracteres especiais.

Porém, como 256 caracteres não são suficientes para suprir todas as línguas existentes no mundo, com suas letras, símbolos e caracteres, o código conhecido como *Unicode* tem ganho cada vez mais aplicações, já que permite representações com 8, 16 e 32 bits, aumentando consideravelmente as possibilidades de representação.

FIGURA 7 | **Codificação em binário de caracteres em ASCII**

CODIFICACIÓN BINARIA ASCII

Character	Binary Code	Character	Binary Code	Character	Binary Code	Character	Binary Code	Character	Binary Code
A	01000001	Q	01010001	g	01100111	w	01110111	-	00101101
B	01000010	R	01010010	h	01101000	x	01111000	.	00101110
C	01000011	S	01010011	i	01101001	y	01111001	/	00101111
D	01000100	T	01010100	j	01101010	z	01111010	0	00110000
E	01000101	U	01010101	k	01101011	!	00100001	1	00110001
F	01000110	V	01010110	l	01101100	"	00100010	2	00110010
G	01000111	W	01010111	m	01101101	#	00100011	3	00110011
H	01001000	X	01011000	n	01101110	$	00100100	4	00110100
I	01001001	Y	01011001	o	01101111	%	00100101	5	00110101
J	01001010	Z	01011010	p	01110000	&	00100110	6	00110110
K	01001011	a	01100001	q	01110001	'	00100111	7	00110111
L	01001100	b	01100010	r	01110010	(	00101000	8	00111000
M	01001101	c	01100011	s	01110011	)	00101001	9	00111001
N	01001110	d	01100100	t	01110100	*	00101010	?	00111111
O	01001111	e	01100101	u	01110101	+	00101011	@	01000000
P	01010000	f	01100110	v	01110110	,	00101100	_	01011111

Fonte: www.areatecnologia.com/informatica/codificacion-binaria.html

A notação binária está bastante presente em outras áreas da TI, como, por exemplo, os sistemas operacionais e as redes de computadores, onde as divisões de redes em sub redes utilizam o **cálculo de máscara de sub rede**, feito em binário. Embora esse tópico fuja ao escopo deste curso, é importante saber o quanto o conhecimento da notação binária é fundamental para o profissional de TI.

CONSIDERAÇÕES FINAIS

Vimos nesta aula alguns aspectos muito importantes no estudo da notação binária:

- A representação de números e caracteres;
- A álgebra booleana e os circuitos digitais;
- As aplicações da notação binária em diversas áreas da TI.

Como vimos, o código binário representa a linguagem de máquina, adequada ao computador. O processo de compilação nos ajuda a transformar o que queremos passar para o computador na linguagem que ele entende (linguagem de máquina).

É fundamental que você tenha entendido a importância e as características da notação binária para nossa próxima aula, em que trataremos não só dela, mas também das notações octal e hexadecimal.

Até lá!

MATERIAIS COMPLEMENTARES

Assista a esse pequeno vídeo sobre a comparação entre a linguagem binária e a linguagem humana, por Pierre Lévy. Link: https://www.youtube.com/watch?v=JN2JrJYR1TA&t=21s.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e organização de computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas operacionais modernos*. 3ª edição. Editora Pearson. Livro (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

AULA 2 – SISTEMAS DE NUMERAÇÃO E CONVERSÃO DE BASES

OBJETIVO DA AULA

Aprender a converter números entre as bases decimal, binária, octal e hexadecimal.

APRESENTAÇÃO

Nesta aula veremos como fazer as conversões entre as bases numéricas importantes para o computador: decimal, binária, octal e hexadecimal.

O processo é muito simples e intuitivo e, com poucos exemplos, você estará habilitado a efetuar essas conversões.

Trata-se de uma aula extremamente prática.

Vamos lá!

1. SISTEMA DE NUMERAÇÃO E CONVERSÃO DE BASES

Para começar nosso estudo, vamos lembrar que trabalharemos com notações posicionais, o que significa que o valor de cada algarismo em um número depende de sua posição.

Nas bases em que vamos trabalhar, é importante lembrar que:

- A base binária utiliza apenas os algarismos 0 e 1;
- A base octal utiliza os algarismos 0, 1, 2, 3, 4, 5, 6 e 7;
- A base hexadecimal utiliza os algarismos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 e as letras A, B, C, D, E e F.

Outra coisa importante para nosso trabalho é que a contagem de posições é feita da direita para a esquerda, começando em zero.

1.1. CONVERSÃO DE BINÁRIO PARA DECIMAL

Considere o número $(1100110)_2$. Vamos ver como convertê-lo para a base decimal.

Os passos são os seguintes:

1) Contar as posições de cada bit da direita para a esquerda, começando por zero:

2) Multiplicar cada algarismo pela potência de 2 referente à sua posição, somando esses valores:

$$1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 =$$

$$64 + 32 + 0 + 0 + 4 + 2 + 0 = 102$$

3) Assim, $(1100110)2 = (102)10$.

Para efeitos práticos, você pode simplesmente desprezar os zeros e considerar somente as posições com algarismo igual a um.

1.2. CONVERSÃO DE OCTAL PARA DECIMAL

Aqui os passos são exatamente os mesmos, com a diferença de que teremos algarismos de 0 a 7.

Considere o número $(2703)_8$. Vamos ver como convertê-lo para a base decimal.

Seguindo os mesmos passos:

1) Contar as posições de cada bit da direita para a esquerda, começando por zero:

$$\overset{3}{2}\ \overset{2}{7}\ \overset{1}{0}\ \overset{0}{3}$$

2) Multiplicar cada algarismo pela potência de 8 referente à sua posição, somando esses valores:

$$2 \times 8^3 + 7 \times 8^2 + 0 \times 8^1 + 3 \times 8^0 = 2 \times 512 + 7 \times 64 + 0 \times 8 + 3 \times 1 = 1024 + 448 + 3 = 1475$$

3) Assim, $(2703)8 = (1475)10$.

1.3. CONVERSÃO DE HEXADECIMAL PARA DECIMAL

Aqui trabalharemos com os algarismos de 0 a 9 e com as letras de A a F e, para facilitar nosso trabalho, vamos usar uma tabela de correspondência entre as letras e seus valores: A = 10, B = 11, C = 12, D = 13, E = 14 e F = 15.

Considere o número $(B3F)_{16}$. Vamos ver como convertê-lo para a base decimal.

Seguindo os mesmos passos:

1) Contar as posições de cada bit da direita para a esquerda, começando por zero:

$$\overset{2}{B}\ \overset{1}{3}\ \overset{0}{F}$$

2) Multiplicar cada algarismo pela potência de 8 referente à sua posição, somando esses valores:

$$B \times 162 + 3 \times 161 + F \times 160 = 11 \times 162 + 3 \times 161 + 15 \times 160 =$$

$$11 \times 256 + 3 \times 16 + 15 \times 1 = 2816 + 48 + 15 = 2879$$

3) Assim, (B3F)16 = (2879)10.

O algoritmo para a conversão é semelhante nos três casos vistos e precisamos ter alguns cuidados:

- Contar as posições da direita para a esquerda;
- Começar a contagem a partir de 0;
- Lembrar que $x^0 = 1$ (muitos alunos se distraem nesse ponto).

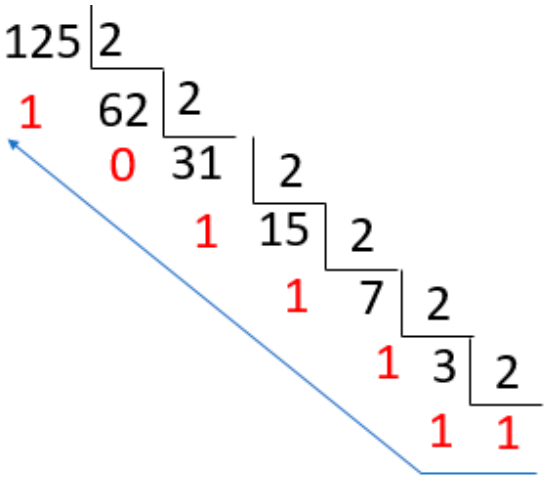
Vamos ver agora como convertemos da base 10 para as bases binária, octal e hexadecimal.

1.4. CONVERSÃO DA BASE DECIMAL PARA A BASE BINÁRIA

Para essa conversão, o raciocínio é o seguinte: vamos dividindo o número por 2 até não podermos mais e anotando os restos das respectivas divisões. Ao final, juntamos o último quociente com todos os restos, do último até o primeiro.

Considere o número $(125)_{10}$.

A divisão deverá acontecer até o quociente ser menor ao divisor.



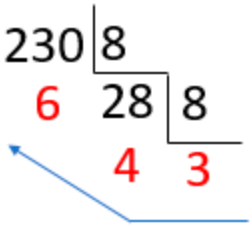
Copiando o último quociente seguido dos restos na direção mostrada, teremos $(1111101)_2$.

Assim, $(125)_{10} = (1111101)_2$.

1.5. CONVERSÃO DA BASE DECIMAL PARA A BASE OCTAL

Aqui usaremos exatamente o mesmo algoritmo da conversão anterior, com a diferença de que o divisor agora é 8.

Vamos considerar o número $(230)_{10}$ e repetir o raciocínio usado para a base binária. A divisão por 8 deverá acontecer até o quociente ser menor ao divisor.



Da mesma forma que fizemos com o binário, vamos copiar o último quociente seguido dos restos a ordem mostrada pela seta e teremos $(346)_8$.

Assim, $(230)_{10} = (346)_8$.

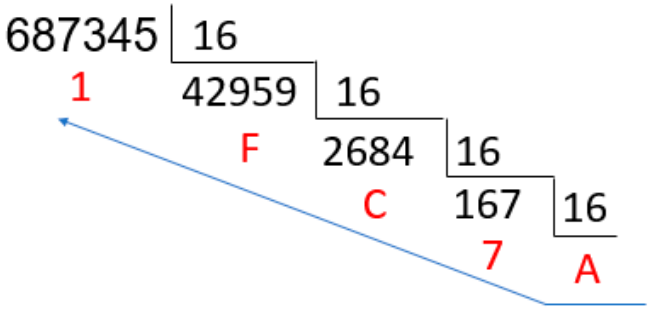
1.6. CONVERSÃO DA BASE DECIMAL PARA A BASE HEXADECIMAL

Novamente, vamos usar o mesmo algoritmo que usamos para as bases binária e octal.

Mas aqui teremos que tomar um cuidado extra: sempre que um resto ou quociente for um número entre 10 e 15 teremos que substituí-lo pela letra correspondente.

Vale colocar a tabela a seu lado quando estiver praticando.

Considere o número $(687345)_{10}$. Vamos dividi-lo por 16 até o quociente ser menor ao divisor e usar o último quociente seguido dos restos, como fizemos nas bases anteriores.



Da mesma forma que fizemos com o binário e o octal, copiaremos o último quociente seguido dos restos a ordem mostrada pela seta e teremos $(A7CF1)_{16}$.

Repare que nos restos que ficaram entre 10 e 15, bem como no último quociente, foram substituídos pelas letras correspondentes.

Assim, $(687345)_{10} = (A7CF1)_{16}$.

Vimos que os algoritmos são semelhantes para qualquer base que precisemos converter.

É apenas uma questão de prática para que você faça essas conversões de forma rápida e segura.

1.7. CONVERSÃO DA BASE BINÁRIA PARA A BASE OCTAL

Podemos fazer essa conversão em duas etapas: primeiro convertemos da base binária para a base decimal e em seguida convertemos da base decimal para a octal.

Porém, há um *macete* para convertemos diretamente. Observe o passo a passo, que é bastante simples.

Considere o número $(100110001)_2$:

O primeiro passo é agrupar os algarismos de 3 em 3, da direita para a esquerda, completando o trio mais à esquerda com zeros, se necessário.

100 110 001.

Em seguida, convertemos cada grupo para o valor equivalente em *decimal*.

Teremos então: $(100)_2 = 4$ $(110)_2 = 6$ $(001)_2 = 1$.

Assim, $(100110001)_2 = (461)_8$.

1.8. CONVERSÃO DA BASE OCTAL PARA A BASE BINÁRIA

O processo agora é inverso. Separamos os algarismos e os convertemos para a base binária, completando com zeros à esquerda quando necessário.

Considere o número $(602)_8$.

Convertendo os algarismos:

$6 = (110)_2$.

$0 = (0)_2$ (completamos os três dígitos com zeros à esquerda, ficando 000).

2 = (10)2 (completamos os três dígitos com um zero à esquerda, ficando 010).

Juntando os valores em binário, teremos $(110000010)_2$.

Assim, $(602)_8 = (110000010)_2$.

1.9. CONVERSÃO DA BASE BINÁRIA PARA A BASE HEXADECIMAL

O processo aqui é semelhante à conversão de binário para octal, considerando que o agrupamento agora é de 4 dígitos e quando o valor for entre 10 e 15 substituiremos pela letra correspondente.

Considere o número $(1001110111111)_2$.

Agrupando de 4 em 4 teremos: 1 0011 1011 1111.

Como o bit mais à esquerda ficou sozinho, vamos colocar zeros à esquerda.

Convertendo:

$(0001)_2 = 1$.

$(0011)_2 = 3$.

$(1011)_2 = 11$ (usaremos a letra B).

(1111)2 = 15 (usaremos a letra F).

Assim, $(1001110111111)_2 = (13BF)_{16}$.

1.10. CONVERSÃO DA BASE HEXADECIMAL PARA A BASE BINÁRIA

Novamente, o processo é semelhante ao usado na conversão de octal para binário.

Agora vamos converter cada dígito do número em hexadecimal para binário, completando com zeros à esquerda quando necessário.

Considere o número $(A3BC)_{16}$.

Vamos converter cada um deles para binário:

A (=10) = $(1010)_{2.}$

3 = $(11)_2$; completando com zeros à esquerda teremos $(0011)_2$.

B (=11) = $(1011)_2$.

C (=12) = $(1100)_2$.

Juntando todos eles, teremos $(1010\ 0011\ 1011\ 1100)_2$.

Assim, $(A3BC)_{16} = (1010001110111100)_2$.

CONSIDERAÇÕES FINAIS

Vimos nesta unidade como converter entre as principais bases usadas em diversos setores da computação: binária, octal e hexadecimal.

As conversões entre essas bases têm como fundamento principal o fato de que trabalhamos com notação posicional em todas elas. Isso torna os algoritmos de conversão semelhantes, seja qual for o caso.

Alguns cuidados são necessários aqui. Em primeiro lugar, a posição de um algarismo em um determinado número é sempre contada da direita para a esquerda, começando por zero. Essa posição determina o valor desse algarismo no número.

Um lembrete final, que pega alguns aprendizes distraídos, é que qualquer número elevado a zero é igual a 1. Não esqueça!

Na próxima aula aprenderemos a aritmética nessas bases. Até lá!

MATERIAIS COMPLEMENTARES

Neste vídeo você poderá rever tudo o que foi apresentado nessa aula de forma didática e simples. Link: https://www.youtube.com/watch?v=0DBUj8ZHAGk.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e organização de computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas operacionais modernos*. 3ª edição. Editora Pearson. Livro (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

AULA 3 – SOMA, SUBTRAÇÃO, DIVISÃO E MULTIPLICAÇÃO

OBJETIVO DA AULA

Efetuar operações aritméticas nas bases binária, octal e hexadecimal.

APRESENTAÇÃO

Na aula anterior conhecemos as bases binária, octal e hexadecimal. Além disso, aprendemos como converter números entre elas.

Agora que já sabemos como funcionam as bases binária, octal e hexadecimal e como converter números entre elas, está na hora de aprendermos a somar, subtrair, multiplicar e dividir nelas.

As operações funcionam da mesma forma que na base decimal. Então, é hora de começarmos.

1. SOMA EM BINÁRIO

Para começarmos, vamos lembrar que:

0 + 0 = 0.

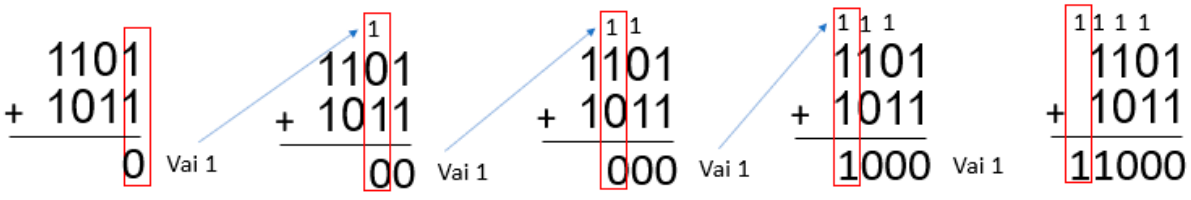
0 + 1 = 1.

1 + 0 = 1.

1 + 1 = 10 (= 2 em binário – vamos usar o “vai 1”).

1 + 1 + 1 = 11 (= 3 em binário – e vamos usar o “vai 1” também).

Como exemplo, vamos somar os números $(1101)_2$ e $(1011)_2$.



Repare que o processo é o mesmo utilizado nas somas em decimal. E você verá que em octal e hexadecimal será da mesma forma.

1.1. SOMA EM OCTAL

Aqui precisamos lembrar que, como não usamos os algarismos 8 e 9, sempre que o resultado da soma igualar ou passar de 8, teremos o *vai 1*.

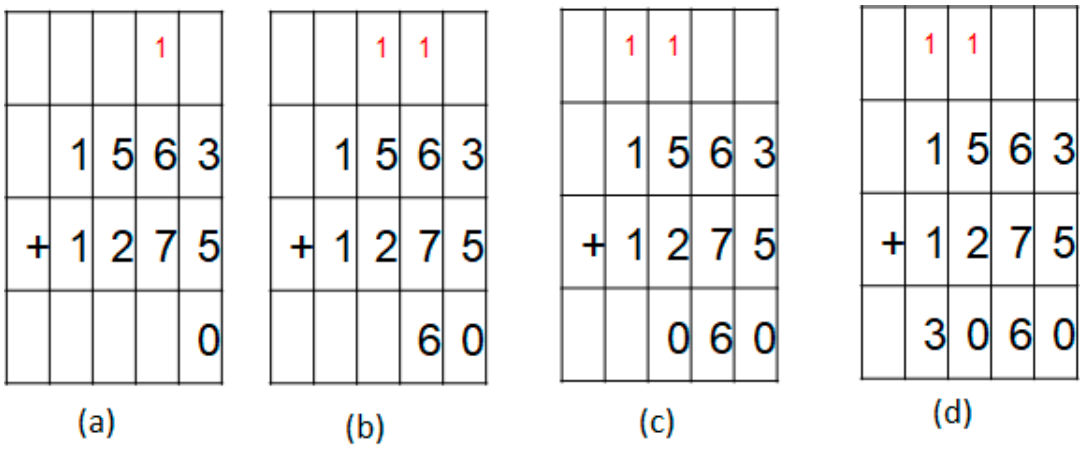
O algoritmo é simples:

1º caso: o resultado da soma está entre 0 e 7 – simplesmente o escrevemos, sem precisar o *vai 1*.

2º caso: o resultado iguala ou ultrapassa 8 – então executaremos os seguintes passos:

- Somamos 1 à próxima coluna (vai 1);
- Na coluna atual colocamos o quanto ultrapassou a base (no caso, 8).

Observe o exemplo abaixo, somando $(1563 + 1275)_8$.



a) 3 + 5 = 8 → colocamos o resultado igual a 0 (zero), pois devemos subtrair 8 (resultado maior do que 7) por 8 (a base octal). Então fica: 8 – 8 = 0. Devemos adicionar 1 a próxima coluna a esquerda, pois a regra informa que ao ultrapassar o valor, subtraímos, inserirmos o resultado e adicionamos 1 a próxima coluna a esquerda.

b) 1 + 6 + 7 = 14 → colocamos o resultado igual a 6, pois devemos subtrair 14 (resultado maior do que 7) por 8 (a base octal). Então fica: 14 – 8 = 6. Devemos adicionar 1 a próxima coluna a esquerda, pois a regra informa que ao ultrapassar o valor, subtraímos, inserirmos o resultado e adicionamos 1 a próxima coluna a esquerda.

c) 1 + 5 + 2 = 8 → colocamos o resultado igual a 0 (zero), pois devemos subtrair 8 (resultado maior do que 7) por 8 (a base octal). Então fica: 8 – 8 = 0. Devemos adicionar 1 a próxima coluna a esquerda, pois a regra informa que ao ultrapassar o valor, subtraímos, inserirmos o resultado e adicionamos 1 a próxima coluna a esquerda.

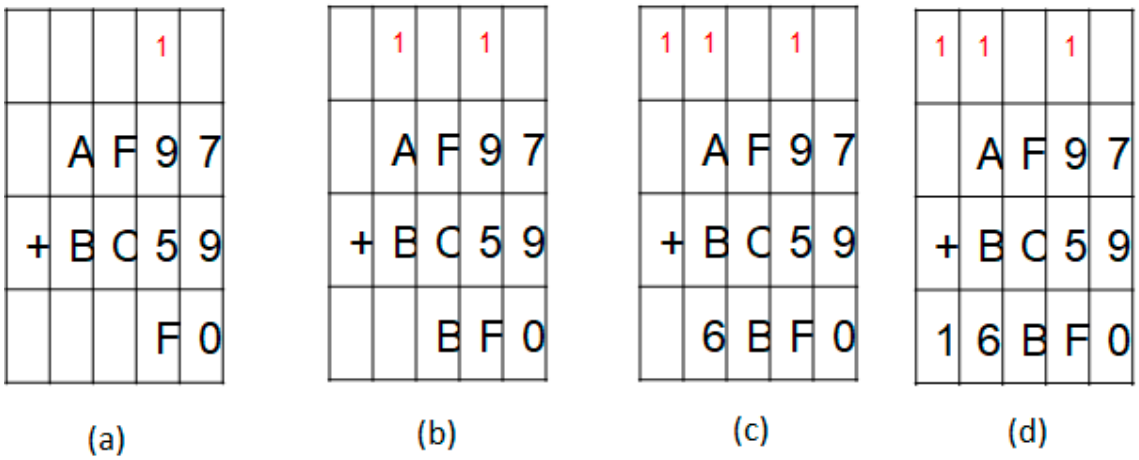
d) 1 + 1 + 1 = 3 → colocamos o resultado igual a 3 e, como está entre 0 e 7, não precisamos adicionar nada na próxima coluna.

1.2. SOMA EM HEXADECIMAL

Aqui usaremos o mesmo raciocínio. Mas lembre-se: como estamos trabalhando na base decimal, tudo o que fizemos em relação a 8 na base octal faremos em relação ao 16 na base hexadecimal.

Assim, quando a soma ficar entre 0 e 15, usaremos o algarismo (ou letra) correspondente, sem precisar adicionar 1 à próxima coluna. E quando igualar ou passar de 16 utilizaremos o mesmo raciocínio do octal.

Como exemplo, vamos efetuar $(AF97 + BC59)_{16}$.



a) 7 + 9 = 16 → colocamos o resultado igual a 0 (zero), pois devemos subtrair 16 (resultado maior do que 15) por 16 (a base hexadecimal). Então fica: 16 – 16 = 0. Devemos adicionar 1 a próxima coluna a esquerda, pois a regra informa que ao ultrapassar o valor, subtraímos, inserirmos o resultado e adicionamos 1 a próxima coluna a esquerda.

b) 1 + 9 + 5 = 15 → como 15 é menor do que 16, colocamos a letra correspondente ao 15, a letra F.

c) 15(F) + 12(C) = 27 → colocamos o resultado igual a B (11), pois devemos subtrair 27 (resultado maior do que 15) por 16 (a base hexadecimal). Então fica: 27 – 16 = 11 (B). Devemos adicionar 1 a próxima coluna a esquerda, pois a regra informa que ao ultrapassar o valor, subtraímos, inserirmos o resultado e adicionamos 1 a próxima coluna a esquerda.

d) 1 + 10(A) + 11(B) = 22 → colocamos o resultado igual a 6, pois devemos subtrair 22 (resultado maior do que 15) por 16 (a base hexadecimal). Então fica: 22 – 16 = 6. Devemos adicionar 1 a próxima coluna a esquerda, pois a regra informa que ao ultrapassar o valor, subtraímos, inserirmos o resultado e adicionamos 1 a próxima coluna a esquerda. Neste caso, ao subir o 1, consequentemente faremos a 1 + 0 (nada, vazio da coluna) e será igual a 1. Resultado: 16BF0

2. SUBTRAÇÃO EM BINÁRIO

Agora vamos usar, como em decimal, o recurso de “pedir emprestado”. Lembrando uma regra simples: **quem empresta perde um e quem ganha emprestado ganha o valor da base.** Isso valerá para todas as bases com que trabalhamos aqui.

Vamos efetuar, como exemplo $(110001 - 101111)_2$.

```
  110001
- 101111
--------
```

Subtração trivial

```
  0 1 1 10
  110001
- 101111
--------
       10
```

Como foi necessário “pedir emprestado”, o primeiro 1 encontrado à esquerda vira zero e haverá uma sequência de empréstimos até a coluna em que estamos trabalhando. Nesse caso o 0 ganha 2 (10). Teremos então $(10 - 1)_2 = 1$

```
  0 1 1
  110001
- 101111
--------
      010
```

A partir daqui as subtrações são triviais e só teremos zeros à esquerda

2.1. SUBTRAÇÃO EM OCTAL

Aqui também precisaremos do recurso de “pedir emprestado”, lembrando de que quem emprestar perderá 1 e quem receber ganhará 8.

Vamos ao exemplo efetuando $(7214 - 1636)_8$.

```
  7214
- 1636
------
```

Como 4 é menor do que 7, terá que pedir emprestado ao 1

```
    12
  7204
- 1636
------
     6
```

Ao emprestar, o 1 virou 0 e o 4, recebendo 8, virou 12. Então, 12 – 6 = 6

```
   8
  7104
- 1636
------
    56
```

Como 0 é menor do que 3, terá que pedir emprestado ao 2, que vira 1 e o 0, ao receber emprestado, vira 8. Finalmente, 8 - 3 = 5

```
   9
  6104
- 1636
------
  5356
```

Como 1 é menor do que 6, terá que pedir emprestado ao 7, que vira 6 e o 1, ao receber emprestado, vira 9. Finalmente, 9 - 6 = 3

Finalizando, 6 – 1 = 5

2.2. SUBTRAÇÃO EM HEXADECIMAL

Aqui executaremos o mesmo algoritmo, sempre lembrando que os valores entre 10 e 15 serão substituídos pelas letras e que, na regra do pedir emprestado, aquele que receber o empréstimo ganhará 16.

Vamos efetuar $(A741 - 8BED)_{16}$.

```
  A7 41
- 8BED
------
```

Como 1 é menor do que D (13), terá que pedir emprestado ao 4

```
   A73 17
- 8BED
------
      4
```

Ao emprestar, o 4 virou 3 e o 1, recebendo 16, virou 17. Então, 17 – 13 = 4

```
   A6 319
- 8BED
------
     56
```

Como 3 é menor do que E (14), terá que pedir emprestado ao 7, que vira 6 e o 3, ao receber emprestado, vira 19. Finalmente, 19 - 14 = 5

```
   96 3122
- 8BED
------
  1B 56
```

Como 6 é menor do que B (11), terá que pedir emprestado ao A, que vira 9 e o 6, ao receber emprestado, vira 22. Finalmente, 22 - 11 = 11 (B)

Finalizando, 9 – 8 = 1

3. MULTIPLICAÇÃO EM BINÁRIO

Aqui também vamos utilizar o mesmo algoritmo que usamos nas multiplicações em decimal.

Lembrando que 0 x 0 = 0, 0 x 1 = 0 e 1 x 1 = 1.

Como exemplo, vamos efetuar $(1101 \times 101)_2$.

```
   1101
x   101
------
   1101
  00000
+ 110100
--------
1000001
```

O primeiro passo é multiplicar cada algarismo do número de baixo pelo número de cima.

O segundo passo é somar as parcelas.

3.1. MULTIPLICAÇÃO EM OCTAL

Para facilitar essa operação, vamos utilizar uma tabela com uma “tabuada” em octal.

x	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7

X	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Com ajuda dessa tabela, vamos efetuar $(32 \times 45)_8$.

$$\begin{array}{r} {\scriptstyle 1}\\ 45\\ \times\ 32\\ \hline 2 \end{array}$$

Usando nossa tabela, veremos que 2 x 5 = 12. Assim colocamos o 2 embaixo e subimos o 1

$$\begin{array}{r} {\scriptstyle 1}\\ 45\\ \times\ 32\\ \hline 112 \end{array}$$

2 x 4 = 10; somando com o 1 que subiu, teremos 11.

$$\begin{array}{r} {\scriptstyle 1}\\ 45\\ \times\ 32\\ \hline 112\\ 157 \end{array}$$

Agora efetuamos a multiplicação por 3.

$$\begin{array}{r} 45\\ \times\ 32\\ \hline 112\\ +157\\ \hline 1702 \end{array}$$

Finalmente, efetuamos a soma.

3.2. MULTIPLICAÇÃO EM HEXADECIMAL

Da mesma forma que fizemos com a base octal, vamos usar uma tabela para nos auxiliar.

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	13	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	13	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B	0	B	16	21	2C	37	42	4D	58	63	6E	69	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Como exemplo, vamos efetuar $(3F \times D9)_{16}$.

```
  8
  D9
x 3F
 --
  7
```

Usando nossa tabela, veremos que F x 9 = 87. Assim colocamos o 7 embaixo e subimos o 8

```
  8
  D9
x 3F
 --
 CB7
```

F x D = C3; somando com o 8 que subiu, teremos CB.

```
  1
  D9
x 3F
 --
 CB7
28B
```

Agora efetuamos a multiplicação por 3.

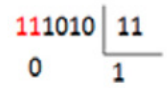
```
  D9
x 3F
 --
 CB7
+28B
 --
3567
```

Finalmente, efetuamos a soma.

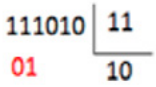
4. DIVISÃO EM BINÁRIO

A operação de divisão em binário segue o mesmo procedimento da divisão em binário, com a diferença de que quando o valor do dividendo é menor do que o divisor, o quociente recebe 0. Se o dividendo foi maior ou igual ao divisor, o quociente recebe 1.

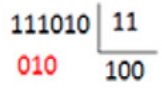
Para entender melhor, vamos a um exemplo, efetuando $(111010 \div 11)_2$.



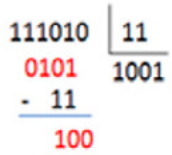
O dividendo em questão é igual ao divisor. Assim, colocamos 1 no quociente e 0 no resto.



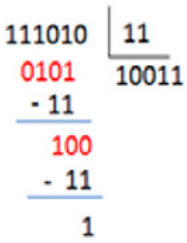
Agora o dividendo é menor do que o divisor. Então colocamos o 0 no quociente e "baixamos" mais um número.



Como o dividendo continua menor que o divisor, colocamos outro 0 e "baixamos" mais um número.



Agora o dividendo é maior do que o divisor. Então colocaos 1 no quociente, subtraímos o divisor do dividendo e "baixamos" o próximo número.



Da mesma forma, subtraímos o divisor do dividendo. Como não temos mais como "baixar" outro número, chegamos ao fim com a divisão tendo resto 1.

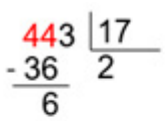
4.1. DIVISÃO EM OCTAL

Vamos agora efetuar a divisão na base octal. Como já mencionamos, o raciocínio é o mesmo das bases binária e decimal, considerando que não vamos usar os algarismos 8 e 9.

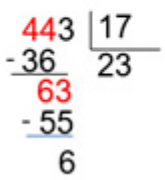
Para isso, vamos efetuar $(443 \div 17)_8$.

Para facilitar essa operação é bom colocarmos uma tabuada em octal do divisor (que é o 17), como a seguir:

17x1 = 17
17x2 = 36
17x3 = 55
17x4 = 74
17X5 = 113
17x6 = 132
17x7 = 151
17x10 = 170



O dividendo é maior do que o divisor. Então, pela tabuada, vemos a linha a ser usada é a segunda.



Agora baixamos o 3 e, com ajuda da tabuada, vemos que a linha a ser usada é a terceira.

Como não temos mais como dividir, o quociente é 23 e o resto da divisão é 6.

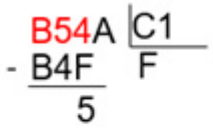
4.2. DIVISÃO EM HEXADECIMAL

Nesta última operação aritmética vamos repetir o processo usado na base octal e criar uma tabuada do divisor para facilitar.

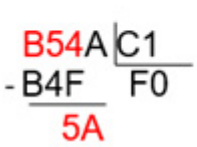
Como exemplo, vamos efetuar $(B54A \div C1)_{16}$.

Para facilitar essa operação é bom colocarmos uma tabuada em octal do divisor (que é o C1), como a seguir:

C1x1 = C1
C1x2 = 182
C1x3 = 243
C1x4 = 304
C1X5 = 3C5
C1x6 = 486
C1x7 = 547
C1x8 = 608
C1x9 = 6C9
C1xA = 78A
C1xB = 84B
C1xC = 90C
C1xD = 9CD
C1xE = A8E
C1xF= B4F
C1x10 = C10



O dividendo é maior do que o divisor. Então, pela tabuada, vemos a linha a ser usada é a décima quinta.



Agora baixamos o A e, como o novo dividendo é menor do que o divisor, colocamos 0 no quociente e o resto da divisão fica sendo 5A.

CONSIDERAÇÕES FINAIS

Nesta aula vimos como trabalhar as quatro operações aritméticas nas bases binária, octal e hexadecimal.

São algoritmos semelhantes usados nas três bases (assim como na base decimal) e, embora pareça complicado a princípio, vemos com a prática que é muito simples trabalhar essas operações.

O mais importante é que, como qualquer atividade ligada à matemática, é fundamental a prática e repetição de exercícios para consolidar o aprendizado.

Como sugestão, utilize a calculadora do Windows, na versão programador, e pratique bastante até consolidar seu conhecimento.

Na próxima aula veremos estudaremos as portas lógicas que compõem os circuitos existentes no computador.

Até lá!

MATERIAIS COMPLEMENTARES

Assista a esse vídeo bem didático com exemplos de multiplicação em octal. Link: https://www.youtube.com/watch?v=3kStGiYGmkc&t=44s.

Nesse material você poderá ler um pouco sobre a importância da base hexadecimal. Link: https://canaltech.com.br/produtos/O-que-e-sistema-hexadecimal/.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e organização de computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas operacionais modernos*. 3ª edição. Editora Pearson. Livro (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

AULA 4 – PORTAS LÓGICAS

OBJETIVO DA AULA

Reconhecer as portas lógicas, seu funcionamento e participação nos circuitos lógicos.

APRESENTAÇÃO

Os diversos circuitos que compõem nossos computadores são compostos pelas portas lógicas. Elas executam diversas operações e, combinadas, podem realizar tarefas mais complexas e muito importantes para nossas necessidades.

Nesta aula estudaremos as portas lógicas, que estão presentes em nosso cotidiano de forma muito mais intensa do que imaginamos.

Os circuitos digitais, existentes não só em nossos computadores, mas na maioria dos nossos eletrodomésticos e automóveis, por exemplo, são compostos de milhares de portas lógicas, que podem ser combinadas de infinitas maneiras para realizar praticamente qualquer tarefa que imaginarmos.

Vamos estudar, uma a uma, as portas lógicas, suas representações e tabelas-verdade, que descrevem seus comportamentos.

Vale destacar aqui que uma tabela verdade é uma estrutura que contém todas as combinações possíveis de entradas para uma porta lógica com suas respectivas saídas.

Basicamente temos seis tipos de portas lógicas: AND, OR, NOT, XOR, NAND e NOR.

As portas possuem entradas e produzem como saída valores lógicos (F e V), que também podem ser representados por 0 e 1, na forma binária.

Vamos estudar agora cada uma delas.

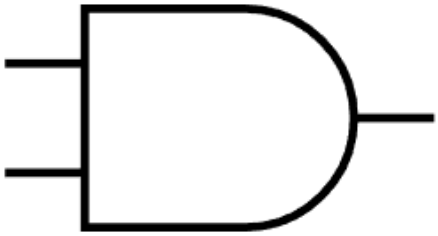
1. PORTA AND

Também conhecida em português como E, essa porta analisa as entradas, produzindo resultado 1 quando todas as entradas forem verdadeiras e 0 quando pelo menos uma das entradas for 0.

Em uma expressão lógica, a operação AND também pode ser representada por um ponto ou pelo símbolo “^”.

A Figura 1 mostra o desenho da porta AND.

FIGURA 1 | **Porta AND**



Elaborado pelo autor.

Teoricamente não há limites para a quantidade de entradas de uma porta AND. Mas, para uma porta com duas entradas como a da figura, teremos 4 combinações possíveis de entradas.

Considerando as entradas como A e B a tabela verdade da porta AND ficaria assim:

A	B	AND
1	1	1
1	0	0
0	1	0
0	0	0

Observe que somente na primeira linha tivemos o resultado 1. Nas outras três linhas, por haver pelo menos uma entrada com 0, o resultado foi 0.

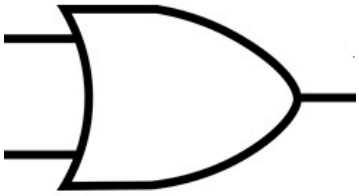
2. PORTA OR

Também conhecida em português como OU, essa porta analisa as entradas, produzindo resultado 0 quando todas as entradas forem 0 e 1 quando pelo menos uma das entradas for 1.

Em uma expressão lógica, a operação OR também pode ser representada por um sinal de “+” ou pelo símbolo “v”.

A Figura 2 mostra o desenho da porta OR.

FIGURA 2 | **Porta OR**



Elaborado pelo autor.

Teoricamente também não há limites para a quantidade de entradas de uma porta OR. Mas, para uma porta com duas entradas como a da figura, teremos 4 combinações possíveis de entradas.

Considerando as entradas como A e B, a tabela verdade da porta OR ficaria assim:

A	B	OR
1	1	1
1	0	1
0	1	1
0	0	0

Note que somente na última linha, onde todas as entradas são 0, o resultado foi 0. Em todas as outras o resultado foi 1.

3. PORTA NOT

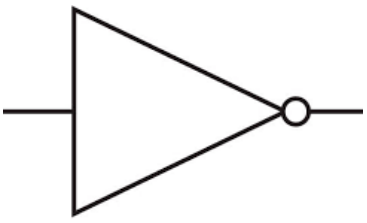
Essa porta tem a função de inverter o valor da entrada, ou seja, se entrar 1 sai 0 e se entrar 0 sai 1.

Como veremos na próxima aula, quando tratarmos de circuitos lógicos, a porta NOT tem muitas utilidades.

Essa operação é representada em expressões lógicas por um traço horizontal acima da variável ou por um til antes dessa variável.

A Figura 3 mostra a porta NOT.

FIGURA 3 | **A porta NOT**



Elaborado pelo autor.

A tabela verdade da porta NOT mostra seu funcionamento.

A	NOT
1	0
0	1

Como vimos, a porta NOT tem um comportamento muito simples, mas será muito útil.

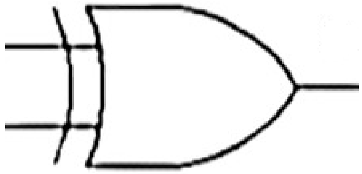
4. PORTA XOR

Também conhecida em português como OU EXCLUSIVO, essa porta analisa as entradas, a porta XOR admite exatamente duas entradas, produzindo resultado 0 quando as entradas forem iguais (1 1 ou 0 0) e 1 quando as entradas tiverem valores inversos (1 0 ou 0 1).

Em uma expressão lógica, a operação XOR também pode ser representada por um sinal de “⊕” ou pelo símbolo “$\underline{\vee}$”.

A Figura 4 mostra o desenho da porta XOR.

FIGURA 4 | **A porta XOR**



Elaborado pelo autor.

A tabela verdade do funcionamento da porta XOR é mostrada abaixo.

A	B	XOR
1	1	0
1	0	1
0	1	1
0	0	0

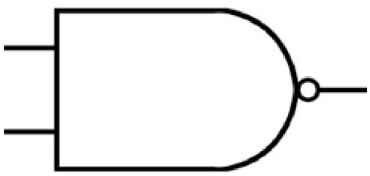
Note que, como mencionamos, o resultado da operação é V quando as entradas são diferentes e F quando as entradas são iguais.

5. PORTA NAND

Esta porta, chamada NOT AND, tem o comportamento exatamente inverso ao da porta AND, ou seja, mostra resultado 1, quando pelo menos uma das entradas tem valor 0, e resultado 0 quando todas as entradas têm valor 1.

A Figura 5 mostra a porta NAND.

FIGURA 5 | **A porta NAND**



Elaborado pelo autor.

Note que o desenho da porta NAND é muito semelhante ao da porta AND, com a diferença de que há um pequeno círculo na saída, indicando a negação.

Como é de se esperar, sua tabela verdade apresenta valores inversos ao da porta AND.

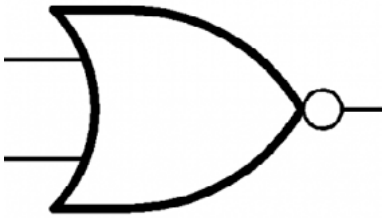
A	B	NAND
1	1	0
1	0	1
0	1	1
0	0	1

6. PORTA NOR

Esta porta, chamada NOT OR, tem o comportamento exatamente inverso ao da porta OR, ou seja, mostra resultado 0, quando pelo menos uma das entradas tem valor 1, e resultado 1 quando todas as entradas têm valor 0.

A Figura 6 mostra a porta NOR.

FIGURA 6 | **A porta NOR**



Elaborado pelo autor.

Assim como ocorre com a porta NAND, a porta NOR tem um desenho quase igual ao da porta OR, com a presença do círculo na saída, indicando a negação.

Da mesma forma, sua tabela verdade tem valores inversos aos da porta OR.

A	B	NOR
1	1	0
1	0	0
0	1	0
0	0	1

Não há nenhum mistério no funcionamento das portas lógicas e o entendimento desse funcionamento será muito importante para nosso próximo assunto: os circuitos digitais.

Os circuitos que vamos estudar na próxima aula são combinações de diversas portas lógicas, umas gerando entradas para outras.

CONSIDERAÇÕES FINAIS

Agora que finalizamos o estudo das portas lógicas, ficará muito mais fácil entender o funcionamento dos circuitos digitais que compõem nossos equipamentos.

Vimos as seis portas utilizadas na confecção desses circuitos e, conhecendo bem o funcionamento delas através de suas tabelas-verdade, poderemos interpretar e avaliar o comportamento de circuitos importantes tais como o decodificador, o somador e o multiplexador.

Certifique se de que você entendeu esse funcionamento tentando refazer suas tabelas-verdade e experimentando algumas variações de entradas e as saídas produzidas.

MATERIAIS COMPLEMENTARES

Assista a esse vídeo simulando o funcionamento das portas lógicas. É ótimo para ver na prática o que acabamos de estudar. Link: https://www.youtube.com/watch?v=oJsJpezNz5w.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e organização de computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas operacionais modernos*. 3ª edição. Editora Pearson. Livro (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.