Matemática Aplicada

Aula 06 - Números Binários

Apresentação

A nossa sociedade está habituada a pensar sobre os números na chamada base dez, que corresponde ao número de dedos de nossas mãos. Na representação em base dez, ou sistema decimal, temos dez símbolos diferentes para representar valores, como veremos nesta aula, e construímos os números que desejamos a partir desses dez símbolos. Por questões tecnológicas, no entanto, os computadores não funcionam bem em base dez e optou-se por projetá-los para que façam seus cálculos em base dois, também chamado de sistema binário. Veremos, nesta aula, um pouco desse sistema, mas antes sugiro que assistam esse vídeo do programa português Isto é Matemática, que fala um pouco dos sistemas de numeração e as motivações para usá-los.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Pensar na representação usual de números como uma notação posicional (em base dez).
- Obter o valor de um número inteiro, dada a sua representação em uma base qualquer.
- Obter a representação binária de um número inteiro.
- Realizar operações aritméticas com números binários.

Notação Posicional

Nós usamos todos os dias, sem perceber, a chamada **notação posicional** para representar números. Esse é um assunto que você com certeza vem estudando desde o início do ensino fundamental, quando vemos que existem maneiras de representar números que não são posicionais, como a usada pelos romanos (I, II, III, IV etc.), e posicionais, como a que herdamos dos árabes, mas que se iniciou com os hindus. Por essa razão, os algarismos que usamos no dia a dia, 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, são chamados de algarismos indo-arábicos.

Notação Posicional Decimal (ou em base dez)

A notação numérica posicional decimal que usamos em nosso dia a dia usa sequências de algarismos para representar valores, e o valor total representado é dado pela soma do valor representado por cada algarismo. Por exemplo, quando escrevemos 13.936, estamos representando o número composto por:

1 dezena de milhar + 3 milhares + 9 centenas + 3 dezenas + 6 unidades

Como sabemos que temos uma dezena de milhar e três dezenas e não o contrário, por exemplo? Pela posição onde o 1 e o 3 aparecem respectivamente. A posição mais à direita da representação corresponde às unidades, a segunda (da direita para a esquerda), às dezenas, e assim por diante. É por isso que a notação é chamada de posicional: o valor representado por cada algarismo depende da posição em que aquele algarismo aparece na representação.

Agora, vamos substituir os nomes correspondentes a cada posição pelo seu significado em potências de 10. O número 13.936 é composto por:

$$1 \cdot 10^4 + 3 \cdot 10^3 + 9 \cdot 10^2 + 3 \cdot 10^1 + 6 \cdot 10^0$$

Vendo dessa maneira, dizemos que cada algarismo da representação posicional é o **coeficiente** a ser multiplicado pela potência da **base** correspondente àquela posição. A base usada no sistema decimal é a base dez, então, cada posição corresponde a uma potência de dez, variando de zero a $n - 1$, onde n é a quantidade de algarismos da representação. No nosso exemplo, $n = 5$.

Uma Definição mais Geral

A notação numérica posicional requer, de maneira geral, a definição de dois objetos:

- Uma base B (que precisa ser no mínimo igual a 2); e
- Um conjunto de B símbolos correspondentes a cada valor inteiro no intervalo que vai de 0 a $(B - 1)$.

A base utilizada em nossa sociedade é o valor 10, e o conjunto de símbolos é composto pelos algarismos de 0 a 9.

Um número s qualquer na notação numérica posicional é, por definição, representado por uma sequência de símbolos do conjunto citado acima. Como estamos tratando de um sistema posicional, associamos um índice à posição de cada símbolo. Se tivermos, por exemplo,

$s = abc$ (onde a, b e c são elementos do conjunto de símbolos),

escrevemos

$$s = abc$$

Essa sequência representa, em base B, o número:

$$s = a \cdot B^2 + b \cdot B^1 + c \cdot B^0$$

Podemos descrever s de maneira ainda mais geral, chamando cada um dos símbolos da sequência que o representa na base B de b_i, onde i é sua posição dentro da representação. Temos então que, se cada b_i pertence ao conjunto de símbolos que representam a base B,

$$s = b_n b_{n-1} \ldots b_2 b_1 b_0$$

representa o número

$$\sum_{i=0}^{n} b_i \cdot b^i = b_0 B^0 + b_1 B^1 + \ldots + b_{n-1} B^{n-1} + b_n B^n$$

$$\sum_{i=0}^{n} b_i \cdot b^i = b_n B^n + b_{n-1} B^{n-1} + \ldots + b_1 B^1 b_0 B^0$$

Se liga!

A representação $s = b_n b_{n-1} \ldots b_2 b_1 b_0$ NÃO É UM PRODUTO! Os b_i's são os símbolos da base b_i que representam o número s. O número 2343 na base $B = 10$, por exemplo, tem o $b_0 = 3$, $b_1 = 4$, $b_2 = 3$ e $b_3 = 2$.

As operações da definição acima (o somatório) são realizadas em base dez. Caso o símbolo b_i não corresponda diretamente a um algarismo em base dez, ele precisa ser convertido para o seu valor na base dez antes da realização da operação de multiplicação $b_i \cdot B_i$. Usualmente, quando trabalhamos com bases menores do que dez, utilizamos como símbolos o subconjunto dos algarismos de 0 até $B - 1$. Para bases maiores, costuma-se utilizar as letras do alfabeto A, B etc., respectivamente para os valores 10, 11 e assim por diante.Na base hexadecimal, por exemplo, os símbolos utilizados são 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F e C seria equivalente a 12 na base 10.

Bases Usuais em Computação

Em computação, usamos muito a base dez, como todo mundo, mas também usamos outras: a base 2 (**sistema binário**), pois é assim que números são representados na memória e no processamento do computador, e a base 16 (**sistema hexadecimal**), que é muito prática para apresentar uma versão mais curtinha dos números. Só para dar uma ideia, o número 15 pode ser representado em binário como

$$1111$$

e em hexadecimal como

$$F$$

Bem mais curtinho, não? Na prática, números são representados em binário dentro do computador, mas, quando é para a leitura por uma pessoa, usa-se decimal ou hexadecimal.

Nesta aula, vamos nos concentrar nos binários.

Convenção: para indicar que uma sequência de algarismos deve ser interpretada como uma representação binária, e não decimal, como seria o usual, podemos usar a convenção de usar um índice 2 ao final da representação ou os caracteres $0B$ no seu início, como no exemplo a seguir:

$$10111_2 \text{ ou } 0B10111$$

Esta segunda é usada especialmente em linguagens de programação, pois não precisa de recursos de formatação, como é o caso do índice.

Sistema de Numeração Binário (ou em Base 2)

O sistema de numeração binário usa a base 2 para representar números. O conjunto de símbolos válidos em uma representação binária é então $\{0, 1\}$, pois vão de 0 ao valor da base menos 1.

Os símbolos 0 e 1 na representação binária são chamados de **bits**. Como agora trabalhamos com a base 2, cada posição em uma representação corresponde a uma potência de 2. Da direita para a esquerda, temos, respectivamente: $2^0, 2^1, 2^2, 2^3, 2^4, 2^5$, etc.

Um número binário é então uma sequência de bits, na qual cada bit é um coeficiente a ser multiplicado pela potência de 2 correspondente, como na definição da notação posicional. Por exemplo, a sequência binária

$$10011101_2$$

representa o número

$$1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$$
$$2^7 + 2^4 + 2^3 + 2^2 + 2^0 =$$
$$128 + 16 + 8 + 4 + 1 = 157$$

,

pois temos a seguinte distribuição dos bits:

Bits	1	0	0	1	1	1	0	1
Posição	7	6	5	4	3	2	1	0
Potência	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valor	128	0	0	16	8	4	0	1

Vídeo 02 - Conversão Binário em Decimal

Atividade 01

1. Converta os seguintes números binários para decimal.

a. 11011_2

b. 10101_2

c. 1001_2

d. 111100_2

Vídeo 03 - Atividade 01

Dessa forma, sabemos traduzir uma representação binária em decimal, ou seja, conseguimos traduzir uma representação que o computador consegue entender para uma que nós conseguimos entender. Mas, quando estamos programando, podemos precisar fazer o contrário. Como então obter a representação binária de um número?

Nesse caso, precisamos usar de novo a definição da notação posicional, mas de maneira um pouco diferente. Para obter a representação binária de um número, que vamos chamar de x, devemos executar uma sequência de passos, que vão transformar nosso número na representação decimal como uma soma de potências de base 2, para podermos obter a representação binária do número. Vamos aos passos:

1. Encontrar a maior potência de 2 menor ou igual a x, ou seja, encontrar n, tal que $2^n \leq x \leq 2^{n+1}$. A representação binária de x terá então $n + 1$ bits (posições de zero a n), com o bit n igual a 1.

2. Subtrair esse valor do valor a ser representado (x). O resultado da subtração é um novo número a ser convertido.

    a. Se esse novo número é maior ou igual a 2, retornar ao passo 1.

    b. Caso contrário (o resultado da subtração é 0 ou 1), esse resultado é o coeficiente de posição zero.

3. Montar a representação binária com os coeficientes identificados no processo em suas respectivas posições, completando eventuais coeficientes ausentes com zeros.

Exemplo

Se $x = 324$, a maior potência de 2 menor ou igual a 324 é $2^8 = 256$. Isso significa que a representação binária de 324 terá 9 bits, indo do bit zero ao bit 8, e que na posição 8 teremos um 1.

Posição	8	7	6	5	4	3	2	1	0
bit	1	?	?	?	?	?	?	?	?

Fazendo agora o passo 2, temos que $324 - 256 = 68$. Como 68 é maior do que 2, a regra 2.a. deve ser aplicada, ou seja, voltamos ao passo 1. A maior potência de 2 menor ou igual a 68 é $26 = 64$. A representação binária de 68 terá então um bit 1 na posição 6.

Posição	8	7	6	5	4	3	2	1	0
bit	1	?	1	?	?	?	?	?	?

Fazendo de novo o passo 2, temos que $68 - 64 = 4$. Como 4 é maior do que 2, aplicamos novamente a regra 2.a. que nos diz para voltarmos ao passo 1. A maior potência de 2 menor ou igual a 4 é $22 = 4$. A representação binária de 4 terá então um bit 1 na posição 2.

Posição	8	7	6	5	4	3	2	1	0
bit	1	?	1	?	?	?	1	?	?

Fazendo mais uma vez o passo 2, temos que $4 - 4 = 0 < 2$. Aplicamos, então, a regra 2.b, que diz que a posição zero deve ser preenchida com esse coeficiente, ou seja, 0. Em seguida, passamos ao passo 3, que diz que todas as posições intermediárias também devem ser preenchidas com zeros. Temos então como resultado final:

Posição	8	7	6	5	4	3	2	1	0
bit	1	0	1	0	0	0	1	0	0

ou 101000100_2.

Adicional: Seguem vídeos mostrando o método das divisões sucessivas.

Vídeo 04 - Conversão decimal para binário

Vídeo 05 - Exemplo

Atividade 02

1. Verifique que o resultado obtido no exemplo anterior está correto convertendo 101000100_2 para a base dez.

Operações Aritméticas em Base 2

O procedimento genérico para realizar operações em diferentes bases é converter os números para decimal, realizar a operação em base dez e reconverter o resultado para a base original. Mas essa solução nem sempre é a melhor. No caso particular de números binários, não é difícil fazer as contas sem passar pela base dez. Vejamos como.

Adição Binária

Como você faz a soma de valores em base dez? Começa da direita para a esquerda e soma os algarismos de mesma posição, não é? Para ajudar, colocamos os dois números a serem somados um sobre o outro, alinhados à direita, para ficar fácil de identificar as posições correspondentes.

Quando a soma dos algarismos é menor do que dez, simplesmente colocamos o resultado na coluna correspondente e, quando é maior ou igual a dez, temos o chamado "**vai um**"(ou “sobe um”). Ficam então na coluna sendo somadas apenas as unidades que ultrapassam dez, e dez unidades são passadas para a coluna seguinte, sob a forma de "vai um".

$$\begin{array}{r} {\scriptstyle 1\,1} \\ 127 \\ +\ 86 \\ \hline 213 \end{array}$$

A adição binária deve ser feita bit a bit, exatamente como fazemos com a adição em base dez, somando os algarismos de cada posição. Então, começando da direita para a esquerda, ou seja, partindo do bit de posição zero, somamos os bits de cada coluna, e se o resultado for maior ou igual a 2, temos um vai um para a coluna seguinte. Como são poucas as opções, é fácil enumerá-las:

- $0 + 0 = 0$ (não há "vai um")
- $0 + 1 = 1 + 0 = 1$ (não há "vai um")
- $1 + 1 = 2$, que é 10_2, em binário. Logo, na coluna sendo somada fica o 0, e o 1 vai para a coluna seguinte, ou seja, vai um.

Exemplo:

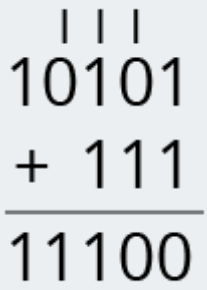
Conta Finalizada	Conta passo a passo
$\begin{array}{r} {\scriptstyle 1\,1\ \ 1\,1} \\ 11011_2 \\ +\ 1001_2 \\ \hline 100100_2 \end{array}$	$\begin{array}{r} 11011_2 \\ +\ 1001_2 \\ \hline \end{array}$

Subtração Binária

Se Liga!

Quando há um "vai um" da coluna i, há um "vem um" na coluna $i+1$. Isso significa que é possível ser necessário fazer a soma de 3 bits de uma vez; os dois que fazem parte dos números sendo somados e o "vem um" (da coluna anterior). Nesse caso, o resultado pode vir a ser $3 = 1 + 1 + 1 = 11_2$, o que resulta em 1 na coluna sendo somada $(i+1)$ e vai um para a coluna seguinte $(i+2)$.

Exemplo:



A subtração binária deve ser feita bit a bit, exatamente como fazemos com a subtração em base dez, subtraindo os algarismos de cada posição. Então, começando da direita para a esquerda, ou seja, partindo do bit de posição zero, subtraímos os bits de cada coluna.

Quando cada coeficiente do minuendo (se você está resolvendo $a - b$, então o a é o minuendo e o b é o subtraendo) em cada posição i é maior ou igual ao coeficiente na mesma posição i do subtraendo, não há necessidade de “pedir ajuda" (empréstimo) ao coeficiente na posição $i+1$, e a subtração se faz pela subtração simples de cada coluna.

Exemplo:

$$\begin{array}{r} 1110 \\ -1010 \\ \hline 0100 \end{array}$$

Quando um coeficiente do minuendo é menor do que o coeficiente na mesma posição i do subtraendo, é necessário o empréstimo, ou seja, o coeficiente da posição $i+1$ é diminuído de 1 e o valor da base é adicionado ao coeficiente da posição i, passando a ser possível fazer a subtração dessa coluna.

Como são poucas as opções, é fácil enumerá-las:

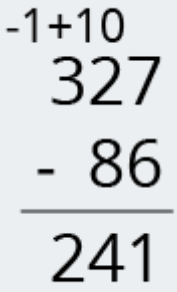
- $0 - 0 = 1 - 1 = 0$
- $1 - 0 = 1$
- $0 - 1$ precisa pedir emprestado à coluna seguinte, ficando $2 - 1$, ou $10_2 - 1 = 1$

Se Liga!

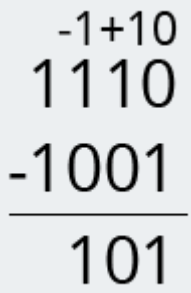
Da mesma maneira que acontece na base dez (mas que nem percebemos mais, de tal forma estamos habituados), quando é feito um empréstimo, é subtraído 1 da posição $i+1$ do subtraendo e somado 10 ao símbolo correspondente na posição i. Se estamos na base dez, 10 é dez, se estamos na base 2, 10_2 é 2...

Exemplo:

- Base 10



- Base 2



Atividade 03

1. Realize as operações a seguir em binário e confira os resultados convertendo os operandos para a base dez, realizando as operações em base dez e reconvertendo para binário.

a. $100100_2 + 1010_2 =$

b. $100100_2 - 1010_2 =$

Multiplicação Binária

A multiplicação binária também segue os mesmos procedimentos que a multiplicação decimal. Colocamos os dois operandos um acima do outro, alinhados à direita, e realizamos a multiplicação de cada bit do segundo operando por cada bit do primeiro. Cada bit do segundo operando dá origem a uma linha em uma adição binária. Como essa linha corresponde ao produto do bit atual pelo primeiro operando, teremos linhas compostas apenas de zeros (se o bit atual é 0) e linhas em que o primeiro operando é repetido (se o bit atual é 1). Cada nova linha é registrada abaixo da anterior com um deslocamento de uma casa para a esquerda em relação a ela.

Vejamos como isso ocorre através de um exemplo. Vamos multiplicar 11000_2 x 10011_2. Começamos posicionando os operandos:

$$\begin{array}{r} 11000_2 \\ \times\ 10011_2 \\ \hline \end{array}$$

Agora, iniciamos o processo a partir do bit b_0 do segundo operando, ou seja, 1. Ficamos então com:

$$\begin{array}{r} 11000_2 \\ \times\ 10011_2 \\ \hline 11000_2 \end{array}$$

Continuando, pegamos agora o segundo bit de 10011, ou seja, 1 de novo. O resultado da multiplicação será mais uma vez o primeiro operando, mas agora deslocado de uma posição para a esquerda.

$$
\begin{array}{r}
11000_{2} \\
\times\ 10011_{2} \\
\hline
11000_{2} \\
11000_{2} \\
\end{array}
$$

Em seguida, temos duas linhas de zeros e, finalmente, uma última repetição do primeiro operando.

$$
\begin{array}{r}
11000_{2} \\
\times\ 10011_{2} \\
\hline
11000_{2} \\
11000_{2} \\
00000_{2} \\
00000_{2} \\
+11000_{2} \\
\end{array}
$$

Finalmente, realizamos a soma de todas as parcelas obtidas (as posições vazias correspondem a zeros na soma):

$$
\begin{array}{r}
11000_2 \\
\times\ 10011_2 \\
\hline
11000_2 \\
11000_2 \\
00000_2 \\
00000_2 \\
+11000_2 \\
\hline
111001000_2
\end{array}
$$

Divisão Binária

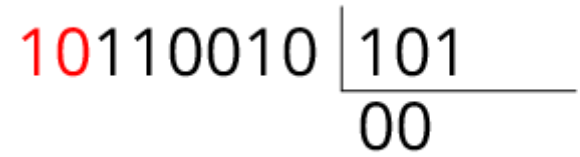
A divisão binária também segue os mesmos procedimentos que a divisão decimal, com a diferença que cada bit do quociente é 0 (o dividendo parcial era menor do que o divisor) ou 1 (o dividendo parcial era maior ou igual ao divisor). Vejamos o algoritmo através de um exemplo.

Vamos realizar a divisão de 10110010_2 por 101_2.

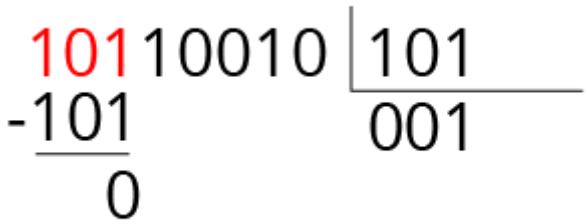
$$
\begin{array}{r|l}
10110010 & 101 \\
\hline
\end{array}
$$

Inicialmente, selecionamos (diz-se “baixar”) o bit mais significativo $(b_7 = 1)$ e procuramos dividi-lo pelo divisor. Chamaremos o valor que estamos tentando dividir em um dado momento de dividendo parcial. Como o divisor é maior do que o dividendo parcial, o quociente correspondente é 0 (1_2 dividido por 101_2 é zero e resto 1_2). Nas figuras a seguir, mostramos o dividendo parcial em vermelho e sempre que o resto é o próprio dividendo parcial, como nesse caso, simplesmente selecionamos o próximo bit.

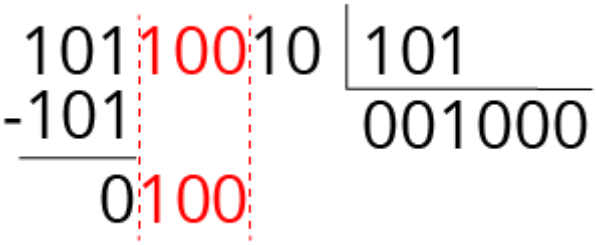
Agora, aumentamos o dividendo parcial, baixando o próximo bit e o concatenando à direita do resto desse passo, que no caso era o próprio dividendo parcial. Temos agora que dividir 10_2 por 101_2, que mais uma vez dá 0, com resto agora de 10_2.



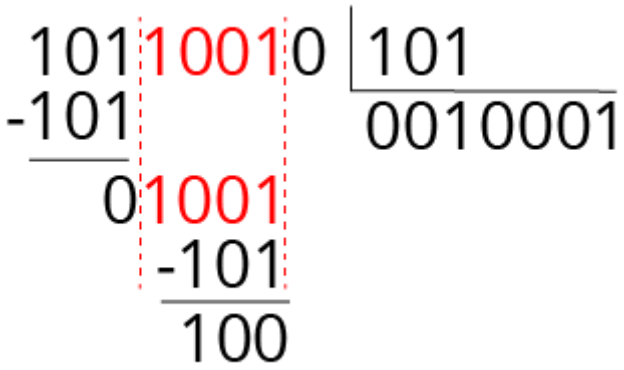
No próximo passo, tentamos dividir 101_2 por 101_2 e, pela primeira vez, obtemos um quociente de 1_2 (inserido à direita dos dois zeros que havíamos obtido anteriormente) e um resto 0.



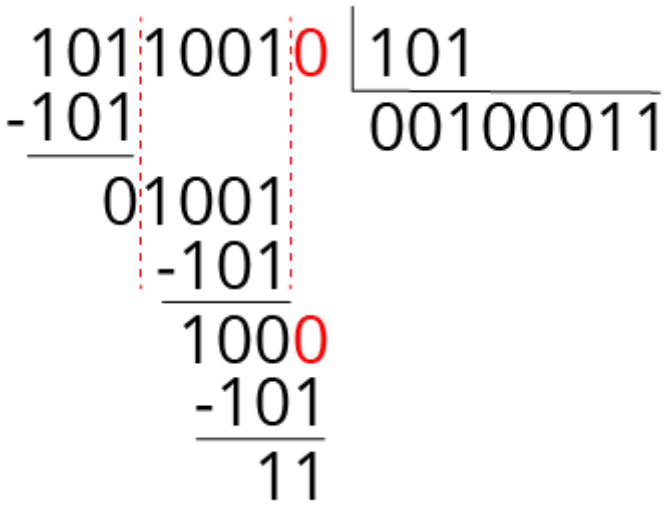
Continuamos o processo baixando o próximo bit $(b_4 = 1)$ para realizar nova divisão. Mas 1 é de novo menor do que o divisor, dando um quociente 0 e resto 1. Precisaremos baixar ainda b_3, b_2 e b_1 para finalmente obtermos um dividendo parcial maior do que o divisor.



Cada uma dessas tentativas até este momento gera um novo 0 no quociente, que a essa altura já é 001000. A divisão de 1001 por 101 dá 1 e resto $1001 - 101 = 100$.



Baixando o último bit ($b_0 = 0$), ficamos agora com a divisão de 1000 por 101, que dá 1 e resto $1000 - 101 = 11$.



Como não há mais nenhum bit para baixar, temos o resultado final, que é 100011 e resto 11.

Se Liga!

Observe que os zeros à esquerda do número não contribuem para o seu valor. Por isso, podemos omiti-los, escrevendo 100011 ao invés de 00100011. Adicionalmente pela mesma razão, em geral, encurtamos o algoritmo, iniciando o processo de divisão com o primeiro dividendo parcial maior ou igual ao divisor. Mas, atenção! Isso só pode ser feito no início da divisão, com os bits mais significativos. Quando estamos no meio, não podemos esquecer os zeros a serem acrescentados no resultado a cada vez que um dividendo parcial é menor do que o divisor.

Vídeo 06 - Operações com Binários

Atividade 04

1. Realize as operações a seguir em binário e confira os resultados convertendo os operandos para a base dez, realizando as operações em base dez e reconvertendo para binário:

    a. $100100_2 \cdot 1010_2 =$

    b. $100100_2 \div 1010_2 =$

Se Liga!

Vimos aqui como representar números inteiros em notação binária e como realizar operações com eles. Isso na realidade é apenas uma parte de tudo o que precisamos saber para entender o funcionamento interno da matemática dos computadores, mas é um primeiro passo. Para ir mais longe, precisamos também saber como representar números racionais (números não inteiros, com vírgula) e lidar com o fato de que nos computadores há um número fixo de bits disponível para representar os números.

Quanto ao primeiro ponto, vamos ver um exemplo:

$$3,12$$

Isso significa que temos 3 unidades (3.10^0), 1 décimo (1.10^{-1}) e 2 centésimos (2.10^{-2}). A mesma coisa acontecerá em binário, com a primeira posição após a vírgula correspondendo a 2^{-1}, a segunda a 2^{-2} e assim por diante.

Quanto ao fato de que o tamanho da representação é fixo e finito, isso pode causar os chamados erros de *overflow*, que acontecem quando o resultado de uma operação necessita de mais bits para ser representado do que o disponível. Por exemplo, se temos 8 bits para representar os números positivos, o maior número que podemos representar é

$$11111111_2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 128 + 64$$

Além disso, ainda temos o problema de como representar os números negativos. Em geral, um bit da representação será usado para indicar se o número é positivo ou negativo e diferentes convenções podem ser usadas para a determinação do valor do número representado. As mais comuns são chamadas de sinal-magnitude e complemento a 2. Não vamos estudá-las nesta aula, mas sugerimos como referência, para quem se interessar, os livros que apresentam o hardware dos computadores.

Leitura Complementar

- IFRAH, Georges. **Os números**: a história de uma grande invenção. 3. ed. Rio de Janeiro: Editora Globo, 1985.

  Este livro conta a história da criação dos números, apresentando diversos sistemas de numeração sob um ponto de vista bastante prático. É uma leitura agradável e muito informativa.

- MENDES, Iran Abreu. **Números**: o simbólico e o racional na história. São Paulo: Editora Livraria da Física, 2006.

  Este outro livro apresenta uma visão um pouco mais aprofundada dos números, com uma visão mais filosófica. É muito recomendado para quem gosta de pensar o porquê das coisas serem como são.

- TANENBAUM, Andrew S. **Organização estruturada de computadores**. 5. ed. São Paulo: Editora Prentice Hall do Brasil, 2007.

  Aqui você encontra algo mais sobre o sistema de numeração binário e o seu uso no hardware dos computadores.

Resumo

Nesta aula, você revisou a chamada notação posicional, que usamos em nosso dia a dia para representar números, e viu como os mesmos princípios podem ser usados para contagem em bases diferentes da base dez, a qual é mais utilizada. Em particular, estudamos a representação binária, que é a representação usada internamente pelos computadores, e vimos como realizar operações aritméticas nessa representação, entendendo um pouco mais sobre o que se passa dentro dos computadores.

Autoavaliação

1. Forneça a representação em binário dos seguintes números:
    a. 3
    b. 15
    c. 758
    d. 1123
    e. 1024
2. Que números (no sistema decimal) correspondem aos números binários a seguir?
    a. 1010_2
    b. 101010_2
    c. 11110000_2
    d. 100000000_2
    e. 00111_2
3. Realize os seguintes cálculos no sistema binário:
    a. $1110_2 + 10000_2$
    b. $1110_2 + 1111_2$

c. $1111_2 - 1010_2$

d. $1111000_2 - 1000_2$

e. $11100_2 \cdot 11100_2$

f. $10_2 \cdot 101_2$

g. $10_2 \div 101_2$

h. $11100_2 \div 11_2$

Referências

GROSSMAN, Peter. **Discrete mathematics for computing**. 2. ed. New York: Editora Palgrave Macmillan, 2002.

TANENBAUM, Andrew S. **Organização estruturada de computadores**. 5. ed. São Paulo: Editora Prentice Hall do Brasil, 2007.