

Infraestrutura de Hardware

**Juliana Regueira Basto Diniz
Abner Corrêa Barros**

Volume 1

Recife, 2009

Universidade Federal Rural de Pernambuco



Reitor: Prof. Valmar Corrêa de Andrade

Vice-Reitor: Prof. Reginaldo Barros

Pró-Reitor de Administração: Prof. Francisco Fernando Ramos Carvalho

Pró-Reitor de Extensão: Prof. Paulo Donizeti Siepierski

Pró-Reitor de Pesquisa e Pós-Graduação: Prof. Fernando José Freire

Pró-Reitor de Planejamento: Prof. Rinaldo Luiz Caraciolo Ferreira

Pró-Reitora de Ensino de Graduação: Profª. Maria José de Sena

Coordenação Geral de Ensino a Distância: Profª Marizete Silva Santos

Produção Gráfica e Editorial

Capa e Editoração: Allyson Vila Nova, Rafael Lira, Italo Amorim, Arlinda Torres e Heitor Barbosa

Revisão Ortográfica: Marcelo Melo

Ilustrações: Allyson Vila Nova, Diego Almeida e Moisés de Souza

Coordenação de Produção: Marizete Silva Santos

Sumário

Conhecendo o Volume 1	4
Apresentação	5
Capítulo 1 – Modelo de um Sistema Computacional.....	7
Capítulo 2 – Conhecendo Melhor as Operações Aritméticas	22
Capítulo 3 – Lógica Digital	75
Considerações Finais	94
Conheça os Autores	96

Conhecendo o Volume 1

Neste primeiro volume, você irá encontrar os conteúdos referentes ao Módulo 1 da disciplina **Infraestrutura de Hardware**. Para facilitar seus estudos, veja os conteúdos encontrados neste primeiro volume.

Módulo 1 - Introdução aos Sistemas Computacionais

Carga horária: 15 h

Objetivo Principal do Módulo 1:

Apresentar um sistema computacional, separando todos os seus subsistemas internos que serão estudados nos próximos módulos. Além disso, apresenta um breve histórico da evolução dos computadores e os conceitos associados à aritmética computacional e à operação dos circuitos lógicos digitais.

Conteúdo Programático do Módulo 1

- » Modelo de um sistema de computação.
- » Evolução dos Computadores
- » Operações Aritméticas
- » Circuitos Lógicos

Apresentação

Caro(a) cursista,

Estamos, neste momento, iniciando uma viagem para conhecer a Organização e a Arquitetura interna dos computadores, ou seja, conheceremos os principais componentes eletrônicos que formam um computador, os circuitos integrados, e como estes componentes se relacionam entre si. Todos os assuntos abordados neste livro estarão, de certa forma, relacionados à estrutura e ao funcionamento de computadores, sobretudo do ponto de vista de Hardware. Por meio desta disciplina, apresentaremos a natureza e as características dos sistemas de computação modernos.

Nosso estudo será uma tarefa interessante e desafiadora, principalmente porque contamos hoje não apenas com uma grande variedade de computadores, mas com uma grande variedade de equipamentos eletrônicos que executam algum tipo de computação. Essa variedade de produtos difere em custo, tamanho, desempenho e tipo de aplicação a ser utilizado. Além disso, observamos uma rápida evolução na tecnologia, que se estende desde circuitos integrados até a combinação desses componentes.

Mesmo diante de uma rápida evolução, certos conceitos aplicam-se a qualquer projeto de computadores. O objetivo desta disciplina é oferecer uma discussão sobre esses conceitos, relacionando-os com as questões de projetos modernos.

Bons estudos!

Juliana Regueira Basto Diniz

Abner Correa Barros

Professores Autores



Capítulo 1

O que vamos estudar?

Neste capítulo, vamos estudar os seguintes temas:

- » Funções do Computador
- » Componentes do Computador
- » Evolução histórica dos Computadores

Metas

Após o estudo deste capítulo, esperamos que você consiga:

- » Compreender que funções básicas são desempenhadas pelos computadores;
- » Identificar que componentes e subsistemas são necessários na organização interna dos computadores para que os mesmos possam desempenhar as suas funções básicas;
- » Conhecer a evolução histórica da informática, desde o primeiro computador eletrônico até as máquinas mais modernas da atualidade.

Capítulo 1 – Modelo de um Sistema Computacional



Vamos conversar sobre o assunto?

Estamos, neste momento, iniciando o estudo de uma das áreas mais importantes da computação, a infraestrutura de hardware. Sem a evolução do hardware e dos seus componentes eletrônicos, toda a disseminação da informática, como vemos hoje, não seria possível. Talvez até este momento você nunca tenha parado para pensar o quanto somos dependentes desta tecnologia. Você consegue imaginar a sua vida, nos dias atuais, sem o uso dos computadores? Imagine como seria este nosso curso sem o uso dos computadores. Seria, no máximo, um curso por correspondência acompanhado por algum tipo de programa televisivo, como acontecia com os cursos a distância no passado, ou seja, não seria possível a sua interação com professores nem colegas.

Todos somos testemunhas do quão rápido a informática evoluiu e passou a fazer parte da nossa vida nestes últimos anos. Em praticamente todos os recursos da vida moderna existe algum sistema computadorizado. Toda esta evolução, entretanto, não seria possível sem os avanços formidáveis ocorridos na arquitetura de hardware dos computadores em geral.

Mas, por falar em evolução das arquiteturas, você sabe o que vem a ser e como é constituída a arquitetura de hardware de um computador?

A arquitetura de computadores é a área da ciência que se preocupa em estudar os atributos dos Sistemas Computacionais que são visíveis aos profissionais que irão programar estes equipamentos, ou seja, a estrutura e o comportamento funcional da máquina. Esses atributos, conforme veremos mais a frente, têm impacto direto sobre a lógica de construção e execução dos programas.

Em paralelo à arquitetura de computadores, estudaremos também a organização dos sistemas computacionais. A organização dos computadores refere-se ao estudo dos aspectos não visíveis ao

programador, ou seja, assuntos referentes à organização dos fluxos de dados, projeto de controle lógico e a implementação física, que irão atender às especificações da arquitetura.

Em termos práticos, são considerados aspectos da arquitetura dos computadores o conjunto de instruções de uma máquina, o número de bits do processador, os mecanismos associados aos periféricos e as técnicas de endereçamento da memória. São aspectos da organização de computadores os Sinais de controle, a interface computador/periférico, a tecnologia de memória utilizada, dentre outros.

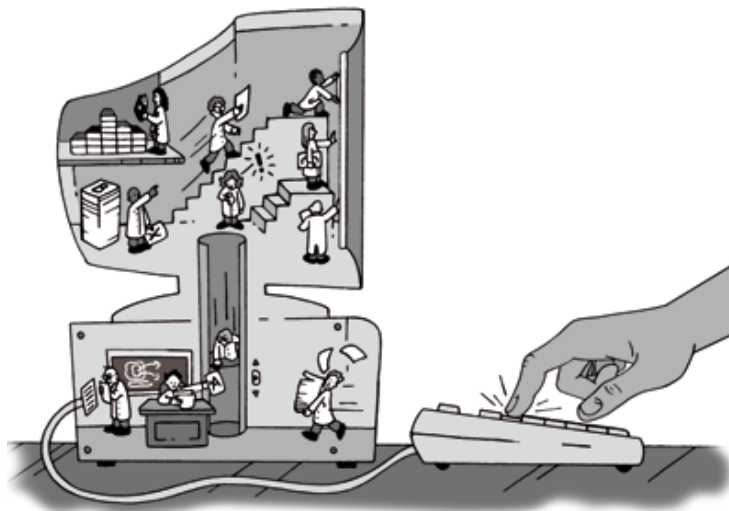


Figura 1 – Como um computador trabalha?

Em síntese, muitos fabricantes oferecem uma família de modelos de computadores, todos com a mesma arquitetura, mas com diferenças na organização, com preços e características de desempenho distintas. Tomemos como exemplo o Sistema 370 da IBM, uma arquitetura introduzida em 1970, com grande número de modelos. Um cliente modesto poderia comprar um modelo inferior e, caso sua demanda aumentasse, poderia migrar para um modelo superior. Como todos os computadores desta família adotavam uma mesma arquitetura de hardware, uma eventual troca de um computador por um outro maior não o obrigaria a abandonar as aplicações que já tivessem sido desenvolvidas para o seu primeiro computador. Ao longo dos anos, a IBM continuou introduzindo modelos novos, mantendo a mesma arquitetura, preservando assim o investimento em software dos clientes.

Um outro exemplo, mais próximo da realidade de cada um de nós, são os nossos computadores pessoais, os famosos PCs. Você

já percebeu que com o passar dos anos foi mantida uma certa compatibilidade entre as diversas gerações deste tipo de computador? Esta compatibilidade se deve ao fato de todos adotarem a arquitetura Intel X86. É por esta razão que mesmo aquele seu programa antigo, de dez ou quinze anos atrás, ainda poder ser executado sem muitos transtornos nos computadores atuais. Como você pode perceber, manter uma mesma arquitetura entre diversas gerações de uma família de computadores permite manter a compatibilidade de código.

A organização interna de um computador, por outro lado, muda de uma máquina para outra dentro da mesma família. É ela que em grande parte define as características de tamanho, desempenho, robustez e preço dos computadores. Ou seja, enquanto uma arquitetura pode sobreviver por anos, a organização muda com a evolução da tecnologia.

1.1 Funções do Computador

Se, por um lado, um computador pode ser considerado um sistema complexo, com milhões de componentes eletrônicos, por outro lado, também podemos considerá-lo um sistema hierárquico, constituído por vários sub-sistemas interrelacionados. Em cada um desses níveis da hierarquia, devemos considerar a **Função** e a **Estrutura**. A função diz respeito à operação de cada componente como parte da estrutura. A estrutura refere-se ao modo como os componentes estão interrelacionados.

Dentre as funções básicas realizadas pelo computador podemos citar:

- ▶ **Processamento de dados:** Um computador deve ser capaz de processar dados. Dados estes com uma grande variedade de tipos e amplos requisitos de processamento.
- ▶ **Armazenamento de dados:** É essencial que um computador seja capaz de armazenar dados. Esse armazenamento pode ser temporário ou por períodos longos, para subsequente recuperação e modificação.
- ▶ **Movimentação de dados:** Um computador deve ser capaz de transferir dados tanto internamente, quanto com o mundo externo. Dados podem ser enviados ou recebido de dispositivos diretamente conectados ao computador. Esse processo é

conhecido como entrada e saída (E/S) e o dispositivo que esta recebendo ou enviando estes dados é conhecido como periférico. Quantos periféricos de computador você conhece? Que tal o disco rígido, o pendrive e o leitor de DVD? Quando essa transferência se dá entre o computador e um dispositivo externo, que esteja a uma distância maior, este processo é conhecido como comunicação de dados.

- **Controle:** Deve existir um controle das três funções abordadas. Em última instância, o controle é exercido pelo indivíduo que fornece instruções ao computador. Num sistema de computação, uma unidade de controle gerencia os recursos do computador e rege o desempenho de suas partes funcionais em resposta a essas instruções.

Dessa forma, o computador pode funcionar como um dispositivo de transferência de dados de um periférico para outro. Também pode funcionar como um dispositivo de armazenamento de dados, sendo os dados transferidos do ambiente externo para a memória (leitura) e vice-versa (escrita). Podem ainda processar estes dados, sendo capazes de transferir o resultado deste processamento para o ambiente externo.

1.2 Componentes do Computador

Agora que você já conhece quais são as principais funções de um computador, já está apto a compreender quais são os principais componentes que permitem a viabilização de tais funcionalidades.

São eles:

- **Central de Processamento**, também conhecido como processador ou CPU (do inglês *Central Processing Unit*), tem a responsabilidade de controlar as operações do computador e realizar as funções de processamento de dados.
- **Subsistema de Memória**, compreende todos os meios de armazenamento para os programas e para os dados dos programas existentes no computador.
- **Subsistema de Entrada e Saída (E/S)**, tem a função de transferir dados entre o computador e o ambiente externo.
- **Subsistema de Interconexão**, composto pelos Mecanismos

que estabelecem a comunicação entre a CPU, memória principal e dispositivos de E/S.

Tais componentes são observados na Figura 2. Neste livro, estudaremos cada um dos subsistemas internos ao computador, iniciando pela parte responsável pelo processamento de dados, ou seja, a CPU.

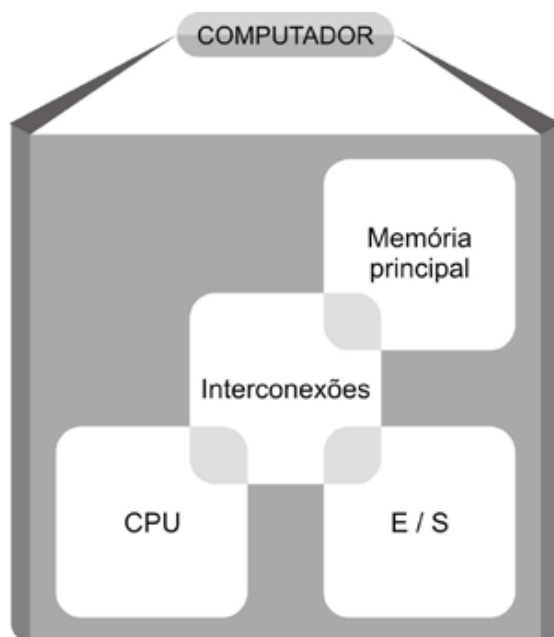


Figura 2 – Componentes internos

Central de Processamento - CPU

A CPU pode ser considerado o cérebro do computador, sendo responsável pela execução de todas as tarefas e pelo processamento de dados. Todas as operações aritméticas e lógicas existentes em um programa de computador são executadas por ela. Uma CPU pode ser composta por um ou vários processadores, os quais podem estar juntos em um mesmo componente, formando o que ficou conhecido como processador *multi-core*, uma expressão inglesa que significa literalmente multi-núcleo, neste caso cada processador é considerado um núcleo de processamento, ou separados, cada processador em um componente distinto. Em ambos os casos dizemos que o sistema é multiprocessado. Como veremos mais a frente, nos dois casos o objetivo é aumentar a capacidade de processamento do computador permitindo que várias operações possam ser executadas simultaneamente pelos diversos *cores* da CPU. A Figura 3 a seguir nos traz uma foto do processador Athlon X2, que possui internamente dois

processadores de 32 bits, sendo portanto da categoria *dual-core*.



Figura 3 – Processador Athlon X2

Os processadores possuem vários componentes internos, conforme observado na Figura 4. Podemos subdividir a CPU nos seguintes componentes internos:

- ▶ **Unidade de controle:** A unidade de Controle, como o próprio nome sugere, controla cada operação da CPU e, portanto, do computador. Ela é responsável por decodificar as instruções presentes no programa em execução emitindo sinais de controle para as demais partes do computador.
- ▶ **Unidade lógica e aritmética (ULA):** A ULA é responsável por executar as funções de processamento de dados do computador, realizando as operações aritméticas e lógicas presentes no programa.
- ▶ **Registradores:** Os registradores fornecem espaço de armazenamento interno para a CPU, servindo como uma memória de acesso ultra-rápido.
- ▶ **Interconexão da CPU:** As interconexões da CPU são os elementos que possibilitam a comunicação entre a unidade de controle, a ULA e os registradores.

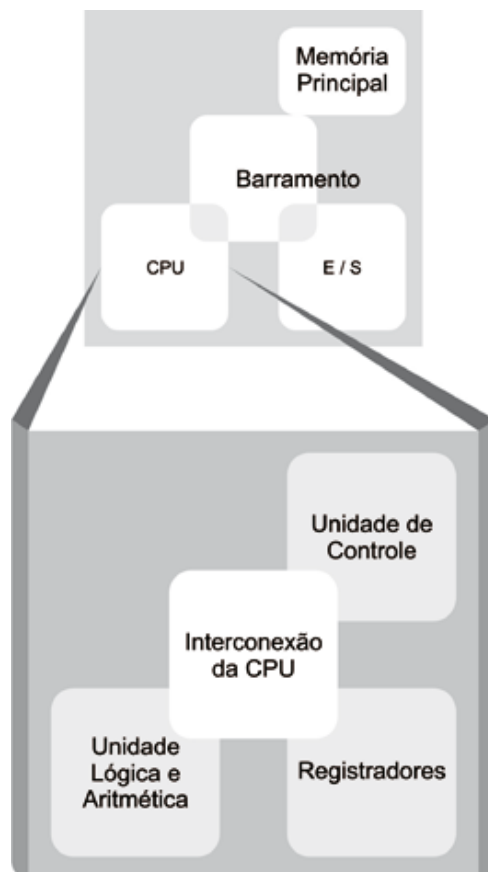


Figura 4 – Componentes internos a CPU

Agora que você já foi apresentado(a) aos subsistemas do computador, que tal conhecermos um pouco mais sobre a evolução dos computadores? Para os computadores chegarem até os dias atuais em termos de organização e arquitetura de hardware muita coisa foi modificada. Nesse sentido, na próxima seção, a nossa proposta é ampliar as discussões sobre a evolução dos computadores. Vamos lá?

1.3 Evolução dos Computadores

As máquinas, sobretudo os computadores, foram criados para auxiliar o trabalho humano. Muitas das suas tarefas se baseiam na construção de algoritmos que visam simular ações do cérebro humano. Podemos fazer uma rápida comparação para constatar que os seres humanos são capazes de se adaptar a novas situações e a aprenderem com os seus erros. Já as máquinas são capazes de executar tarefas repetidamente, com rapidez, sem sofrer de monotonia e com perfeição. As máquinas também têm capacidade de armazenar

grandes volumes de dados e informações.

O primeiro computador eletrônico da história surgiu em 1946, fabricado pela Universidade da Pensilvânia e chamava-se ENIAC (*Eletronic Numeric Integrator and Computer*). Ele pesava 50 toneladas, media 5,50 metros de altura e 25 metros de comprimento. Isso equivale a ocupar uma área de um ginásio de esportes. A primeira vez que o ENIAC foi ligado, ele consumiu tanta energia que as luzes de Filadélfia piscaram. Seu funcionamento se assemelhava a das calculadoras atuais, sendo operado manualmente, com circuitos lógicos constituídos por milhares de válvulas. O ENIAC foi criado para fins militares, na época da Segunda Guerra Mundial e pode ser observado na Figura 5.



Figura 5 – Fotografia do ENIAC

Hiperlink
Caso você deseje
visualizar a
fotografia original
do ENIAC acesse
o site: [http://
www.arl.army.
mil/www/default.
cfm?Action
=20&Page307](http://www.arl.army.mil/www/default.cfm?Action=20&Page307)

Após o **ENIAC**, seguindo a primeira geração dos computadores, surgiu o UNIVAC I. Esse computador teve 15 unidades vendidas e um tamanho aproximado de 20m². Seus circuitos eletrônicos também eram baseados em válvulas e o seu armazenamento de dados era realizado através de papel perfurado. Em geral, os computadores da primeira geração esquentavam muito e ocupavam grandes espaços físicos. Também apresentavam grande consumo de energia e quebravam com muita frequência.

A partir de 1958, surgiu a segunda geração de computadores. Sua principal inovação em relação aos computadores da primeira geração foi a utilização de transistores em substituição às válvulas. Um transistor é um dispositivo que controla a passagem da corrente elétrica através de materiais semicondutores inteiramente sólidos.

Os transistores são aproximadamente 100 vezes menores que as válvulas, o que tem como consequência direta, a redução do tamanho dos computadores. Os computadores da segunda geração, além de menores também eram mais rápidos que a geração anterior e possuíam capacidade para executar milhares de operações por segundo, consumindo menos energia que a geração anterior.

A terceira geração dos computadores data de 1958 e teve como principal inovação a utilização de circuitos integrados. Os transistores e demais componentes eletrônicos foram miniaturizados, reduzindo bastante o consumo de energia. Com o passar dos anos, a escala de integração foi aumentando e cada vez mais tornava-se possível a utilização de mais componentes em um mesmo chip ou pastilha. Os mainframes foram os principais computadores desta geração. Como exemplo, podemos citar a família de produtos IBM 360, conforme apresentado na Figura 6.



Figura 6

A partir de 1975, iniciou a quarta geração de computadores, sendo marcada pelo surgimento do microprocessador. O microprocessador foi o principal marco desta geração e o ponto chave na larga proliferação da informática. Ele ocasionou uma baixa espetacular nos preços e uma escala de integração ainda mais acentuada, onde milhões de circuitos integrados puderam ser colocados em um único chip. A Figura 7 apresenta um computador desta geração.



Figura 7

Alguns autores consideram o surgimento da quinta geração de computadores, iniciando a partir de 1981 com o advento dos microcomputadores. A partir desta geração, os computadores puderam ser utilizados por usuários comuns, não se limitando ao uso corporativo. A Figura 8 apresenta uma imagem de um PC IBM, muito utilizado na década de 80.



Figura 8 – IBM PC

Seguindo a evolução na quinta geração, surgiram as máquinas com processamento paralelo e a arquitetura RISC (*Reduced Instruction Set Architecture*). Mas o que significa processamento paralelo? Para compreender esse conceito, deveremos introduzir dois outros: multiprogramação e multiprocessamento.

Multiprogramação é uma tentativa de manter o processador sempre ocupado com a execução de um ou mais programas por vez. Com a multiprogramação, diversos programas são carregados para a memória e o processador comuta rapidamente de um para o outro. Essa técnica permite que o processador não passe por períodos de ociosidade, à espera de resposta oriunda de programas em execução, permitindo otimizações e melhorias de desempenho de processamento. O multiprocessamento, por sua vez, caracteriza-se pelo uso de vários processadores que atuam em paralelo, cada um executando as suas tarefas

Para garantir melhor desempenho, ainda é possível que os processadores atuais utilizem multiprocessamento com multiprogramação, que é a junção dos conceitos, onde mais de um processador é utilizado e cada um deles implementa a multiprogramação, permitindo efetivamente que programas sejam executados paralelamente. Os computadores pessoais evoluíram com processadores que implementam a multiprogramação e comportam o multiprocessamento.

Seguindo a evolução na quinta geração, surgiram as máquinas com processamento paralelo, as máquinas de arquitetura RISC (*Reduced Instruction Set Architecture*), **superescalares e superpipelines**.

A quinta geração também é marcada pela miniaturização dos computadores e pela computação ubíqua, ou seja, computação a toda hora e em todo o lugar, utilizando uma grande diversidade de dispositivos. A disseminação da computação ubíqua foi impulsionada pela evolução das redes de comunicação, permitindo larguras de banda superiores, comunicação sem fio e acesso através de dispositivos de menores dimensões, como celulares, *tablet PC* e *Personal Digital Assistant* (PDA). O conceito de Computação Ubíqua foi introduzido por **Mark Weiser** e colegas (Weiser M.; Gold e Brown, 1999) que afirmavam que o usuário poderia ter acesso ao seu ambiente computacional a partir de qualquer lugar, em qualquer momento, de várias formas, através de diferentes dispositivos e tecnologias de comunicação. Com a computação ubíqua, a relação entre usuários e dispositivos computacionais muda em relação aos computadores pessoais, o que era de um para um, passa a ser de um para muitos (um usuário para vários dispositivos).

Weiser e colegas (Weiser M.; Gold e Brown, 1999) argumentam que computadores de pequenas dimensões estarão embutidos em objetos do cotidiano do usuário, tais como livros, roupas e objetos pessoais em geral. A figura 9 apresenta uma série de dispositivos computacionais que estão presentes na quinta geração.



Figura 9 – Dispositivos Computacionais

Existem diversas aplicações atualmente que compõem o universo da computação ubíqua. Desde soluções para o cenário residencial, comercial, hospitalar, dentre outros. Um exemplo interessante de uma solução proposta para computação ubíqua dentro do cenário hospitalar é o *Ubiquitous Health Service* (UHS), que é representado na Figura 10. O UHS constitui uma rede de serviços de saúde, da qual fazem parte um conjunto de hospitais geograficamente distribuídos e médicos associados que podem ter acesso aos serviços desta rede

Atenção

Os detalhes de tais arquiteturas serão apresentados no segundo volume deste material didático.

Saiba Mais

Mark D. Weiser nasceu em 1952 em Chicago. Estudou Ciência da Computação e Comunicação na Universidade de Michigan, tendo recebido seu Ph.D. em 1979. Weiser foi cientista chefe da Xerox PARC e é considerado o pai da computação ubíqua. Para saber mais sobre a sua trajetória de vida consulte <http://www-sul.stanford.edu/weiser/>

de qualquer lugar, usando um conjunto de diferentes dispositivos e redes de acesso. Esta rede favorece o relacionamento entre os médicos associados, sendo, portanto, o médico o usuário alvo do cenário. Os médicos, por sua vez, são colaboradores de alguns dos hospitais credenciados e podem acessar os serviços disponibilizados pelos hospitais, podendo fazê-lo também de sua residência, local de lazer, em trânsito ou do seu consultório.

Além de serviços específicos de cada hospital credenciado, também é possível ao médico realizar acesso ao Prontuário Eletrônico de Pacientes, a qualquer hora e de qualquer lugar, usando algum dispositivo disponível. Os hospitais, por sua vez, podem oferecer serviços específicos, como por exemplo, serviço de reserva de salas, marcação de consultas no ambulatório, localização de médicos, etc. Para estes serviços serão necessárias algumas validações de segurança e algoritmos de autenticação que são gerenciados pelo próprio hospital que disponibiliza os serviços. O cenário possibilita ao médico iniciar uma sessão de acesso ao Prontuário Eletrônico de Pacientes usando um dispositivo e transferi-la para outro dispositivo durante a sua execução.

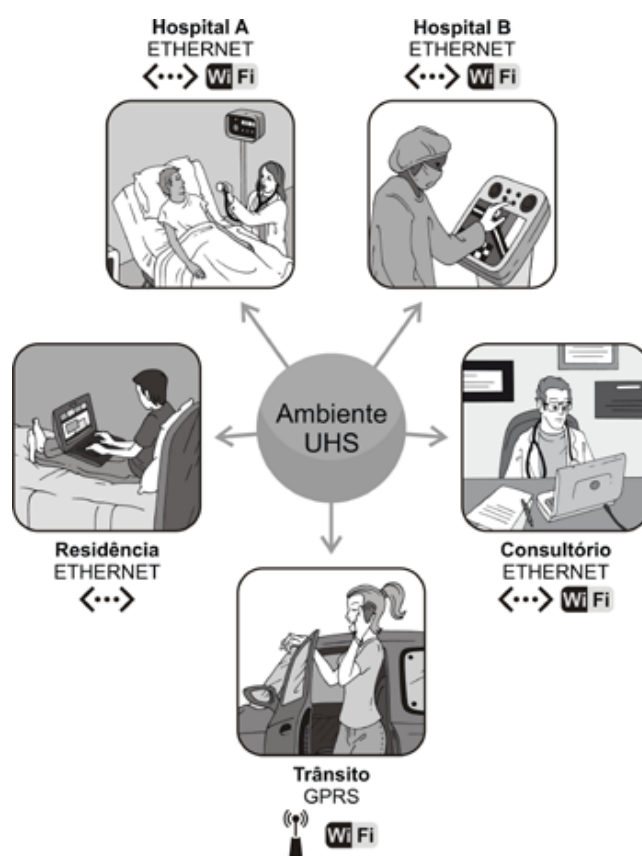


Figura 10 – Ambiente UHS

O cenário UHS é apenas um exemplo de como a computação ubíqua pode ser inserida nas nossas atividades rotineiras. Hoje somos completamente dependentes de computadores, internet, telefones celulares. A tendência é que esses dispositivos tenham cada vez mais recursos de processamento e que possam estar presentes em nossas atividades do dia a dia.



Conheça Mais

Você poderá pesquisar mais sobre arquitetura e organização de computadores em alguns websites. Seguem alguns sites para você explorar essa temática:

Computer Architecture Page: <http://www.cs.wisc.edu/arch/www/>

MIT Computer Architecture Group: <http://groups.csail.mit.edu/cag/>

Clube do Hardware: <http://www.clubedohardware.com.br/>

CPU Info Center: <http://bwrc.eecs.berkeley.edu/CIC/>

Se possível, leia também:

“**Arquitetura e Organização de Computadores**” do autor William Stallings e “**Organização Estruturada de Computadores**” do autor Andrew S. Tanenbaum.



Atividades e Orientações de Estudo

Dedique, pelo menos, 2 horas de estudo para o Capítulo 1. Você deve organizar uma metodologia de estudo que envolva a leitura dos conceitos que serão apresentados neste volume e pesquisas sobre o tema, usando a Internet e livros de referência.

Os fóruns temáticos desta disciplina podem ser utilizados para troca de informações sobre o conteúdo no ambiente virtual, pois a interação com colegas, tutores e o professor da disciplina irá ajudá-lo a refletir sobre aspectos fundamentais tratados aqui. Os chats também serão muito importantes para a interação em tempo real com o seu tutor virtual, seu professor e seus colegas.

Também é importante que você leia atentamente o guia de estudo da disciplina, pois nele você encontrará a divisão de conteúdo semanal, ajudando-o a dividir e administrar o seu tempo de estudo semanal. Procure responder as atividades propostas como atividades somativas para este capítulo, dentro dos prazos estabelecidos pelo seu professor, pois você não será avaliado apenas pelas atividades presenciais, mas também pelas virtuais. Muitos alunos não acessam o ambiente e isso poderá comprometer a nota final. Não deixe que isso aconteça com você!



Vamos Revisar?

Neste capítulo, você estudou as funções do computador e os seus componentes principais. Foram apresentados os subsistemas internos ao computador: *Central de Processamento ou subsistema de processamento*, *Subsistema de Memória*, *Subsistema de Entrada e Saída (E/S)* e *Subsistema de Interconexão*.

Aprendemos que a CPU possui o papel de controlar as operações do computador e processar dados. O *Subsistema de Memória* armazena dados e o *Subsistema de Entrada e Saída (E/S)* transfere dados entre o computador e o ambiente externo. Por fim, o *Subsistema de Interconexão*, responsabiliza-se pela comunicação entre a CPU, memória principal e dispositivos de E/S. Nos demais volumes deste material didático estudaremos com detalhes cada um desses subsistemas.

Em seguida, você pôde observar a evolução histórica dos computadores, partindo do primeiro computador eletrônico até a miniaturização dos computadores e da computação ubíqua. Aprendemos que esse conceito baseia-se na computação a toda hora e em todo o lugar, utilizando uma grande diversidade de dispositivos.

Nos próximos capítulos desse volume iremos aprender um pouco mais sobre a CPU ou subsistema de processamento iniciando com um estudo sobre operações aritméticas e lógicas.



Capítulo 2

O que vamos estudar?

Neste capítulo, vamos estudar os seguintes temas:

- » Sistemas de numeração
- » Bases numéricas e conversões entre bases
- » Operações aritméticas em diferentes sistemas de numeração

Metas

Após o estudo deste capítulo, esperamos que você consiga:

- » Conhecer os principais sistemas de numeração;
- » Realizar operações de conversões entre bases distintas;
- » Aprender as operações aritméticas básicas em sistemas de numeração binário, octal e hexadecimal.

Capítulo 2 – Conhecendo Melhor as Operações Aritméticas



Vamos conversar sobre o assunto?

2.1 Sistemas de Numeração

Desde muito cedo em sua história, a humanidade se deparou com a necessidade de contar, enumerar e ordenar as coisas, os animais e até mesmo os outros seres humanos que o cercavam. E não apenas contar, enumerar e ordenar, mas também registrar estas informações de forma clara e precisa. O sucesso tanto no comércio quanto nas atividades produtivas exigia que se pudesse tomar nota das quantidades e valores envolvidos em suas transações.

No início, enquanto os volumes eram pequenos, podia-se adotar um registro simples e direto destas informações, normalmente utilizando coisas do cotidiano para representar as quantidades envolvidas. Veja alguns exemplos destes registros na Figura 1 a seguir.

Naquela época, era comum ver pastores de ovelhas carregando consigo uma pequena bolsa de couro com tantas pedrinhas quantas fossem as ovelhas aos seus cuidados. Não existia ainda a associação de um símbolo fonético ou gráfico para as quantidades representadas, mas apenas uma associação de um para um entre as ovelhas e as pedras.



Figura 1 – Exemplos de registros de contagens feitos pelas antigas civilizações

Este recurso, por mais simples que possa parecer, permitia a estes homens controlar seus rebanhos, descobrindo, ao fim de um dia, se alguma ovelha havia se desgarrado, ou ainda se uma operação de troca de ovelhas por outras mercadorias seria ou não vantajosa para ele.

Com o passar do tempo, entretanto, o avanço natural nos volumes dos negócios e, por conseguinte das quantidades de animais e objetos envolvidos, obrigou que se elaborasse um sistema mais eficiente para registrar e contabilizar estas transações. Já não era mais possível associar convenientemente uma pedra em uma sacola ou um nó em uma corda para cada um dos elementos envolvidos nas transações.

Os **egípcios** foram os primeiros que se tem notícia a adotar um sistema gráfico estruturado de tal forma a permitir o registro simples e claro de grandes quantidades de elementos através de um pequeno número de símbolos.

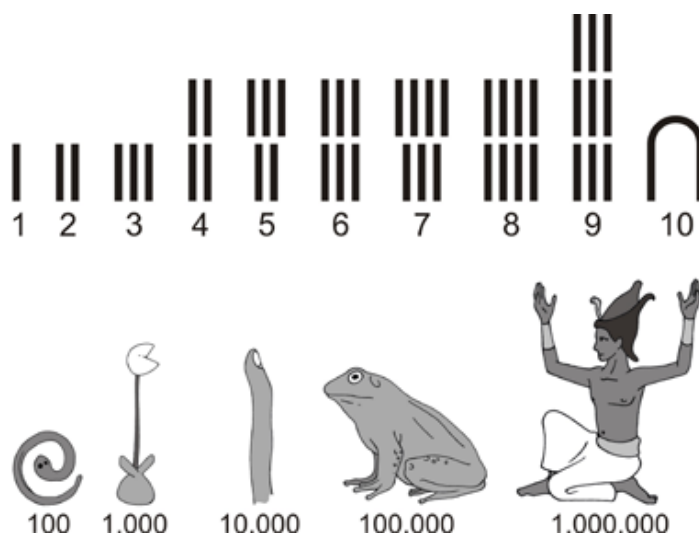


Figura 2- Símbolos adotados no sistema de numeração egípcio

Em sua representação, conforme podemos ver na Figura 2, os egípcios utilizavam dez símbolos distintos para representar valores entre 1 e 9 e, a partir daí, mais seis símbolos para representar valores da forma 10^n , com $n \geq 1$.

No sistema egípcio, de igual modo ao que fazemos hoje em dia, o valor total representado era calculado a partir do somatório dos valores atribuídos aos símbolos utilizados. E isto, por si só já simplificava em muito a forma de registrar e manipular com as informações numéricas.



Saiba Mais

Conheça um pouco mais sobre a história dos números visitando estes sites:

<http://br.geocities.com/superbetorpf/evolnum.htm>

<http://pessoal.sercomtel.com.br/matematica/fundam/numeros/numeros.htm>

Foi no norte da Índia, por volta do século V da era cristã, que provavelmente surgiu o sistema de representação que utilizamos atualmente.

Por ter sido largamente empregado pelos árabes, os quais o introduziram na Europa, este ficou conhecido como sistema de numeração Hindo-Árábico. A Figura 3 a seguir nos traz um exemplo dos símbolos gráficos utilizados no sistema Hindo-Árábico e a sua evolução até os nossos dias.

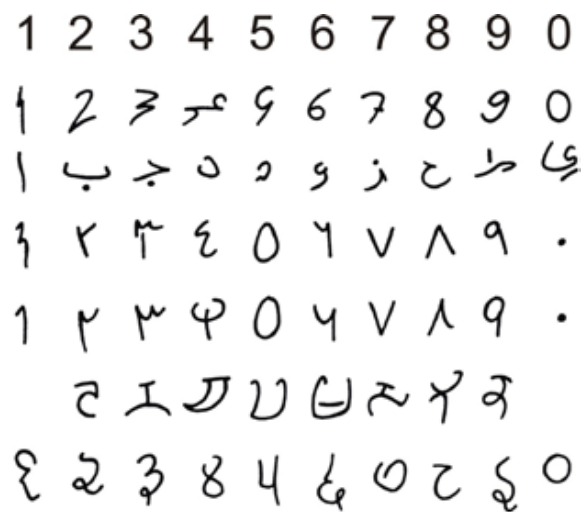


Figura 3 – Evolução da grafia dos símbolos do sistema Hindo-Árábico até os nossos dias

Diferente dos demais sistemas numéricos existentes à época, o sistema Hindo-Árábico já lançava mão de um recurso matemático conhecido hoje como base numérica, exatamente nos mesmos moldes que adotamos hoje em dia.

No sistema Hindo-Árábico, todo e qualquer valor numérico podia ser representado utilizando apenas 10 símbolos elementares, utilizando um outro recurso matemático que depois ficou conhecido como notação posicional.

Você sabe o que é uma base numérica? E a notação posicional, você sabe já ouviu falar dela? Sabe como ela funciona? Pois bem, é isto o que vamos descobrir na continuação deste capítulo.

2.2 Base Numérica

Denominamos como base numérica ao conjunto de símbolos utilizados para representar valores numéricos.

De um modo geral podemos dizer que as bases numéricas mais importantes são:

- ▶ **Base decimal**, com dez símbolos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
- ▶ **Base binária**, com dois símbolos {0, 1},
- ▶ **Base octal**, com oito símbolos {0, 1, 2, 3, 4, 5, 6, 7},
- ▶ **Base hexadecimal**, com dezesseis símbolos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

Destas, a mais importante, sem dúvida nenhuma, é a base decimal. Sua importância é tamanha que desde pequenos aprendemos a associar os seus elementos com os dedos de nossas próprias mãos. Experimente perguntar a uma criança na pré escola qual a sua idade, e de pronto ela lhe estenderá as mãozinhas mostrando nos dedos quantos aninhos já tem.

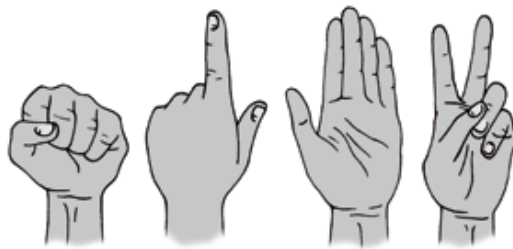


Figura 4 – Associação dos números com os dedos da mão

Mas, o que define uma base numérica? Será que nós mesmos podemos definir a nossa própria base numérica? Que regras devemos seguir para definir uma base numérica?

Para que um conjunto de símbolos possa ser considerado uma base numérica, ele deve ser formado de modo a atender algumas regras básicas de construção:

- » Cada elemento do conjunto deve estar associado a um valor numérico, numa relação de um para um. Ou seja, cada símbolo deve estar associado a um único valor numérico e vice-versa.
- » O conjunto deve possuir no mínimo dois símbolos, um para representar o zero e outro para representar o um.
- » Não devem existir lacunas na representação, ou seja, se um conjunto possuir representações para os valores n e $n+2$, então este mesmo conjunto também deve possuir uma representação para o valor $n+1$.

Atenção

Você pode estar se perguntando por que nos referimos aos números presentes nas bases numéricas como símbolos e não como números simplesmente. Como você deve ter percebido, a base hexa decimal utiliza tanto números como letras para representar os valores numéricos, desta forma, para não causar confusão aos nos referirmos às letras A,B,C,D,E e F como sendo números, nos referiremos aos números e letras utilizados, indistintamente, como sendo apenas símbolos.

Respeitadas estas regras, podem-se construir bases numéricas com tantos **símbolos** quantos se façam necessárias. Mas, pensemos um pouco, “por que se dar ao trabalho de definir outras bases numéricas se a base decimal, aparentemente, já resolve todos os nossos problemas?”

A verdade é que as bases numéricas são definidas para simplificar o dia a dia de quem as utiliza. Mesmo sem perceber, constantemente estamos lançando mão de outras bases numéricas. Por exemplo, imagine as horas do dia, você já percebeu que normalmente nós contamos as horas utilizando uma base com apenas 12 elementos? Pois é, mesmo sabendo que o dia tem 24 horas, parece-nos bem melhor pensar em termos de duas séries de 12 horas, uma começando a meia noite e indo até o meio dia, e outra começando ao meio dia e indo até a meia noite.

Como você irá perceber um pouco mais à frente, foi o desenvolvimento do estudo das bases numéricas que permitiu o desenvolvimento dos computadores digitais como nós os conhecemos hoje em dia.

Agora, pense um pouco, que outras bases numéricas você pode identificar em suas atividades cotidianas? Que tal uma base com 60 elementos, só para começar?

Mas, como será que funciona uma base numérica? Será que existe alguma relação entre as diversas bases numéricas existentes? Isto e muito mais é o que vamos descobrir a partir de agora com a continuação do nosso estudo.

2.3 Bases Numéricas e a Notação Posicional

Para entender como funciona uma base numérica e a notação posicional, vamos tomar como exemplo a base decimal. Uma vez que temos dez símbolos nesta base, podemos facilmente utilizá-los para representar conjuntos com até nove elementos. Observe a Figura 5 a seguir.

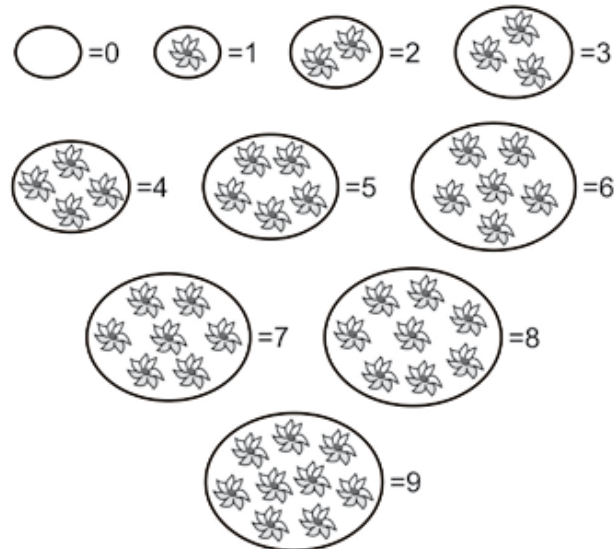


Figura 5 – Associação entre os elementos da base decimal e valores de contagem

Observe, entretanto, que um problema surge quando precisamos registrar valores para os quais não temos um símbolo associado na base, como, por exemplo, o que vemos na Figura 6 a seguir.

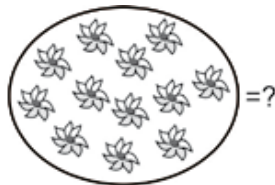


Figura 6 – Como registraremos esta quantia com a base decimal?

Como faremos para registrar os doze elementos desta figura utilizando estritamente os símbolos contidos na base decimal?

Ah, você pode estar pensando - Esta é fácil, basta utilizar os símbolos 1 e 2, exatamente nesta sequência e pronto, teremos a representação da quantidade de elementos constante nesta figura.



Se você pensou assim, é claro que a sua resposta está correta.

Entretanto, pense um pouco mais.

Se olharmos para a base decimal, veremos que os símbolos que estamos utilizando representam literalmente uma unidade seguido de duas unidades. Isto em nada se parece com as doze unidades que estamos querendo representar. De onde então tiramos a ideia que se utilizarmos estes dois símbolos em conjunto, exatamente da forma como fizemos, estaremos criando uma representação para a quantidade de elementos constantes naquela figura?

A resposta é ao mesmo tempo simples e complexa. Mesmo sem perceber, para expressar valores que não se encontram representados diretamente na base, lançamos mão de um recurso matemático inventado pelos indianos denominado **Notação Posicional**.

É o uso da notação posicional que nos permite dizer que o 1 do número 12 vale uma dezena e não uma unidade, o qual somado às duas unidades representadas pelo 2 nos dá como resultado os doze elementos que estávamos querendo representar.

A ideia por trás da notação posicional é extremamente simples e pode ser facilmente ilustrada pela observação do que ocorre quando contamos as horas em um relógio com ponteiros.

Observe que um dia tem vinte e quatro horas e um relógio com ponteiros possui apenas doze marcações, indo do 1 ao 12. Desta forma, após contarmos as 12 primeiras horas do dia, acabamos retornando ao nosso ponto de partida, em um processo conhecido como **estouro da base**. Nesta situação, se continuarmos contando, simplesmente perderemos a referência de que horas realmente estamos **registrando**.

Atenção

Notação Posicional nada mais é que um recurso de representação que nos permite alterar o valor atribuído aos símbolos de uma determinada base numérica, de forma a poder utilizá-los para representar valores que não existam na referida base.

Saiba Mais

A expressão **Estouro da Base** indica que estamos tentando representar um valor maior que o maior valor representável diretamente pela base numérica adotada.

Curiosidade...

Você conhece algum relógio com ponteiros com alguma indicação para 13 horas?

No caso do relógio, isto se resolve associando à informação indicada nos ponteiros com uma informação adicional que indica se esta é a primeira ou a segunda volta que este executa no dia. Normalmente, quando os ponteiros retornam ao ponto de partida nós,

ou adicionamos 12 ao valor indicado, como podemos ver na Figura 7 a seguir, ou simplesmente adicionamos uma informação de que são tantas horas da manhã, da tarde ou da noite.



Figura 7 – Que horas são neste relógio? 10 horas e dez minutos ou 22 horas e 10 minutos?

Ainda que extremamente simples, esta observação adicional indicando quantas voltas o ponteiro do relógio já efetuou é o elemento chave para a compreensão do como funciona uma base numérica e o mecanismo da notação posicional.

Vamos agora tentar estender esta ideia para a base decimal.

2.4 Base Decimal

Considere a Figura 8 a seguir.

Nela temos uma representação da base decimal imitando as horas de um relógio.

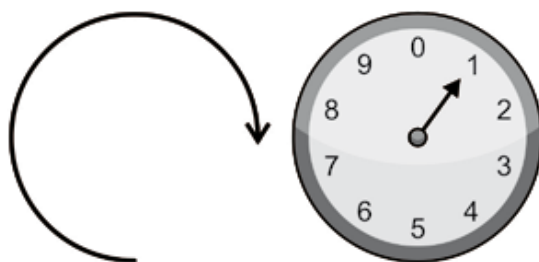


Figura 8 – Representação esquemática da base decimal

Tentemos agora utilizar este nosso disco para contar de zero até cem.

Começemos no zero e, a cada número que contarmos avancemos o nosso ponteiro de uma posição. Perceba que podemos contar de zero a nove sem problemas, mas ao chegarmos ao dez, à semelhança do que ocorre com os ponteiros de um relógio ao indicar o meio dia

ou a meia noite, ocorre um estouro da base, e retornamos ao nosso ponto inicial, perdendo assim a nossa referência de contagem. Se continuarmos contando, não haverá nenhuma forma de indicarmos claramente a que número estamos nos referindo simplesmente pelo que estamos apontando em nosso disco.

Desta forma, a fim de evitar esta confusão, incluiremos um segundo disco numerado, semelhante ao primeiro, no qual contaremos quantas voltas o primeiro disco já efetuou.

Uma vez que este disco adicional contará quantas vezes o primeiro disco contou até dez, chamar-lhe-emos de disco das dezenas. Ao primeiro disco chamaremos de disco das unidades. Por convenção, o disco das unidades ficará a direita do disco das dezenas.

O nosso novo esquema ficará conforme indicado na Figura 9 a seguir.



Figura 9 – Novo esquema de representação para contagem de dezenas e unidades, indicando a contagem de dez elementos

Fique por Dentro

O termo “vai um” indica que precisamos somar um na contagem do segundo disco.

Podemos assim continuar nossa contagem, de tal forma que a cada volta completa do disco das unidades, avançaremos o disco das dezenas de uma posição. Ou seja, sempre que ocorre um estouro da base no disco das unidades, dizemos que ocorre um “vai um” para o disco das dezenas.

Este esquema funciona muito bem até chegarmos à contagem noventa e nove. Neste ponto percebemos que o disco de dezenas também está prestes a completar uma volta completa, de tal forma que corremos o risco novamente de perder nossa referência.

Por sorte, podemos adotar o mesmo artifício já utilizado anteriormente com o disco das unidades, e incluir um novo disco para contar quantas voltas o disco das dezenas completou. A este novo disco chamaremos de disco das centenas. Nosso esquema ficará

como representado na Figura 10 a seguir.



Figura 10 – Esquema com discos para as centenas, dezenas e unidades, indicando a contagem de cem elementos

Perceba que este artifício de incluir um novo disco para contar quantas voltas o disco anterior completou pode continuar indefinidamente, de forma que podemos incluir tantos discos quantos sejam necessários, à medida que formos precisando.

A contagem total representado pelos nossos discos poderá ser facilmente verificada somando as indicações de cada um dos ponteiros multiplicadas pelos números de voltas que cada disco representa.

Uma vez compreendida esta estratégia, podemos deixar de lado o esquema com discos, substituindo cada um destes diretamente pelo seu valor, conforme estamos acostumados a fazer, com um algarismo para as unidades, um para as dezenas, um para as centenas e assim por diante. A Figura 11 a seguir nos dá um exemplo do que estamos dizendo.

Este processo todo que acabamos de descrever é a base da notação posicional. Ou seja, é ele que nos permite utilizar uma base numérica para representar todo e qualquer número inteiro que possamos imaginar.

Como pudemos ver, com exceção do primeiro dígito, o que conta as unidades, todos os demais contam quantas vezes o seu antecessor já contou até o limite da base adotada, ou seja, quantas vezes ocorreu um estouro da base no dígito que o antecede. No caso específico da base decimal, isto indica quantas vezes o dígito a sua direita contou até dez.

Desta forma, como podemos ver pela Figura 11, o valor total da contagem é obtido somando-se o valor registrado em cada um dos dígitos, multiplicado pela contagem que a sua posição representa. Muito fácil não é mesmo?

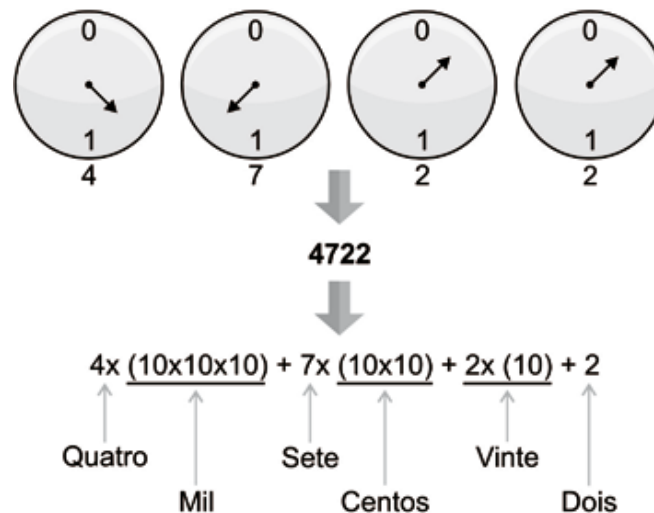


Figura 11 – Exemplo de relação entre os valores indicados nos disco e o valor que representam

Nos livros em geral, você encontrará este procedimento normalmente expresso na forma de um somatório ponderado de termos, no caso, dos dígitos utilizados, a semelhança do que vemos na Figura 12 a seguir.

$$\text{Valor Representado} = \sum_{i=0}^{n-1} d_i * \beta^i$$

Figura 12 – Expressão matemática equivalente à notação posicional

Esta expressão indica que o valor representado por uma sequência numérica é dado pelo somatório dos símbolos que a compõem, aqui indicados por d_i , ponderados pela posição que ocupam na sequência dada, aqui indicado pelo índice i à direita do d . Em resumo, nesta expressão, β é a base numérica adotada, d é o valor atribuído ao símbolo e i é a posição que o símbolo ocupa na sequência.

Um detalhe importante a ser observado, e que pode não ficar claro pela expressão, é que **os símbolos devem sempre ser ponderados da direita para a esquerda**. Ou seja, o primeiro dígito, ou o dígito mais a esquerda, será ponderado pela base elevada a zero, o que significa que o seu valor será multiplicado por 1. O segundo dígito será ponderado pela base elevada a um, o que significa para a base decimal que este será multiplicado por dez, e assim por diante.

**Curiosidade...**

Você sabia que no início da sua utilização os números em notação posicional eram escritos da esquerda para a direita, ao contrário do que escrevemos hoje? Os Hindus, inventores da notação posicional, escreviam os números da mesma forma como nós escrevemos as letras em um texto, da esquerda para a direita. Por que será que hoje nós escrevemos os números ao contrário? Descubra mais nestes links:

<http://pessoal.sercomtel.com.br/matematica/fundam/numeros/numeros.htm>
<http://professorjairojr.blogspot.com/>

Para fixar o que aprendemos, tentemos analisar como a notação posicional foi utilizada na formação dos números a seguir:

a) $500 = 5 * 100 = 5 * 10^2$

b) $321 = 300 + 20 + 1 = 3 * 10^2 + 2 * 10^1 + 1 * 10^0$

c) $782 = 700 + 80 + 2 = 7 * 10^2 + 8 * 10^1 + 2 * 10^0$

d) $1247 = 1000 + 200 + 40 + 7 = 1 * 10^3 + 2 * 10^2 + 4 * 10^1 + 7 * 10^0$

Você consegue perceber que o que temos aqui nada mais é que a aplicação direta da expressão matemática apresentada na Figura 12? É muito importante que isto fique bem claro para você. Caso esteja sentindo alguma dificuldade, pesa ajuda ao seu monitor.

Esta análise, por simples que pareça, é essencial para que compreendamos o funcionamento da notação posicional e será de suma importância para entendermos também como funcionam as outras bases numéricas.

Agora que já sabemos como expressar os números inteiros utilizando a notação posicional, podemos buscar entender como podemos utilizar esta mesma notação posicional para representar também os números fracionários.

2.5 Números Fracionários

Observe os exemplos a seguir:

a) $\frac{1}{10} = 0,1 = 1 * 10^{-1}$

$$b) \frac{3}{10} = 3 * \frac{1}{10} = 0,3 = 3 * 10^{-1}$$

$$c) \frac{5}{100} = 5 * \frac{1}{100} = 5 * \frac{1}{10} * \frac{1}{10} = 5 * 0,01 = 0,05 = 5 * 10^{-2}$$

$$d) \frac{5}{100} = \frac{10}{100} + \frac{5}{100} = 1 * \frac{1}{10} + 5 * \frac{1}{100} = 0,15 = 1 * 10^{-1} + 5 * 10^{-2}$$

$$e) 111,11 = 1 * 10^2 + 1 * 10^1 + 1 * 10^0 + 1 * 10^{-1} + 1 * 10^{-2}$$

Você conseguiu notar algo de familiar com o que acabamos de estudar?

Ao analisarmos estes exemplos, a primeira coisa que percebemos é a inclusão da vírgula como elemento separador entre a parte inteira e a parte fracionária dos números.

Outro ponto muito importante é que os dígitos colocados à direita da vírgula, que representam a parte fracionária do número, são ponderados pela base com o **expoente negativo**, o que significa que em vez de estarem sendo multiplicados estão sendo divididos por ela.

Observe novamente os exemplos **d** e **e**. Veja que da mesma forma como ocorre com os números inteiros, o valor expresso por um número fracionário na notação posicional também é obtido como o somatório de todos os termos que o compõem.

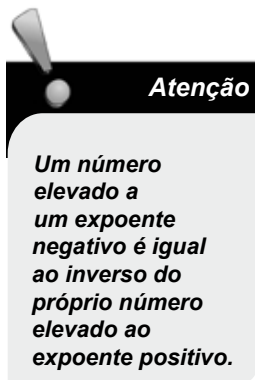
Em resumo, para representar um número fracionário na notação posicional, basta incluir a vírgula para separar a parte inteira da parte fracionária, ponderando os dígitos que ficam à esquerda desta pela base elevada a um expoente positivo e os que ficam à direita com a base elevada ao expoente negativo.

Para finalizar, podemos agora reescrever a expressão apresentada na Figura 12, de forma que esta contemple também os números fracionários. A Figura 13 a seguir mostra como ficará nossa nova expressão.

$$\text{Valor Representado} = \sum_{i=0}^n d_i * \beta^{i-k}$$

Figura 13 – Expressão Matemática para a Notação Posicional incluindo os números fracionários

Esta nova expressão difere da anterior pela inclusão do k o qual indica quantos dígitos existem à direita da vírgula, ou seja, quantos



dígitos foram reservados para a representação da parte fracionária do número.

Que tal agora exercitarmos um pouco tudo o que aprendemos fazendo uma análise aprofundada dos números a seguir. Utilize a expressão apresentada na Figura 13. Repita o que foi feito nos três exercícios seguintes.

a) 12,41

$n = 4$ (Quantidade de dígitos do número)

$k = 2$ (Quantidade de dígitos da parte decimal do número)

$$\text{Valor Representado} = 1 * 10^{-2} + 4 * 10^{-1} + 2 * 10^0 + 1 * 10^1$$

b) 73,5309

$n = 6$

$k = 4$

$$\text{Valor Representado} = 9 * 10^{-4} + 0 * 10^{-3} + 3 * 10^{-2} + 5 * 10^{-1} + 3 * 10^0 + 7 * 10^1$$

c) 1025

$n = 4$

$k = 0$

$$\text{Valor Representado} = 5 * 10^0 + 2 * 10^1 + 0 * 10^2 + 1 * 10^3$$

d) 832,0

$n =$

$k =$

Valor Representado =

e) 0,322

$n =$

$k =$

Valor Representado =

f) 10,005

$n =$

$k =$

Agora que já sabemos tudo o que precisamos sobre o que é uma base numérica e como funciona a notação posicional, podemos nos aventurar a trabalhar com as outras bases numéricas anteriormente citadas.

A primeira base que estudaremos será a base binária.

2.6 Base Binária

Atenção

Como veremos mais a frente, nem todos os números fracionários podem ser representados na base binária.

Como vimos no início deste capítulo, a base binária é formada por um conjunto com apenas dois símbolos: o zero e o um, $\{0, 1\}$. Mesmo assim, acredite, lançando mão da notação posicional podemos representar com a base binária **praticamente todos** os números que você possa imaginar.

Por ser a base numérica adotada nas **Unidades Lógicas e Aritméticas** (ULA) de todos os processadores, esta base numérica é extremamente importante.

Compreender como representar e operar com números na base binária é essencial à compreensão de como um computador funciona. Por isto, vamos tentar ser bem minuciosos em nosso estudo.

Devido a sua importância, alguns termos foram definidos para melhor retratar alguns elementos presentes nos números expressos na base binária, são eles:

- ▶ **Bit** – Nome dado aos dígitos de um número expresso na base binária. Desta forma, é comum utilizar-se expressões como 8 bits, 16 bits e 32 bits para se referir à quantidade de dígitos de um número expresso na base binária.
- ▶ **MSB** – Sigla inglesa derivada da expressão *Most Significant Bit* que significa literalmente Bit Mais Significativo. Este é o nome dado ao bit com maior peso associado, ou seja, ao bit localizado mais a esquerda do número.
- ▶ **LSB** – Sigla inglesa derivada da expressão *Least Significant Bit* que significa literalmente Bit Menos Significativo. Este é o nome dado ao bit com menor peso associado, ou seja, ao bit localizado mais a direita do número.
- ▶ **Palavra** – Nome dado ao conjunto de bits utilizados para representar um número. Esta expressão é normalmente utilizada para indicar a quantidade de bits que um processador

utiliza internamente para representar suas informações, em expressões do tipo: Os processadores Core Duo utilizam palavras de 32 bits.

A Figura 14 a seguir demonstra como estes elementos estão presentes na representação dos números representados na base binária.

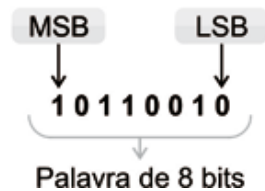


Figura 14 – Elementos presentes nos números expressos na base binária

Atenção

A Unidade Lógica e aritmética é uma das partes mais importantes da Unidade Central de Processamento dos computadores. Ela é responsável por todas as operações lógicas e aritméticas dos programas que são executados.

Agora que já conhecemos alguns detalhes dos números expressos na base binária, podemos começar o nosso estudo de como esta base realmente funciona. Para começar, vamos fazer uma análise semelhante a que fizemos com a base decimal.

Primeiramente, tomemos um disco com a representação da base binária.



Figura 15 – Disco com a representação da base binária

Agora, utilizando este nosso disco binário, vamos tentar contar até dez.

Acompanhe a contagem na Figura 16 a seguir.

Começamos contando: zero, um, mas quando vamos tentar contar o dois percebemos que já estamos chegando ao zero novamente, ou seja, se continuarmos contando estouraremos a base e perderemos a nossa referência.

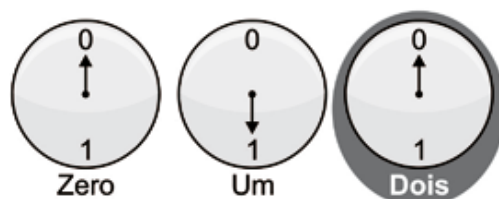


Figura 16 – Configuração do disco binário ao contarmos do zero ao dois

O que fazer então? Vamos adotar a mesma estratégia que já utilizamos na base decimal. Vamos incluir um segundo disco para contar quantas vezes o primeiro disco já deu. Veja como ficará nossa configuração na Figura 17 a seguir.

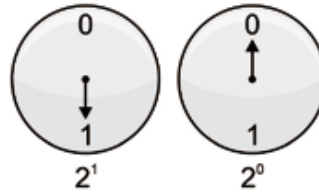


Figura 17 – Posição dos discos binários indicando que já contamos até dois

Como não temos um nome especial para dar aos discos, como tínhamos na base decimal, vamos apenas identificá-los pelo valor que cada um representa. Como o segundo disco conta quantas vezes o primeiro disco deu uma volta completa, o que ocorre a cada duas contagens, vamos identificá-lo pelo 2^1 .

Podemos, então, continuar a nossa contagem do ponto aonde paramos: Dois, três, mas quando chegamos ao quatro, vemos que o primeiro disco completa mais uma volta, o que força o segundo disco a avançar mais um passo, levando-o a completar uma volta também, voltando ao zero novamente, como podemos ver pela Figura 18 a seguir.

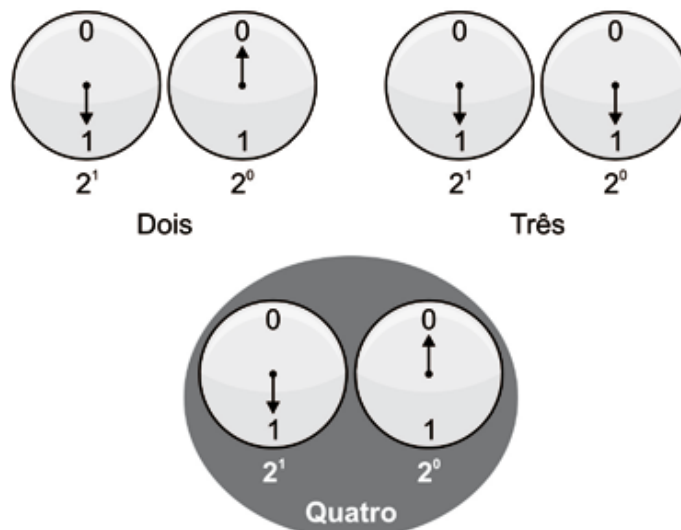


Figura 18 – Configuração dos discos binários ao contarmos dois, tres e quatro

Como sabemos, podemos simplesmente incluir mais um disco e continuar nossa contagem, conforme fizemos a pouco quando chegamos à contagem do dois. Nossa nova configuração ficará

conforme podemos ver na Figura 19 a seguir.

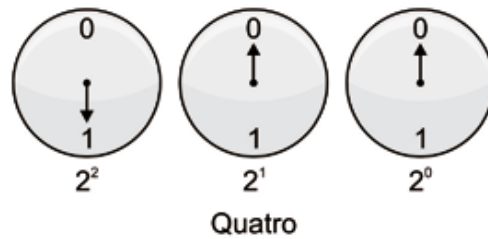


Figura 19 – Posição dos discos binários indicando que já contamos até quatro

Com mais um disco binário, podemos continuar contando. Acompanhe a nossa contagem na Figura 20 a seguir.

Devemos lembrar que sempre que o primeiro disco completa uma volta o ponteiro do segundo disco avança uma posição. E sempre que o segundo disco completa uma volta, o ponteiro do terceiro disco também avança uma posição.

Continuando a nossa contagem de onde paramos, temos: quatro, cinco, seis, sete, e quando chegamos a oito o ponteiro do terceiro disco, o que registra 2^2 , avança uma posição e também completa uma volta, retornando ao zero.

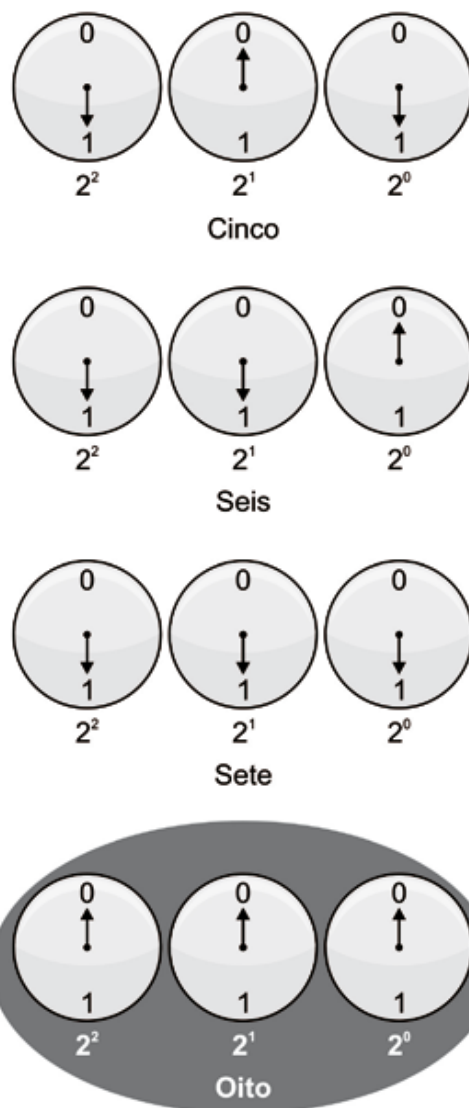


Figura 20 – Configurações dos discos binários aos contarmos de cinco até oito

Como nosso objetivo é contar até dez, vamos incluir mais um disco e continuar a nossa contagem.

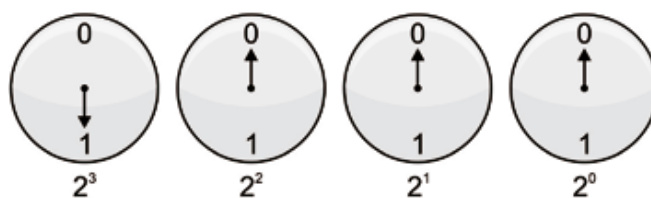


Figura 21 – Configuração dos discos binários ao contarmos oito

Agora sim, com quatro discos binários podemos concluir nossa contagem indo do oito até o dez, conforme podemos ver na Figura 22 a seguir.

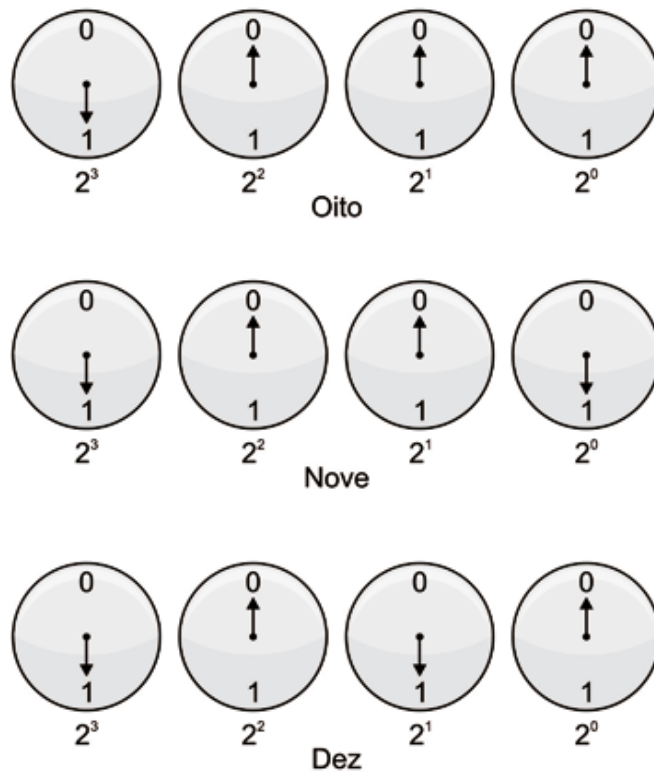


Figura 22 – Configuração dos discos binários na contagem de oito a dez

Da mesma forma que fizemos com a base decimal, uma vez que já entendemos como funciona a base binária, podemos agora deixar de lado os discos binários e passar a trabalhar diretamente com símbolos apontados em cada um dos discos. Desta forma, a nossa contagem de zero a dez em binário ficará da seguinte forma:

$$\begin{array}{lll}
 0 = 0000_2 & 4 = 0100_2 & 8 = 1000_2 \\
 1 = 0001_2 & 5 = 0101_2 & 9 = 1001_2 \\
 2 = 0010_2 & 6 = 0110_2 & 10 = 1010_2 \\
 3 = 0011_2 & 7 = 0111_2 &
 \end{array}$$

Figura 23 – Representação da contagem de zero a dez em binário

Observe que podemos aplicar a notação posicional a cada uma destas representações, conforme vimos na Figura 12, exatamente como fizemos com a base decimal, apenas substituindo a base 10 pela base 2 ($\beta = 2$). A Figura 24 a seguir demonstra como ficaria a aplicação da notação posicional a cada um dos valores encontrados:

$$0_{10} = 0000_2 = 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0$$

$$1_{10} = 0001_2 = 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$$

$$2_{10} = 0010_2 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2$$

$$3_{10} = 0011_2 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 2 + 1$$

$$4_{10} = 0100_2 = 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 4$$

$$5_{10} = 0101_2 = 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 1$$

$$6_{10} = 0110_2 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2$$

$$7_{10} = 0111_2 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 4 + 2 + 1$$

$$8_{10} = 1000_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8$$

$$9_{10} = 1001_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 1$$

$$10_{10} = 1010_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 2$$

Figura 24 – Aplicação da Notação Posicional para verificar os valores gerados na contagem de zero a dez em binário

Observe que uma vez que a base binária possui apenas dois valores, o zero e o um, nosso somatório se reduziu a computar apenas as posições aonde encontramos o um, uma vez que zero vezes qualquer outro número é igual a zero. Se pensarmos bem isto simplifica muito as coisas, uma vez que no nosso somatório aparecerão apenas termos na forma 2^n , tais como 2^0 , 2^1 , 2^2 , 2^3 e assim por diante.

De fato, o processo de conversão de um número qualquer da base decimal para a base binária consiste em encontrar esta sequência de termos da forma 2^n que quando somados resultem no valor que estamos querendo converter.

Observe os resultados obtidos nos exemplos apresentados na Figura 24. Para cada número expresso na base decimal, encontramos uma sequência equivalente de números da forma 2^n . Baseados nesta observação, vamos agora aprender duas técnicas de conversão de números da base decimal para a base binária. Nas duas abordagens nosso objetivo é o mesmo, ou seja, encontrar este conjunto de termos da forma 2^n que quando somados resulte no valor que estamos querendo converter.

2.7 Técnica de Conversão por Divisões Sucessivas

Esta técnica parte do princípio que podemos expressar qualquer número em função do resultado obtido da sua divisão inteira por outro número qualquer mais o resto encontrado. Neste caso, estaremos sempre dividindo pelo dois, uma vez que queremos expressar o número como um somatório de termos da forma 2^n , ou seja, na base binária. Veja os exemplos a seguir:

A)

$$\begin{array}{r} 3 \overline{) 2} \\ \underline{2} \\ 1 \end{array} \Rightarrow 3 = (1 \times 2) + 1 \Rightarrow 3 = 2^1 + 2^0 \Rightarrow 3 = 11_2$$

B)

$$\begin{array}{r} 5 \overline{) 2} \\ \underline{4} \\ 1 \end{array} \Rightarrow 5 = (2 \times 2) + 1 \Rightarrow 5 = 2^2 + 2^0 \Rightarrow 5 = 101_2$$

C)

$$\begin{array}{r} 9 \overline{) 2} \\ \underline{4} \\ 1 \end{array} \Rightarrow 5 = (2 \times 4) + 1 \Rightarrow 9 = 2^3 + 2^0 \Rightarrow 9 = 1001_2$$

Veja que com esta técnica não precisamos de muito esforço para converter os números 3, 5 e 9 da base decimal para a base binária. Bastou uma única divisão para encontrarmos o que queríamos. Muito simples não é mesmo?

Mas nem sempre é tão simples assim. Veja o exemplo a seguir:

D)

$$\begin{array}{r} 6 \overline{) 2} \\ \underline{2} \\ 3 \end{array} \Rightarrow 6 = (2 \times 3)$$

Observe que o resultado obtido não pode ser expresso diretamente em função da base 2, uma vez que o 3 nem é da base e nem é potência desta.

Quando isto acontece, precisamos reaplicar o processo no termo que não pode ser expresso, neste caso no 3, de forma a poder expressá-lo diretamente em função da base desejada.

Por sorte, como nós acabamos de decompor o 3 em função da base 2, no exemplo **a**, podemos simplesmente substituir o 3 do nosso resultado por $2+1$. Desta forma, o nosso exemplo ficaria:

$$\text{D) } 6 = (2 \times \overset{\substack{\uparrow \\ 3}}{(2 + 1)})$$

Agora, para concluir nosso processo, precisamos apenas eliminar todas as multiplicações presentes no resultado, uma vez que na notação posicional os números devem ser expressos exclusivamente como um somatório de termos. Faremos isto aplicando a propriedade distributiva da multiplicação em todas as multiplicações presentes. Vejamos a seguir como proceder:

$$\text{D) } 6 = (2 \times (2 + 1)) = (2 \times 2) + (2 \times 1) = 2^2 + 2^1 = 1 \ 1 \ 0_2$$

Observe que na expressão final nós incluímos um zero como primeiro dígito. Isto foi feito porque precisamos preencher todos os dígitos em nossa representação, exatamente como fazemos na base decimal. Este procedimento, como podemos constatar, não altera o resultado obtido.

Desta forma, ao fim deste processo concluímos que o 6 pode ser expresso na base binária através da sequência 110_2 , ou seja $1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$.

Um pouco mais trabalhoso, não é mesmo?

Ainda bem que existe uma forma bem mais prática e rápida de efetuarmos esta conversão, sem ter que fazer todas estas contas, de tal modo a obter o mesmo resultado com muito menos esforço.

Com este novo método, precisaremos apenas ir dividindo sucessivamente o número a ser convertido por dois, anotando sempre o resto obtido na divisão, até que o quociente obtido seja igual a 1. Terminado o processo, para encontrar a representação do número na base binária precisamos apenas escrever do último quociente obtido, seguido dos restos encontrados nas etapas de conversão na ordem inversa em que foram obtidos. Veja nos exemplos a seguir como esta técnica é rápida e prática:

E)

$$\begin{array}{r}
 6 \overline{) 2} \\
 \underline{2} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 3 \overline{) 2} \\
 \underline{2} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 2 \overline{) 1} \\
 \underline{2} \\
 0
 \end{array}
 \Rightarrow 6 = 110_2$$

3º dígito 2º dígito 1º dígito

F)

$$\begin{array}{r}
 7 \overline{) 2} \\
 \underline{2} \\
 1
 \end{array}
 \quad
 \begin{array}{r}
 3 \overline{) 2} \\
 \underline{2} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 2 \overline{) 1} \\
 \underline{2} \\
 0
 \end{array}
 \Rightarrow 7 = 111_2$$

3º dígito 2º dígito 1º dígito

G)

$$\begin{array}{r}
 11 \overline{) 2} \\
 \underline{10} \\
 1
 \end{array}
 \quad
 \begin{array}{r}
 5 \overline{) 4} \\
 \underline{4} \\
 1
 \end{array}
 \quad
 \begin{array}{r}
 2 \overline{) 2} \\
 \underline{2} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 2 \overline{) 1} \\
 \underline{2} \\
 0
 \end{array}
 \Rightarrow 11 = 1011_2$$

4º dígito 3º dígito 2º dígito 1º dígito

Aplique agora esta técnica para verificar como ficariam os números a seguir quando expressos na base binária. Depois, aproveite para verificar se os valores encontrados em binário estão corretos aplicando a notação posicional a cada um deles, faça como demonstramos na Figura 24.

a) 25 =

b) 83 =

c) 142 =

d) 65 =

e) 17 =

f) 39 =

Bem prático, não é mesmo?

Para finalizar, vamos ver uma segunda técnica de conversão que podemos utilizar para converter números pequenos, menores que 128, que podemos utilizar tanto para converter da base decimal para a base binária como da base binária para a base decimal. A esta técnica chamaremos de conversão direta.

2.8 Técnica de Conversão Direta

A ideia por trás da conversão direta é decorar os pesos a serem atribuídos a cada um dos oito primeiros dígitos dos números expressos na base binária. Por difícil que possa parecer, esta tarefa é até bem simples, uma vez que começando do bit menos significativo, o LSB, o qual é ponderado com 2^0 , a cada dígito que andarmos à esquerda simplesmente multiplicamos o valor do último dígito por dois. A Figura 25 a seguir nos traz os valores a serem utilizados.

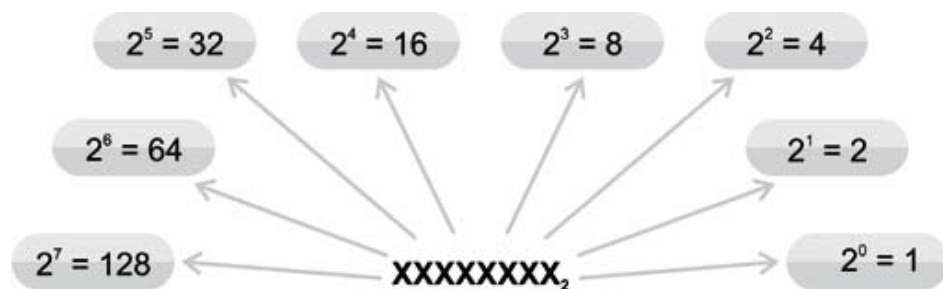


Figura 25 – Pesos a serem atribuídos aos primeiros 8 dígitos de um número escrito na base decimal

Desta forma, para convertermos um número de binário para decimal, basta observar pela figura em quanto devemos ponderar cada dígito do número dado, somado os valores obtidos, exatamente como faríamos com um número expresso na base decimal.

Com um pouco de prática, ao longo do tempo, mesmo sem perceber acabamos decorando os pesos a serem atribuídos a cada um dos dígitos, de tal forma que ao ver um número, instintivamente, ponderaremos o primeiro dígito por um, o segundo por dois, o terceiro por quatro e assim por diante.

Desta forma, a tarefa de converter um número da base binária para a base decimal será quase tão simples como trabalhar diretamente com números da base decimal.

Que tal fazermos um teste? Utilizando os valores indicados na Figura 25, tente converter os números a seguir da base binária para a base decimal. Veja na letra **a** um exemplo de como proceder.

a) $10101010_2 = 2 + 8 + 32 + 128 = 170$

b) 01010101_2

c) 10010011_2

d) 11110011_2

e) 11000111_2

Muito fácil, não é mesmo?

Vejamos agora como podemos utilizar esta mesma técnica para converter um número da base decimal para a base binária.

Partindo do princípio que todo e qualquer valor deve ser expresso na forma de um somatório de termos, ou seja, na forma de um somatório dos valores apresentados na Figura 25, podemos simplesmente fazer a operação inversa, e passarmos a efetuar subtrações sucessivas entre o número a ser convertido e os valores apresentados, assinalando os termos que estamos utilizando.

Os únicos cuidados que devemos tomar é sempre começar nossas subtrações pelo maior número que for possível subtrair, de forma a obter o menor resto possível, e ir assinalando os números que forem sendo utilizados.

Ao fim do processo, quando obtivermos o resto zero em nossas subtrações, basta simplesmente formar a nossa representação em binário substituindo os números que foram utilizados por 1 e os que não foram utilizados por 0.

Tomemos como exemplo o número 26, vamos tentar convertê-lo para a base binária utilizando a técnica da conversão direta.

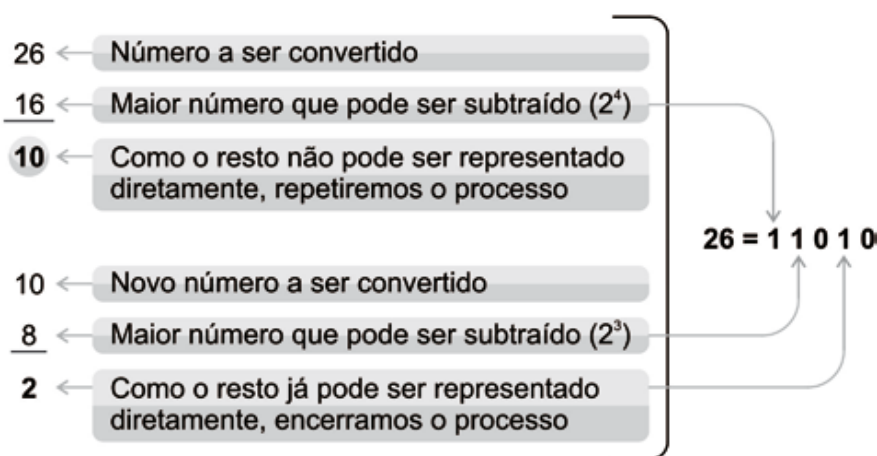


Figura 26 – Exemplo de conversão para a base binária utilizando a conversão direta

Agora tente você mesmo, converta os números a seguir para a base binária utilizando a conversão direta.

a) 35

b) 24

c) 72

d) 47

Muito fácil, não é mesmo?

Para finalizar esta primeira parte do nosso estudo, vamos ver como podemos converter os números fracionários da base decimal para a base binária.

2.9 Números Fracionários na Base Binária

A representação dos números fracionários na base binária se dá exatamente da mesma forma como aprendemos na base decimal.

Para começar, vamos relembrar como ponderamos os dígitos utilizados para representar a parte fracionária de um número na base decimal. Por exemplo, qual a diferença dos pesos atribuídos aos dígitos de 10,0 e 0,1? Observe a Figura 27 a seguir.

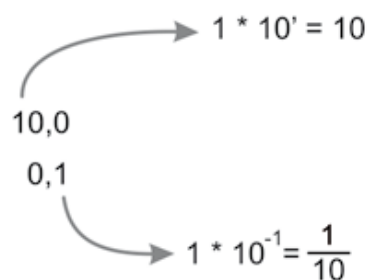


Figura 27 – Exemplo de pesos atribuídos aos dígitos de um número fracionário

Veja que, no primeiro caso, o 1 antes da vírgula foi ponderado com 10^1 , ou seja, por dez mesmo. Já no segundo caso, o 1 após a vírgula foi ponderado com 10^{-1} , ou seja, por $\frac{1}{10}$. Nos dois casos, o número 1 estava uma posição distante da posição das unidades, que é a primeira posição a esquerda da vírgula. Ambos foram ponderados de acordo com a sua posição, só que no primeiro número, como o 1 estava à esquerda da vírgula, este foi ponderado positivamente e o que estava à direita negativamente.

Desta forma, dizemos que os dígitos reservados à representação da parte fracionária, os quais são colocados à direita da vírgula, são ponderados negativamente, enquanto que os reservados à parte inteira, os que ficam à esquerda da vírgula são ponderados positivamente.

Esta é também a estratégia adotada para a representação dos números fracionários na base binária. Da mesma forma que na base decimal, dividem-se os dígitos por uma vírgula, ficando os da esquerda reservados para a representação da parte inteira e os da direita para a parte fracionária.

Vejam alguns exemplos:

$$0,1_2 = 0 * 2^0 + 1 * 2^{-1} = 0 + \frac{1}{2^1} = 1 + \frac{1}{2} = 0,5$$

$$0,01_2 = 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} = 0 + 0 * \frac{1}{2^1} + 0 * \frac{1}{2^2} = 0 + 0 * \frac{1}{2} + 1 * \frac{1}{4} = 0,25$$

$$1,01_2 = 1 * 2^0 + 1 * 2^{-2} = 1 + \frac{1}{2^2} = 1 + \frac{1}{4} = 1,25$$

$$11,11_2 = 1 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} = 1 + 2 + \frac{1}{2^1} + \frac{1}{2^2} = 3,75$$

Para efetuarmos a conversão de um número fracionário da base decimal para a base binária, podemos ou utilizar o processo de conversão direta, conforme acabamos de ver, apenas incluindo os valores a serem atribuídos aos dígitos à direita da vírgula, ou podemos utilizar um algoritmo muito semelhante ao adotado para conversão por divisões sucessivas, só que em vez de divisões efetuaremos multiplicações sucessivas.

2.10 Algoritmo de Conversão por Multiplicações Sucessivas

O processo de conversão de um número fracionário da base decimal para a base binária pode ser efetuado de maneira simples e direta a partir do seguinte algoritmo.

Dado um número a ser convertido, separe a parte inteira da parte fracionária. Primeiramente converta a parte inteira através de uma das técnicas já apresentadas. Feito isto, podemos passar à conversão da parte fracionária da seguinte forma:

1. Verifique se a parte fracionária do número a ser convertido é igual a zero. Em caso afirmativo, o processo de conversão se encerra e a sequência de dígitos obtida é a representação desejada.
2. Caso a parte fracionária seja diferente de zero, multiplique esta

por dois. Anote e subtraia o primeiro dígito da parte inteira do resultado obtido. Este dígito deverá ser colocado á direita dos dígitos já obtidos durante o processo de conversão da parte inteira.

3. Retorne ao primeiro passo do algoritmo.

O exemplo a seguir demonstra como devemos aplicar este algoritmo.

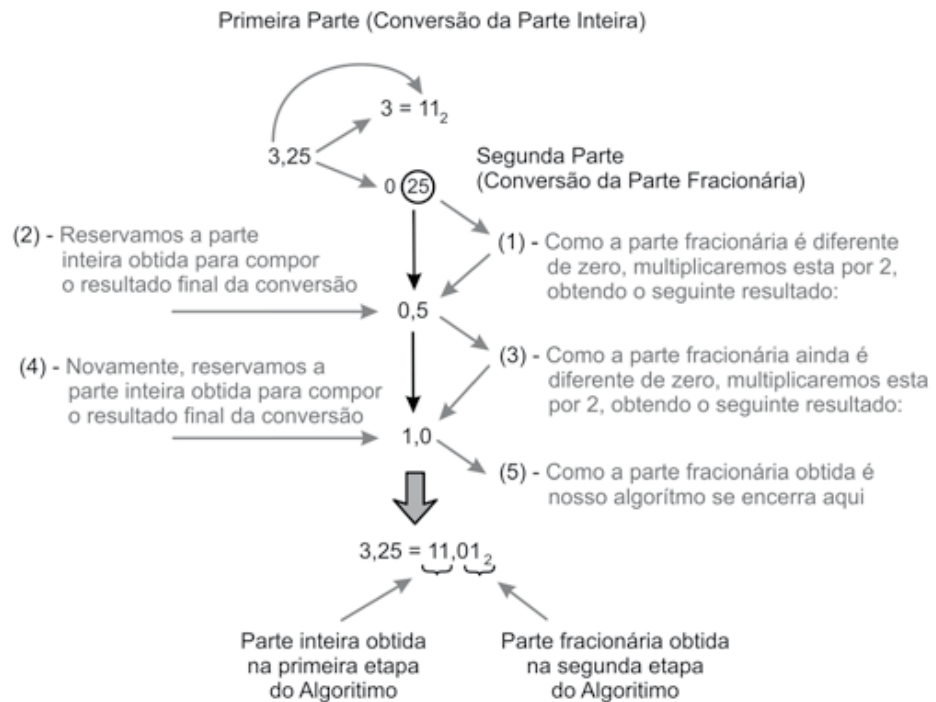


Figura 28 – Demonstração do algoritmo de conversão de Números Fracionários da Base Decimal para a Base Binária

Tente agora converter os seguintes números fracionários para a base binária:

- a) 0,125
- b) 7,750
- c) 0,1875

2.11 Operações Aritméticas na Base Binária

Agora que já conhecemos a base binária e como podemos utilizá-la para representar valores numéricos inteiros e fracionários, podemos passar a estudar como efetuar as duas principais operações aritméticas nesta base.

Lembre-se, como comentamos a princípio, nosso principal objetivo ao estudarmos a base binária é buscar entender como as Unidades Lógicas e Aritméticas dos processadores funcionam, e como estas conseguem operar com os números representados na base binária.

Operação de Adição

De um modo geral podemos dizer que não existem grandes diferenças entre as operações de adição que nós efetuamos utilizando a base decimal, e as operações de adição que as Unidades Lógicas e Aritméticas dos processadores efetuam utilizando a base binária. Na verdade, a única coisa que muda é o valor no qual ocorre o nosso conhecido “vai um”.

Nas Figuras 28, 29, 30 e 31 a seguir, temos alguns exemplos de operações de adição efetuadas tanto na base decimal quanto na base binária.

Vamos primeiro analisar o que ocorre em uma operação de adição na base decimal. Acompanhe nosso raciocínio na Figura 29 a seguir.

a) $\begin{array}{r} 09 \\ + 01 \\ \hline 10 \end{array}$ Como $9+1=10$, ocorre um “vai um e fica zero”

b) $\begin{array}{r} 125 \\ + 36 \\ \hline 161 \end{array}$ Como $5+6=11$, ocorre um “vai um e ficam um”

c) $\begin{array}{r} 193 \\ + 95 \\ \hline 288 \end{array}$ Como $3+5=8$, ocorre um “vai um e ficam oito”

Figura 29 – Exemplos de ocorrência de “vai um” na base decimal

Como sabemos, a soma de dois números quaisquer é sempre efetuada a partir de somas parciais efetuadas par a par sobre cada um dos seus algarismos. Somamos primeiro as unidades, em seguida as dezenas, as centenas e assim por diante. E, sempre que o resultado de uma destas somas parciais for maior que nove, o que caracteriza um estouro da base, efetuamos um “vai um”, *correspondente ao dígito das dezenas* do resultado obtido, ficando apenas o dígito correspondente a parte das unidades para ser registrado na posição onde ocorreu a adição.

Eu tenho certeza que você já está cansado(a) de saber disto, não é mesmo?

Pois bem, para nossa felicidade a operação de adição na base binária é até bem mais simples do que isto. Lembre-se que a base binária possui apenas dois símbolos, o zero e o um, e isto por si só já simplifica muito as coisas. Uma operação de adição entre dois

algarismos da base binária só pode assumir uma das seguintes configurações: $0 + 0$, $0 + 1$, $1 + 0$ e $1 + 1$. Não podia ser mais fácil, não é mesmo?

Vamos começar com algo bem simples, acompanhe a operação do exemplo da Figura 30 a seguir.

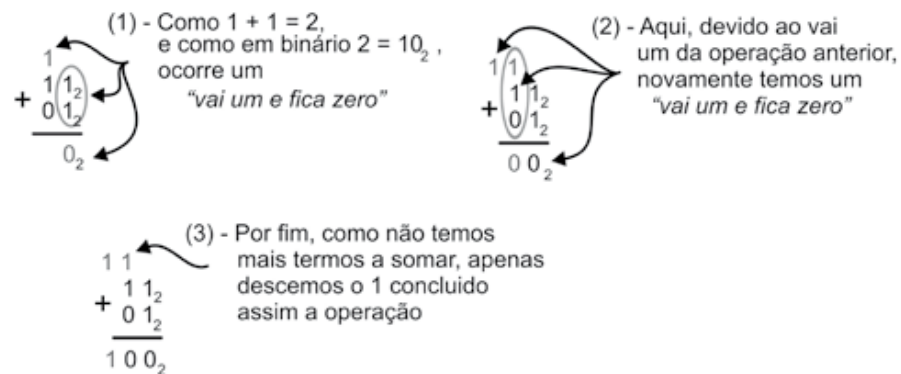


Figura 30 – Exemplo de ocorrência de “vai um” na base binária

Observe os passos 1 e 2 de nossa operação. Como você deve ter notado sempre que um resultado parcial for maior que 1, o que indica um estouro da base binária, a exemplo do que ocorre com a base decimal quando um resultado parcial é maior que nove, precisamos fazer um vai um para o dígito seguinte. Como em ambos os casos os resultados parciais são iguais a dois, e como dois em binário é representado como 10_2 , efetuaremos um vai um e fica zero. Exatamente como fizemos no exemplo a da Figura 29 quando o resultado de uma soma parcial foi igual a onze.

Observemos agora o nosso próximo exemplo:

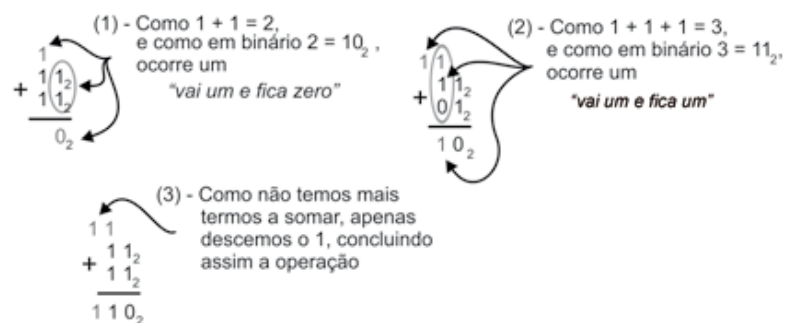


Figura 31 – Mais um exemplo de ocorrência de “vai um” na base binária

Aqui, além do vai um e fica zero ocorrido na operação anterior,

temos também um vai um e fica um, resultante da operação $1+1+1$ ocorrido no passo dois. Como todos sabemos, $1+1+1$ é igual a 3 e como 3 em binário é igual a 11, à semelhança do que vimos no exemplo b da Figura 29, precisamos fazer um “vai um e fica um” a fim de propagar o estouro da base observado no resultado obtido.

Muito simples, não é mesmo? Nosso único cuidado deve ser observar corretamente quando efetuar o “vai um”, que em binário ocorre sempre que o resultado for maior que “1”. O restante da operação ocorre exatamente como estávamos acostumados operando com a base decimal.

Agora, para terminar, observemos na Figura 32 a seguir o que pode ocorrer quando somamos mais de dois números de uma vez só em binário.

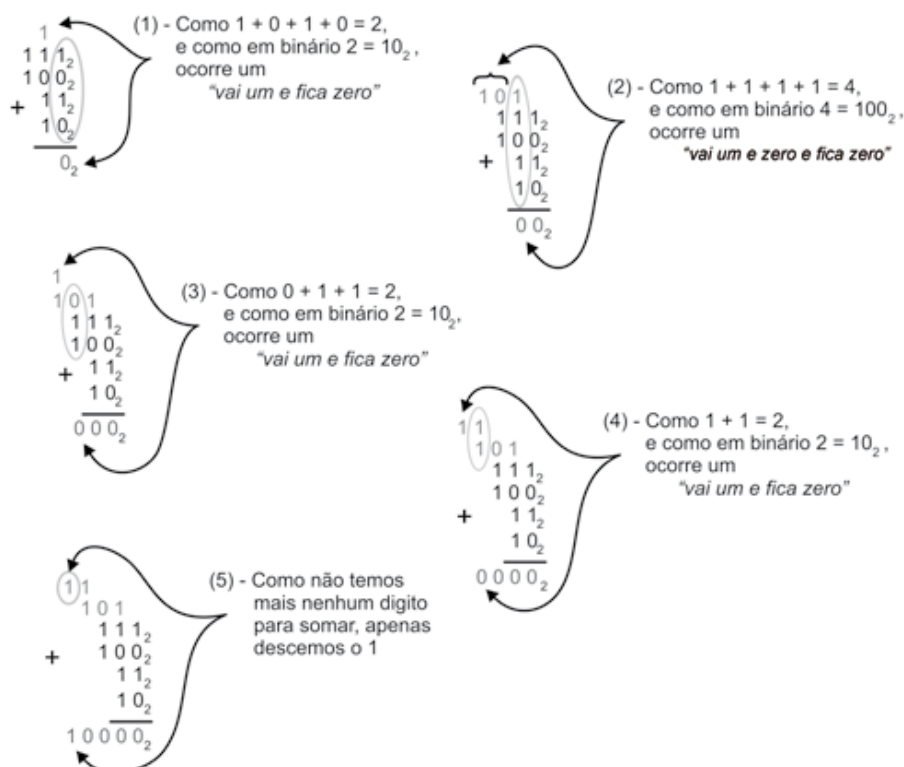


Figura 32 – Exemplo de “vai um e zero e fica um” em binário

Neste exemplo, temos uma situação bem peculiar, que dificilmente observaríamos na base decimal. No segundo passo da nossa operação, quando somamos $1 + 1 + 1 + 1$ temos um “vai um e zero e fica zero”, ou seja, temos a propagação de um vai um para a mais de uma casa à esquerda da posição que estamos operando. Isto ocorre porque, em decimal, $1 + 1 + 1 + 1$ é igual a 4 e 4 em binário é “100”,

**Fique
por Dentro**

É comum nos referirmos às posições que os dígitos de um número ocupam como casas. Como, por exemplo, a casa das unidades ou a casa das centenas.

desta forma, como sempre ficamos apenas com o primeiro dígito **passando os restantes para as casas seguintes**, temos que fazer um “*vai um e zero e fica zero*”.

Muito estranho, não é mesmo? Mas você não tem com o que se assustar, a regra é sempre a mesma, primeiramente some os dígitos da operação parcial, exatamente como se estivesse operando na base decimal, em seguida converta o resultado obtido para binário, por fim preserve apenas o dígito menos significativo do resultado obtido, propagando os demais dígitos em uma operação do tipo vai n.

Operação de Subtração

Por questões de economia, a fim de reduzir o tamanho e a complexidade do hardware empregado, as Unidades lógicas e Aritméticas dos processadores convertem as operações de subtração em operações de adição entre o primeiro operando (minuendo) e o inverso do segundo operando (subtraendo).

Como era de se esperar, e como podemos ver pela expressão a seguir, apesar de reduzir enormemente o hardware necessário nas Unidades Lógicas e Aritméticas, isto em nada altera o resultado obtido.

$$A - B \equiv A + (-B)$$

Desta forma, uma vez que já sabemos como se processa uma operação de adição na base binária, a única coisa que precisamos aprender para efetuar uma operação de subtração na base binária é aprender como podemos representar o inverso de um número nesta base.

Antes disto, entretanto, precisamos definir claramente o que é o inverso de um número. Isto será muito importante para que compreendamos o que iremos fazer para representar os números negativos na base binária.

Em qualquer sistema numérico, o inverso de um número é aquele que somado ao primeiro, tem zero como resultado.

Na base decimal, por convenção, representamos um número negativo da mesma forma que representamos um número positivo, com a única diferença que o número negativo será precedido pelo

sinal “-”. Este padrão de representação que simplesmente associa um sinal de “-” para formar a representação de um número negativo, é conhecido como representação de **Magnitude e Sinal**.

Na base binária além da representação de Magnitude e Sinal temos também uma outra, conhecida como representação de **Complemento a Dois**.

Como já dissemos, dentro da Unidade Lógica e Aritmética de um processador todos os números são representados utilizando apenas a base binária. Nem mesmo os sinais de “+” e “-” existem. De um modo geral, tanto na representação de Magnitude e Sinal quanto na representação de **Complemento a Dois**, adota-se reservar o dígito mais significativo ou MSB (*Most Significant Bit*), para sinalizar se um número é positivo ou negativo. Normalmente utiliza-se o “0” para indicar que o número é positivo e o “1” para indicar que o número é negativo.

No sistema de Magnitude e Sinal, a semelhança do que ocorre na base decimal, simplesmente ajustamos o MSB para indicar se o número é positivo ou negativo, deixando o restante da representação inalterada. A Figura 33 a seguir demonstra como ficaria a representação de alguns números positivos e negativos neste sistema.

1 = 00000001	-1 = 10000001	
3 = 00000011	-3 = 10000011	
0 = 00000000	-0 = 10000000	← ERRO !!!

Figura 33 – Representação no sistema Magnitude e Sinal

Observe que este sistema apresenta duas incoerências:

Em primeiro lugar, permite duas representações distintas para o zero, uma positiva e outra negativa.

E, em segundo lugar, se tentarmos operar a soma de um número com o seu inverso, nos moldes do que aprendemos até agora para a base binária, não obteremos zero como resultado. Ou seja, a representação de Magnitude e Sinal é muito útil e interessante para nós seres humanos, mas para os computadores é um verdadeiro desastre. Por este motivo, os sistemas computacionais normalmente adotam a representação de Complemento a Dois como representação padrão para os números dentro das Unidades Lógicas e Aritméticas.



Saiba Mais

O termo **Magnitude e Sinal** indica que temos uma representação baseada na magnitude do número representado, ou seja, o seu módulo, associado a um sinal algébrico que indica se este é um número positivo ou negativo. Normalmente não se exige o uso do sinal “+” para indicar que um número é positivo.



Saiba Mais

Apesar de também possuir uma indicação do sinal algébrico, ou seja, de sinalizar claramente se um número é positivo ou negativo, a representação em complemento a dois, como iremos ver um pouco mais a frente, adota estratégias diferentes para registrar a magnitude ou o tamanho de números negativos e positivos.

Notação de Complemento a Dois

A Notação de Complemento a Dois não apenas corrige o problema da indicação de zero negativo e positivo como também permite expressar convenientemente o inverso de um número, de tal forma que a adição deste com o seu inverso, em binário, resulte em zero binário.

Considere um número N_2 qualquer expresso na base binária. Se existe uma representação para $(-N_2)$ em binário, esta representação deve ser tal que:

$$N_2 + (-N)_2 = 0_2 \iff (-N)_2 = 0_2 - N_2$$

Figura 34 – Definição do inverso de um número em binário

Ou seja, se $(-N_2)$ existir, este certamente pode ser obtido pela operação subtração do próprio N_2 da representação do zero em binário (0_2).

Sendo assim, só para começar tentemos encontrar uma representação para o inverso do número 1 em binário, ou seja, uma representação para (-1) , de tal forma que quando somarmos esta representação ao próprio 1 em binário obtenhamos o zero binário como resultado.

Antes de qualquer coisa, precisamos primeiramente definir quantos dígitos a nossa representação terá. Isto é importante porque, conforme dissemos a princípio, o MSB de um número em binário deve ser reservado para indicar se este é positivo ou negativo. Desta forma, apenas por convenção, vamos adotar uma representação baseada em uma palavra de 8 bits. Feito isto, acompanhemos agora os demais passos apresentados na Figura 35 a seguir.

Observe que começamos definindo uma representação para o zero e outra para o um. Observe também que o MSB de ambos os números é zero, o que indica que ambos são números positivos.

No passo 1, exatamente como faríamos na base decimal, começamos nossa operação subtraindo os primeiros algarismos da representação do um e do zero, que, por coincidência são respectivamente 1 e 0.

Como 1 é maior que 0, precisamos pedir algo emprestado para

que possamos continuar a nossa operação. Esta operação de pedir emprestado, em binário é também conhecida como “vem um”. Entretanto, como podemos ver, o segundo dígito da representação também é zero, ou seja, não pode emprestar nada ao primeiro. O terceiro, o quarto, o quinto, o sexto, o sétimo e até o oitavo dígitos são todos iguais a zero. Ou seja, aparentemente nossa operação não pode continuar. Lembre-se, se estivéssemos operando na base decimal, nós simplesmente inverteríamos a operação e o sinal do seu resultado. Aqui em vez disto, como podemos ver no passo dois, faremos um “vem um externo” que, como veremos, não apenas “inverterá o sinal do nosso número” como também nos permitirá concluir convenientemente nossa operação de subtração.

Um detalhe importante aqui, como podemos ver pelo passo 3, é que, diferente do que ocorre na base decimal aonde um vem um significa somar dez ao dígito anterior, na base binária o vem um significa somar dois ao dígito anterior. Desta forma, ao se propagar, o nosso “vem um” vai fazendo com que todos os dígitos menos o LSB da representação do zero fiquem igual a 1, conforme podemos ver pelo passo quatro da nossa operação. Como o LSB não vai emprestar para ninguém, este fica com valor igual a dois ou “10” na base binária, conforme podemos ver no passo 5 da nossa operação.

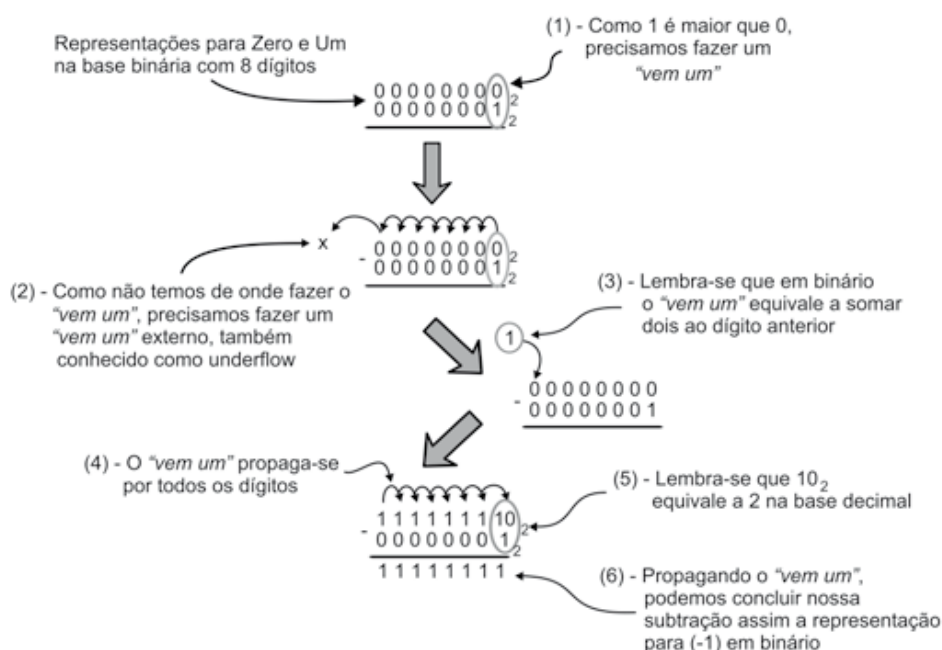


Figura 35 – Estratégia para encontrar uma representação para -1 na base binária

Agora sim, feita a propagação do “vem um externo”, podemos completar nossa operação subtração, tendo como resultado 11111111_2 ,

que é a nossa representação para (-1) na notação de complemento a dois.

Como você pode perceber, ainda que um pouco trabalho, o processo todo é até bem simples.

Vamos agora verificar se o valor que encontramos para representar o (-1), atende a prerrogativa estabelecida, ou seja, se a sua adição com a representação do 1 utilizada para obtê-la resulta em zero.

Para tanto, vamos somar a representação encontrada com a representação para o número 1 anteriormente utilizada.

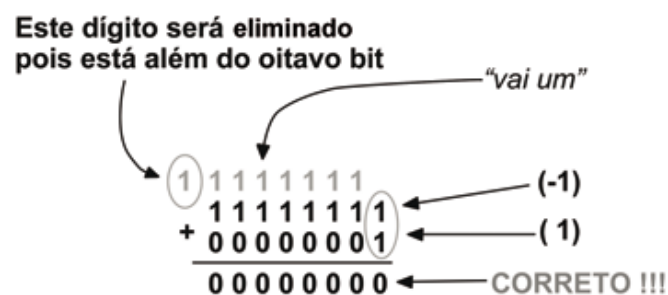


Figura 36 – Verificando se a representação de -1 encontrada está correta

Como podemos ver, com exceção do “vai um” para o nono dígito, todos os demais resultaram igual a zero, o que é uma representação válida para o zero no formato de número binário adotado.

O “vai um” do nono dígito será considerado apenas como o inverso do “vem um externo” ocorrido na operação de obtenção do (-1), sendo desta forma apenas uma indicação da inversão do sinal do resultado.

Observe que este processo não apenas atendeu à prerrogativa estabelecida, como indica corretamente o sinal do número representado. Observe que o MSB da representação para o (-1) é igual a 1, indicando que este é um número negativo. Ao passo que o MSB da representação para o (1) é igual a zero, indicando que este é um número positivo.

Muito interessante, não é mesmo?

Vamos agora ver um processo bem mais simples e prático de obter este mesmo resultado.

Na verdade, a representação em “Complemento a Dois” recebe este nome devido ao algoritmo que iremos apresentar agora.

Dado um número qualquer, inverta o valor de todos os seus dígitos, ou seja, aonde tiver “1” coloque “0” e aonde tiver “0” coloque “1”. Este processo recebe o nome de complemento a um, ou simplesmente complemento.

Adicione “1” ao resultado obtido e, como num passe de mágica, obtenha o inverso do número que estava procurando.

Na Figura 37 a seguir, temos uma demonstração deste algoritmo aplicado para obter o inverso de 1.

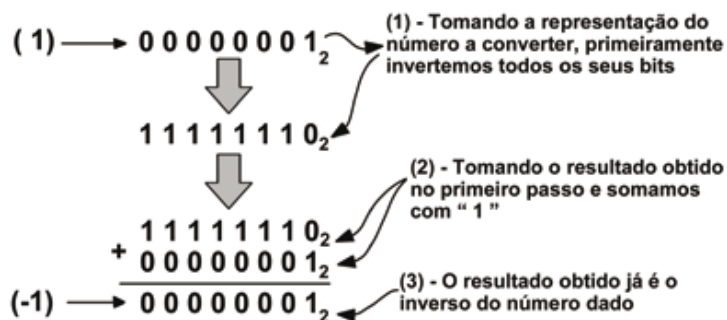


Figura 37 – Aplicação do algoritmo de conversão de números para complemento a dois

Muito interessante, não é mesmo?

Este algoritmo pode ser aplicado a qualquer número expresso na base binária. Mesmo um número negativo pode ser “invertido” e transformado em um número positivo por este processo. No exemplo a seguir temos este processo aplicado para encontrar o inverso da representação de (-1) em complemento a dois.

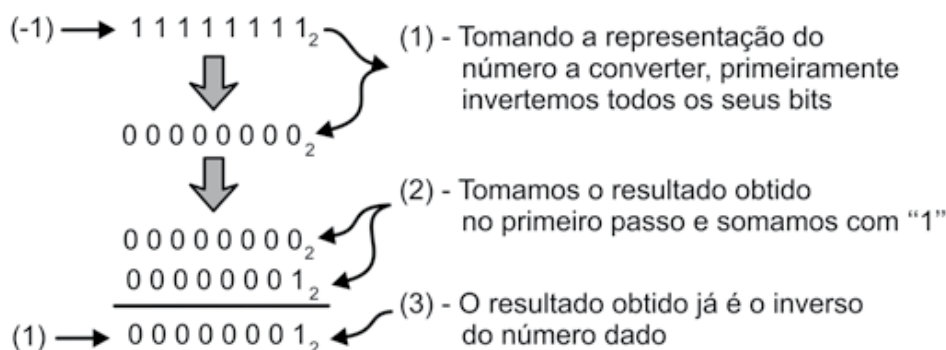


Figura 38 – Aplicação do algoritmo de conversão para complemento a dois aplicado a um número negativo

Para finalizar nosso estudo das operações aritméticas na base binária, que tal verificar o que aprendemos fazendo alguns exercícios?

A seguir tem alguns exemplos já resolvidos. Tente fazê-los por si mesmo, e no final compare os resultados obtidos. Em todos eles nós iremos adotar o padrão binário de 8 bits em complemento a dois. Para os números negativos, primeiramente vamos representá-los como números positivos e em seguida convertê-los para números negativos no padrão de complemento a dois.

$$\text{a) } 7 - 5 = 00000111 - (00000101) = 00000111 + 11111011 = 00000010 \text{ (2)}$$

$$\text{b) } 13 - 45 = 00001101_2 - (00101101_2) = 00001101_2 + 11010011_2 = 11100000_2 \text{ (-32)}$$

$$\text{c) } -21 - 43 = -(00010101_2) - (00101011_2) = 11101011_2 + 11010101_2 = 11100000_2 \text{ (-64)}$$

$$\text{d) } -35 + 102 = -(00100011_2) + 1100110_2 = 11011101_2 + 01100110_2 = 01000011_2 \text{ (67)}$$

2.12 Poder de Representação da Base Binária

Nesta última parte do nosso estudo, vamos dar uma paradinha e considerar o poder de representação desta base numérica tão importante. Para tanto, comecemos tentando respondendo algumas perguntas, que até certo ponto podem parecer meio confusas, mas que ao respondermos, nos darão uma visão mais clara deste ponto tão importante.

1. Quantos números podemos representar na base binária, adotando uma representação sem sinal, com uma palavra de N bits? E ainda, quais seriam o maior e o menor número representáveis para uma palavra deste tamanho?
2. E se adotássemos uma representação com sinal, digamos em complemento a dois, o que mudaria em nossos valores? Nesta condição, quantos números poderíamos representar e qual seria o maior e o menor deles?
3. Quantos bits, ou dígitos, seriam necessários para representar um número R qualquer, dado originalmente na base decimal, se adotássemos uma representação utilizando a base binária em uma notação com e sem sinal?

Para responder estas perguntas, pensemos um pouco... Vamos começar com a base numérica que estamos mais acostumados.

Vamos primeiramente analisar a base decimal e depois vamos aplicar as nossas conclusões à base binária.

Quantos números podemos representar na base decimal se adotarmos um formato N dígitos? Acompanhe nosso raciocínio pela Figura 39 a seguir.

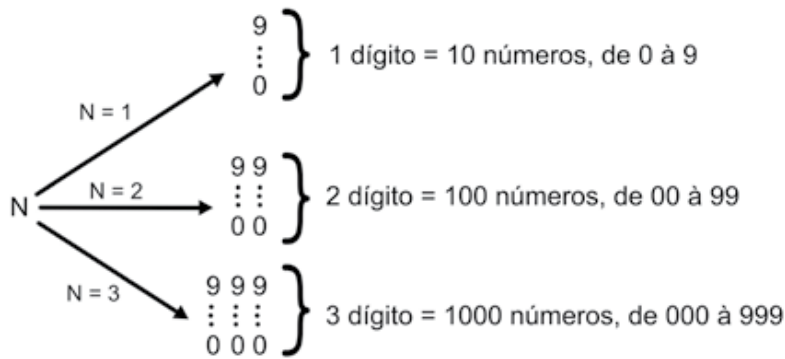


Figura 39 – Quantidade de números representáveis na base decimal com N dígitos

Observe que cada dígito que incluímos em nossa representação nos permite multiplicar por dez a quantidade de números representáveis. Com 1 dígito, se representarmos apenas números positivos podemos representar até 10 números, entre 0 e 9. Com 2 dígitos podemos representar até 100 números, entre 0 e 99. Com 3 dígitos até 1000 números, 0 e 999 e assim por diante.

Vamos agora aplicar a mesma análise para a base binária. Acompanhe na Figura 40 a seguir:

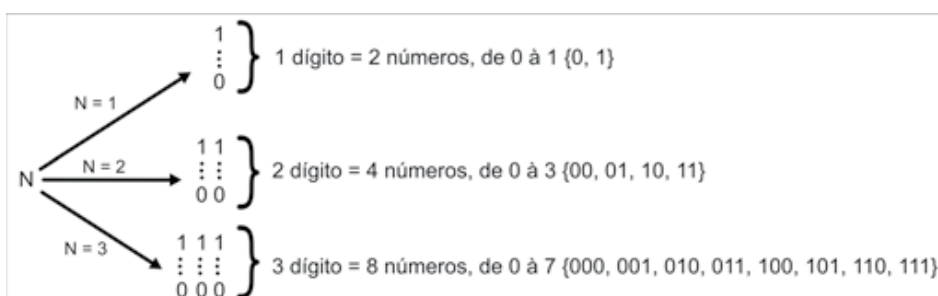


Figura 40 – Quantidade de números representáveis na base binária com N dígitos

Você percebeu que com a base binária ocorre algo semelhante ao que tínhamos observado na base decimal?

A cada novo dígito que incluímos multiplicamos o total de números representáveis por 2, ou seja, da mesma forma que na base decimal, a cada novo dígito que incluímos multiplicamos o total de números representáveis pela quantidade de símbolos da base adotada.

Observe ainda que, tanto na base decimal quanto na base binária, como estamos representando apenas números sem sinal, ou seja, apenas números positivos, o menor número representável foi sempre o zero e o maior foi sempre igual a β^{N-1} , onde β = base numérica adotada e N é a quantidade de dígitos utilizados.

Verifique por você mesmo... Quantos números você pode formar na base decimal com 5 dígitos? E na base binária, quantos números você poderia formar com os mesmos 5 dígitos?

Por fim, vamos agora observar o que acontece quando utilizamos a base binária para representar números com sinal, ou seja, números positivos e negativos.

Como você deve estar lembrado, tanto no padrão de Magnitude e Sinal quanto no padrão de Complemento a Dois, o bit mais significativo, o MSB, deve ser reservado para representação do sinal do número. Ou seja, dos N bits que temos apenas N-1 são efetivamente utilizados para representar a magnitude ou o valor do número.

Observe as Figuras 41 e 42 a seguir.

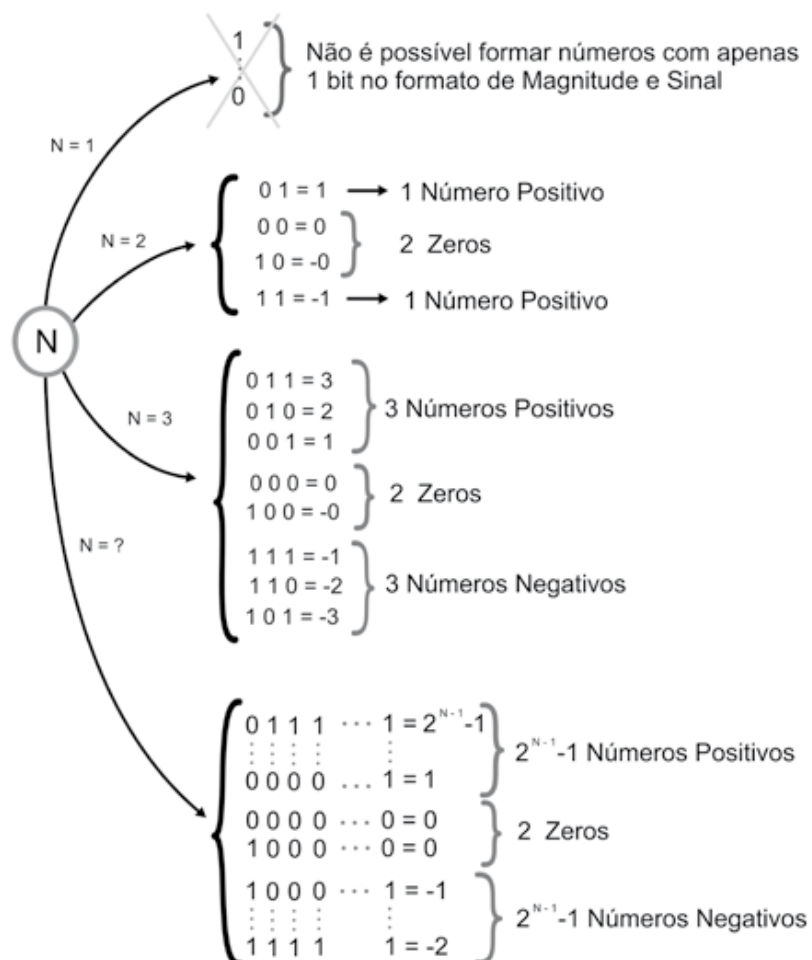


Figura 41 – Quantidade de números representáveis com N dígitos no padrão de Magnitude e Sinal

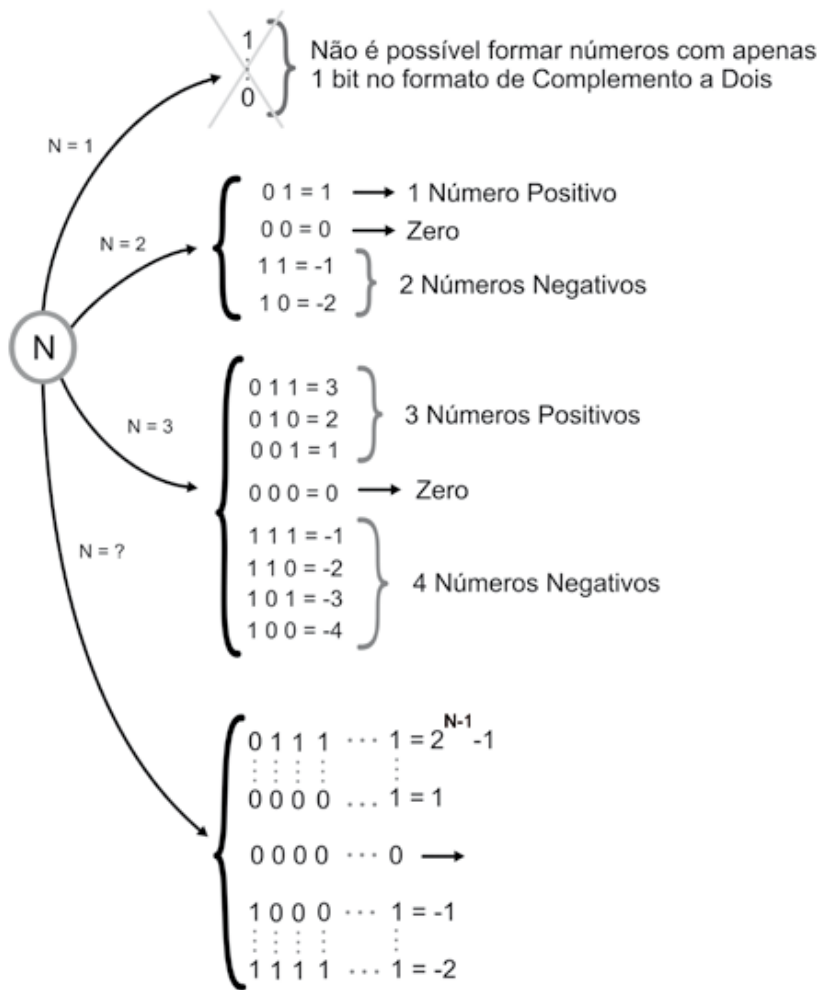


Figura 42 – Quantidade de números representáveis com N dígitos no padrão de Complemento a Dois

Você conseguiu perceber que tanto no padrão de Magnitude e Sinal como de Complemento a Dois o total de números gerados continua sempre igual a 2^N ?

Observe com bastante atenção, este não é um detalhe fácil de ser percebido. Veja por exemplo no padrão de Complemento a Dois, na última parte da Figura 42. Se somarmos a quantidade de números positivos, a quantidade de números negativos e o zero teremos:

$$2^{N-1} - 1 + 2^{N-1} + 1 = 2^N$$

Agora tente fazer o mesmo para o padrão de Magnitude e Sinal e veja se você consegue confirmar este mesmo resultado.

Para finalizar a nossa análise, temos na tabela da Figura 43 a seguir um comparativo entre as representações de números com sinal e sem sinal para uma palavra de 4 bits.

Representação com Sinal						Representação sem Sinal		
Notação de Complemento a Dois			Notação de Magnitude e Sinal			Binário sem Sinal		
$2^{(N-1)-1}$ Números Positivos	0111	7	$2^{(N-1)-1}$ Números Positivos	0111	7	$2^{(N)-1}$ Números Positivos	1111	15
	0110	6		0110	6		1110	14
	0101	5		0101	5		1101	13
	0100	4		0100	4		1100	12
	0011	3		0011	3		1011	11
	0010	2		0010	2		1010	10
	0001	1		0001	1		1001	9
Zero	0000	0	Zero	0000	0		1000	8
$2^{(N-1)}$ Números Negativos	1111	-1	- Zero	1000	-0		0111	7
	1110	-2	$2^{(N-1)-1}$ Número Negativos	1001	-1		0110	6
	1101	-3		1010	-2		0101	5
	1100	-4		1011	-3		0100	4
	1011	-5		1100	-4		0011	3
	1010	-6		1101	-5		0010	2
	1001	-7		1110	-6		0001	1
	1000	-8		1111	-7		Zero	0000

Figura 43 – Comparativo entre o poder de representação para números representados em Complemento a Dois, Magnitude e Sinal e Binário sem Sinal, para uma palavra binária com 4 bits

Para concluir nosso estudo da base binária, que tal aprendermos mais um método para encontrar o inverso de um número no padrão de Complemento a Dois? Este método que iremos aprender agora tem a vantagem de ser extremamente simples para nós, seres humanos, ainda que extremamente complexo para a Unidade Lógica e Aritmética executar, motivo pelo qual não é muito utilizado na prática.

Para converter um número em seu inverso, simplesmente execute os seguintes passos:

1. Dado um número qualquer, expresso no padrão de Complemento a Dois, comece a percorrer todos os seus bits da direita para a esquerda, parando no primeiro bit igual a 1 que encontrar.
2. Inverta o valor de todos os bits que estiverem à esquerda deste primeiro bit encontrado.

3. O resultado obtido será o inverso do número dado no padrão de Complemento a Dois. Ou seja, se o número dado for positivo o resultado será a sua representação negativa, e se este for negativo o resultado será a sua representação positiva.

Os exemplos das Figuras 44 e 45 a seguir nos demonstram como podemos aplicar este método tanto para números positivos como negativos.

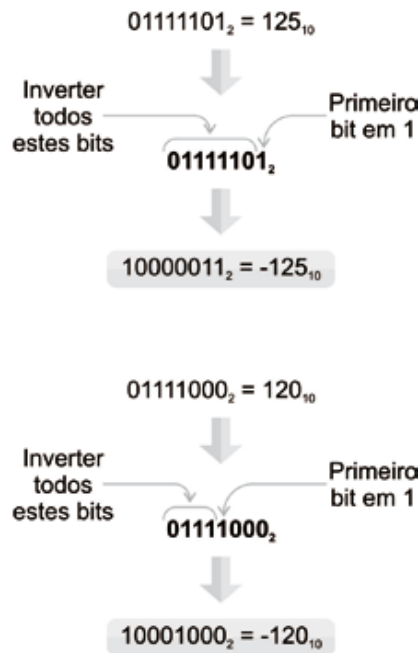


Figura 44 – Exemplos de conversão de números positivos para negativos no padrão de Complemento a Dois pelo método simplificado

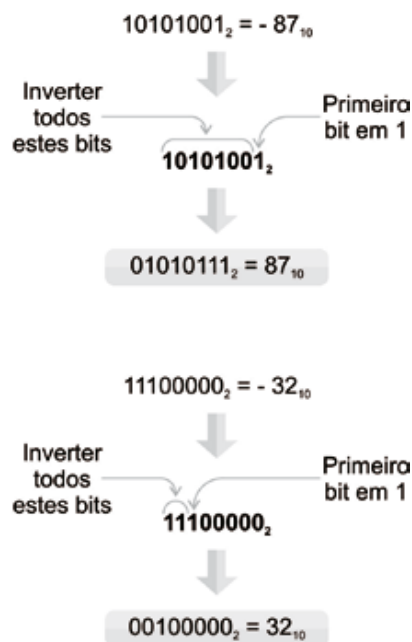


Figura 45 – Exemplos de conversão de números negativos para positivos no padrão de Complemento a Dois pelo método simplificado

Para concluir nosso estudo da base binária, nos falta ainda responder nossa terceira e última questão: “Quantos dígitos, ou bits, precisamos para representar um número R qualquer?”

A resposta desta questão está intimamente ligada ao que acabamos de estudar, no tocante ao poder de representação da base binária.

Como acabamos de aprender, se adotarmos uma palavra com N bits poderemos representar números entre 0 e 2^N-1 no padrão binário sem sinal e entre -2^{N-1} e $2^{N-1}-1$ no padrão de Complemento a Dois.

Desta forma, para definir quantos bits, ou qual o N que precisamos adotar para representar um número R qualquer na base binária precisamos simplesmente resolver a seguinte equação.

$$2^N \geq R, \text{ ou seja: } N \geq \log_2 R$$

Acompanhe os exemplos a seguir:

1. Quantos bits, no mínimo, seriam necessários para representar o número 237 na base binária sem sinal?

Como $\log_2 237 = 7,9$ teremos que utilizar uma palavra com 8 bits

2. Quantos bits, no mínimo, seriam necessários para representar o número 87 na base binária sem sinal?

Como $\log_2 87 = 6,44$ teremos que utilizar no mínimo 7 bits

3. Quantos bits, no mínimo, seriam necessários para representar o número 87 na base binária no padrão de Complemento a Dois?



Lembrete

Lembre-se que, no padrão de Complemento a dois e de Magnitude e Sinal além dos bits normalmente utilizados, nós precisamos incluir um bit a mais para representar o sinal do número. Desta forma, como já havíamos calculado que seriam necessários 7 bits para representá-lo sem sinal, precisaremos, então, de 8 bits para representá-lo no padrão de Complemento a Dois.

4. Quantos bits, no mínimo, seriam necessários para representar o número -35 na base binária no padrão de Complemento a Dois?

Como $\log_2 35 = 5,13$, e como temos que incluir um bit a mais para

representar o sinal do número, teríamos que utilizar no mínimo 7 bits.

Com isto concluímos nosso estudo da base binária. Acompanhe agora os exercícios propostos no seu Caderno de Exercícios para fixar o que você aprendeu.

2.13 Bases Octal e Hexadecimal

Conforme dissemos no início do nosso estudo, as bases numéricas são definidas a fim de simplificar o dia a dia de quem as utiliza.

A base binária, por exemplo, é extremamente prática para ser empregada na Unidade Lógica e Aritmética dos computadores, pois permite implementar circuitos simples e eficazes.

Por outro lado, esta mesma base binária é extremamente confusa para nós, seres humanos. Nossa mente está melhor adaptada a trabalhar com a informação em um grau maior de abstração.

Desta forma, a fim de simplificar o trabalho com valores originalmente expressos na base binária, foram definidas duas outras bases numéricas, as bases octal e hexadecimal.

Estas bases oferecem a vantagem de permitir o mapeamento, ou conversão, direta dos valores expressos na base binária. Desta forma, a conversão entre a base binária e as bases octal e hexadecimal, e vice versa, ocorre de maneira simples e natural.

Conforme vimos, a base octal possui apenas oito símbolos, {0, 1, 2, 3, 4, 5, 6, 7}. E, como vimos no nosso estudo da base binária, para representar valores entre zero e sete precisamos exatamente de 3 bits, ou dígitos, binários. Ou seja, para fazermos a conversão entre a base binária e a base octal basta que agrupemos os bits utilizados em palavras de 3 bits, substituindo-os pelo seu valor equivalente pela notação posicional. A conversão da base octal para a base binária se dá exatamente pelo caminho inverso, ou seja, simplesmente substituímos cada símbolo da base octal pelo seu equivalente na base binária. O único cuidado que precisamos ter nesta conversão é que cada símbolo da base octal equivale a 3 bits da base binária e vice-versa.

Observe os exemplos de conversão apresentados na Figura 46 a seguir:

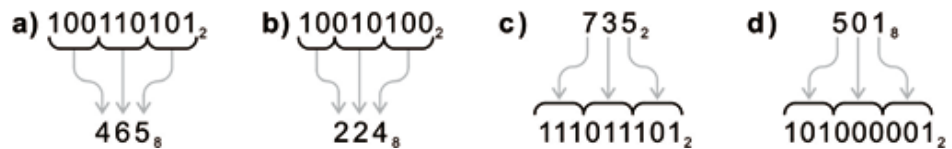


Figura 46 – Exemplos de conversões entre a base binária e a base octal e vice-versa

Muito simples, não é mesmo?

Com a base hexadecimal não é diferente. Como esta base possui exatamente 16 símbolos, {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}, podemos associar cada um dos seus símbolos a 4 bits, ou um **nibble**, da base binária.

A conversão entre a base binária e a base hexadecimal se dá exatamente da mesma forma que acabamos de ver com a base octal. Veja alguns exemplos na Figura 46 a seguir:

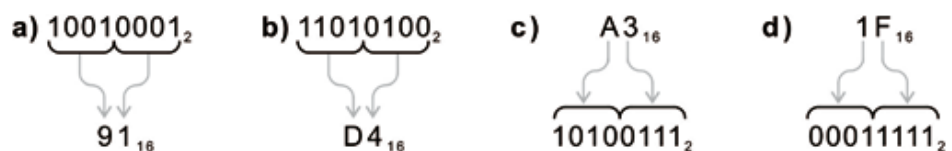
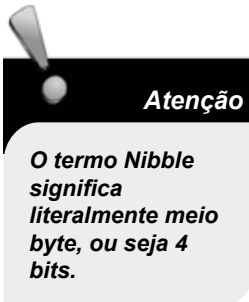


Figura 47 – Exemplos de conversões entre a base binária e a base hexadecimal a vice-versa

Atualmente, a base octal se encontra praticamente em desuso, havendo quase uma hegemonia da base hexadecimal. Mesmo assim, devido ao seu valor histórico, é muito importante que aprendamos a operar com números tanto na base octal quanto na base hexadecimal.

A fim de evitar ambiguidades, todas as linguagens de programação utilizam algum tipo de indicação para identificar a base numérica adotada em determinado número. Normalmente utiliza-se um prefixo ou um sufixo no lugar do subscrito que temos adotado em nosso texto. Os padrões mais utilizados são:

- » Base Hexadecimal: utiliza-se o prefixo “0x” antes do número. Por exemplo: 0xA3, 0x37, 0xAA35, e assim por diante.
- » Base Octal: utilizam-se apenas a letra “O” como prefixo. Por exemplo: O35, O71, O11, e assim por diante.
- » Base Binária: A maioria das linguagens existentes atualmente não dá suporte direto a dados expressos na base binária. Desta forma, o artifício mais comum, quando necessário, é lançar mão



do mapeamento existente entre esta base e as bases octal e hexadecimal.

Conversão das bases Octal e Hexadecimal para Decimal

A conversão de números escritos tanto da base octal quanto da base hexadecimal para a base decimal se dá exatamente da mesma forma como fizemos na base binária, ou seja, pela aplicação da notação posicional. Para tanto, utilizamos o somatório visto na Figura 12, apenas lembrando que devemos substituir o β pelo valor correspondente à base adotada. Devemos lembrar também que as letras utilizadas como símbolos da base hexadecimal possuem valor numérico, ou seja, devemos lembrar que na base hexadecimal o símbolo A equivale a 10, o B equivale a 11, o C equivale a 12, o D equivale a 13, o E equivale a 14 e o F equivale a 15.

Vejam os exemplos a seguir:

a) $0x36 = 3 \times 16^1 + 6 \times 16^0 = 54$

b) $0xAA = 10 \times 16^1 + 10 \times 16^0 = 160 + 10 = 170$

c) $0x378 = 3 \times 16^2 + 7 \times 16^1 + 8 \times 16^0 = 888$

d) $O35 = 3 \times 8^1 + 5 \times 8^0 = 64$

e) $O77 = 7 \times 8^1 + 7 \times 8^0 = 31$

f) $O121 = 1 \times 8^2 + 2 \times 8^1 + 1 \times 8^0 = 81$

Conversão da base Decimal para as bases Octal e Hexadecimal

As conversões da base decimal para as bases octal e hexadecimal podem ser efetuadas de duas maneiras. A primeira é através de uma variação do algoritmo apresentado para a conversão da base decimal para a binária, que denominamos de técnica por divisões sucessivas. A segunda é convertendo o número primeiramente para binário e depois fazendo a conversão direta para a base desejada, octal ou hexadecimal.

Como todos já sabemos converter da base decimal para a base binária, vamos demonstrar apenas como converter utilizando a variação do algoritmo de divisões sucessivas. Vamos começar pela base octal. Observe os exemplos da Figura 48 a seguir.

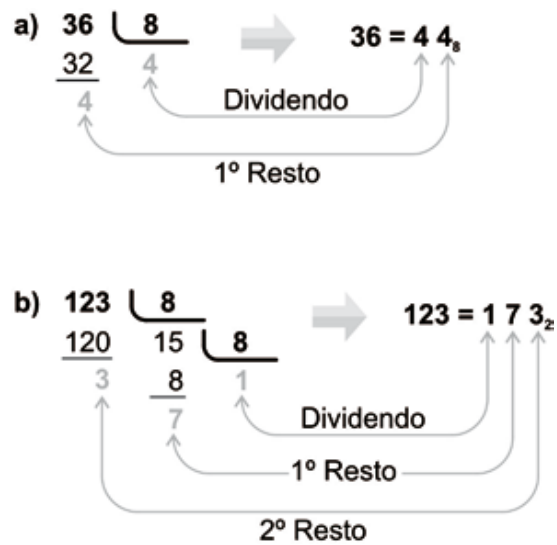


Figura 48 – Exemplos de conversão da base decimal para a base octal

Agora vamos ver como fica a conversão para a base hexadecimal. Observe a Figura 49 a seguir.

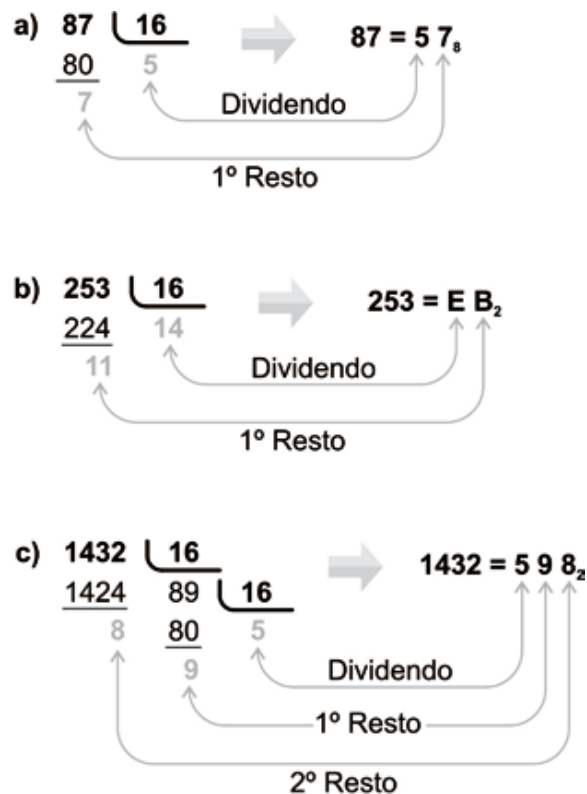


Figura 49 – Exemplos de conversão da base decimal para a base hexadecimal

Muito simples, não é mesmo?

Agora que já sabemos como converter e representar números nas bases octal e hexadecimal, nos falta apenas ver como ficam as operações de adição e subtração nestas bases.

A primeira coisa que devemos ter em mente ao pensar na execução das operações aritméticas nas bases octal e hexadecimal é que estas bases nada mais são do que uma outra forma de ver a base binária. Ou seja, que estas bases na verdade apenas apresentam os valores obtidos na base binária de uma forma mais agradável para o ser humano.

A operação de adição permanece exatamente como fazemos na base decimal, com a única diferença com o valor no qual ocorre o “vai um”, que em octal ocorre quando o resultado parcial for maior que 7 e em hexadecimal quando o resultado parcial for maior que 15.

Para a operação de subtração, por sua vez, devemos levar em conta todas as considerações feitas para a base binária, ou seja, que na Unidade Lógica e Aritmética dos computadores todas as operações de subtração são substituídas por operações de adição entre o primeiro operando e o inverso do segundo operando.

Pelos motivos já expostos, mesmo nas bases octal e hexadecimal adotamos o formato de Complemento a Dois para representar o inverso de um número.

Assim sendo, a estratégia mais interessante para as operações de subtração nas bases octal e hexadecimal, quando os operandos já não estiverem expressos no formato de Complemento a Dois, deverá ser:

- » Primeiramente converter os operandos para a base binária;
- » Transformar o subtrator para o seu inverso, em Complemento a Dois, por uma das técnicas apresentadas;
- » Efetuar a operação diretamente na base binária;
- » Converter os resultados obtidos para a base desejada.

Observe os exemplos da Figura 50 a seguir. No exemplo a, temos uma operação entre operandos expressos na base octal. No exemplo b, temos uma operação com operandos expressos na base hexadecimal.

Preste atenção ao exemplo b. Perceba que ao operarmos com a base hexadecimal temos que tomar cuidado não apenas com o valor no qual ocorre o “vai um”, mas também nas conversões entre os resultados parciais encontrados e os símbolos a serem utilizados.

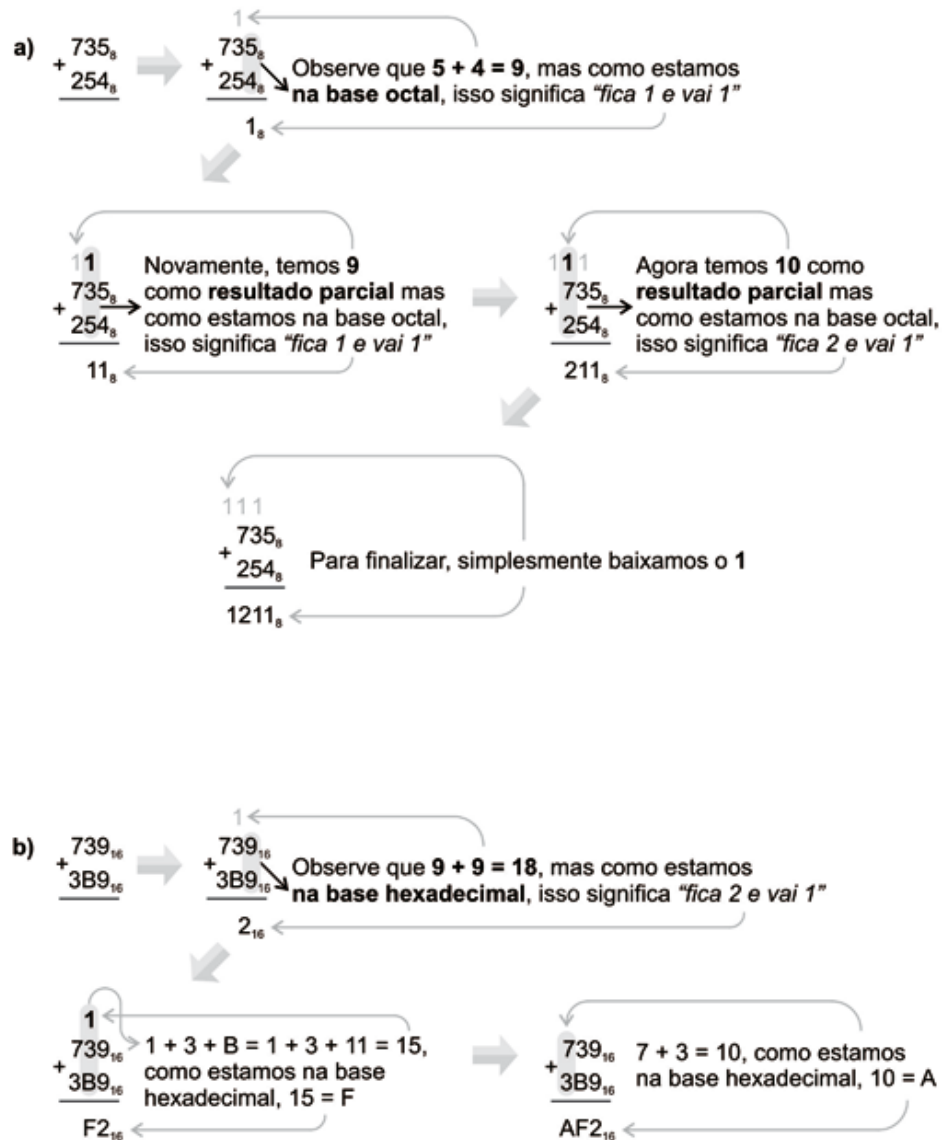


Figura 50 – Exemplos de operações nas bases octal a hexadecimal

Com isto, encerramos nosso estudo sobre bases numéricas.

Esperamos que tenham gostado. Este é um conhecimento extremamente útil para todos aqueles que pretendem entender um pouco mais como um computador realmente funciona.

Para os que se desejarem se aprofundar um pouco mais neste assunto, recomendamos a leitura dos seguintes trabalhos científicos: "What every computer scientist should know about floating-point arithmetic.pdf" disponível em <<http://dlc.sun.com/pdf/800-7895/800-7895.pdf>> e "Where did all my decimals go.pdf" disponível em <<http://jetlib.com/mirrors/test/uvu.freshsources.com/page1/page2/page5/files/decimals.pdf>>, além de acompanharem o conteúdo disponível na bibliografia recomendada.



Vamos Revisar?

Neste capítulo, aprendemos sobre as Bases Numéricas e sobre a Notação Posicional e a sua importância para o desenvolvimento dos computadores digitais.

Aprendemos também que as duas bases numéricas mais importantes são:

- » A Base Decimal, formada por dez símbolos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, utilizada por nós, seres humanos.
- » A Base Binária, formada apenas pelos símbolos 0 e 1, utilizada pela Unidade Lógica e Aritmética dos Computadores.

Além destas bases, conhecemos também as bases Octal e Hexadecimal, que foram criadas com o intuito de facilitar a visualização e a operação de valores expressos na base binária.

Também pudemos aprender diversas técnicas e métodos que podem ser empregados na conversão de valores expressos em diferentes bases numéricas. Com isto, pudemos constatar que todos os valores inteiros podem ser expressos em qualquer das bases numéricas estudadas, e que, por outro lado, nem todo valor decimal pode ser livremente convertido, uma vez que alguns destes tornam-se dízimas periódicas quando passados de uma base para outra.

Um outro ponto muito importante que aprendermos foi que as operações aritméticas são integralmente preservadas em todas as bases estudadas.



Capítulo 3

O que vamos estudar?

Neste capítulo, vamos estudar os seguintes temas:

- » Portas lógicas e funções lógicas
- » Álgebra booleana
- » Circuitos digitais

Metas

Após o estudo deste capítulo, esperamos que você consiga:

- » Conhecer as principais portas lógicas utilizadas na construção dos circuitos digitais básicos.
- » Construir circuitos digitais combinatórios utilizando tabela verdade e vice versa.

Capítulo 3 – Lógica Digital



Vamos conversar sobre o assunto?

Você já parou para pensar como são projetados e construídos os componentes básicos que dão inteligência a um computador? Como são projetados os circuitos integrados digitais?

Esta é uma ciência fascinante e em franca expansão, e que está na base de todo o desenvolvimento tecnológico que vivenciamos em nosso dia a dia. Desde um simples relógio de pulso até ao computador mais avançado, tudo o que nos rodeia está repleto de lógica digital.

A expressão “Lógica Digital” traz consigo duas definições carregadas de significado:

- ▶ **Lógica:** O termo lógica é comumente definido como sendo a ciência das leis do pensamento, ou ainda, a disciplina que se dedica ao estudo das regras que permitem tornar o pensamento válido.
- ▶ **Digital:** O termo digital se refere à representação da informação através de dois estados ou símbolos lógicos básicos, o falso e o verdadeiro, o zero e o um. Quando dizemos que um determinado sistema é digital, em outras palavras estamos dizendo que seu funcionamento é baseado nestes dois estados ou símbolos lógicos básicos: o zero e o um.

Desta forma, podemos entender a lógica digital como sendo a tecnologia que nos permite modelar a informação e o raciocínio lógico através de zeros e uns.

A lógica digital que iremos trabalhar é baseada em uma área especial da matemática conhecida como **álgebra de Boole**, ou Álgebra Booleana.

A álgebra booleana é muito parecida com a álgebra que aprendemos na escola, a qual está baseada em operadores e/ou funções com a única diferença que na álgebra booleana, independente da função que você esteja utilizando, tanto os valores de entrada como o resultado obtido serão sempre zeros e uns.



Saiba Mais

George Boole foi um Cientista Alemão do século 19 considerado o pai da lógica matemática.

Na lógica digital, a informação é processada através de equações ou funções booleanas, as quais são implementadas através de circuitos especiais conhecidos como portas lógicas.

Uma porta lógica nada mais é que um circuito eletrônico que implementa a funcionalidade definida por uma função booleana. Desta forma, cada porta lógica recebe o nome da função lógica que implementa.

Interessante, não é mesmo? Pois se prepare, a nossa aventura está só começando. Vamos começar estudando as principais portas lógicas e suas funcionalidades, em seguida, vamos aprender como utilizar estas portas lógicas para construir nossos próprios circuitos lógicos mais complexos.

3.1 Portas Lógicas e Funções Lógicas Básicas

Quando o famoso matemático inglês George Boole, publicou as bases de sua teoria em 1854, em um trabalho intitulado “*An Investigation of the Laws of Thought on Which to Found the Mathematical Theories of Logic and Probabilities*”, os computadores digitais ainda nem pensavam em existir. Entretanto, foi o trabalho deste matemático brilhante que serviu de alicerce de tudo o que hoje conhecemos como lógica digital.

Em seu trabalho, Boole definiu 3 funções lógicas essenciais, a partir das quais derivaram-se todas as outras. São elas:

- » A função *And*, que define que **uma resposta só será verdadeira** se, e somente se, todas as suas entradas forem verdadeiras.
- » A função *Or*, que define que uma resposta só será verdadeira se ao menos uma das sua entradas for verdadeira
- » A função *Not*, que possui apenas uma entrada e uma saída, e que define que a resposta só será verdadeira se e somente se a sua entrada for falsa.

A partir destas três funções lógicas, foram definidas mais três funções lógicas básicas as quais também são utilizadas como elementos básicos para a criação de circuitos lógicos mais complexos:

- » A função lógica *XOR*, que possui duas entradas e uma saída e

Lembrete

Lembre-se que, por convenção, em lógica digital a informação verdadeira está associada ao nível 1 e a falsa ao nível 0.

que em português poderia ser traduzida como “ou exclusivo”, a qual define que uma resposta só será verdadeira se, e somente se, uma das entradas for verdadeira e a outra falsa.

- » A função lógica *Nand*, formada a partir da associação da função lógica *And* seguida da função lógica *Not*, que define que uma resposta só será verdadeira se e somente se todas as suas entradas forem falsas
- » A função lógica *Nor*, formada a partir da associação da função lógica *Or* seguida da função lógica *Not*, que define que uma resposta só será verdadeira se ao menos uma das suas entradas for falsa.

A fim de permitir a representação da funcionalidade esperada de um circuito baseado em lógica digital, cada função/porta lógica possui um símbolo esquemático, ou diagrama, e um símbolo matemático, também conhecido como equação booleana, associado.

Apesar de utilizar operadores semelhantes aos utilizados nas equações aritméticas, as operações descritas pelas equações booleanas são em essência operações lógicas. Por este motivo, devemos tomar muito cuidado para não nos confundir no momento de avaliar uma equação booleana.

Além do símbolo esquemático e do símbolo matemático, cada porta lógica possui também uma tabela que descreve a sua funcionalidade, conhecida como Tabela Verdade. A Tabela Verdade é uma tabela na qual estão listadas todas as possíveis configurações para os sinais de entrada e a saída produzida para cada uma destas configurações pela porta lógica associada.

A Figura 1, a seguir, nos traz os símbolos matemático e gráfico e os nomes associados às seis principais funções lógicas que acabamos de apresentar.

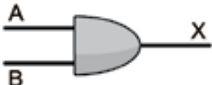

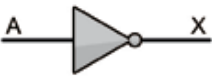

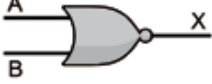

Porta lógica	Símbolo matemático	Símbolo gráfico
AND	$\cdot X = A \cdot B$	
OR	$+ X = A + B$	
NOT	$- X = \overline{A}$	
NAND	$X = \overline{A \cdot B}$	
NOR	$X = \overline{A + B}$	
XOR	$\oplus X = A \oplus B$	

Figura 1 – Portas Lógicas básicas e seus símbolos gráficos e matemáticos

Agora que já fomos apresentados às principais portas lógicas, seus símbolos gráficos e equações booleanas, que tal estudarmos um pouco mais detalhadamente cada uma destas portas e entender como elas funcionam a partir da análise das suas Tabelas Verdades.

3.2 Portas Lógicas

Porta Lógica And

Nas Figuras 2 e 3 a seguir, temos o símbolo gráfico, a equação booleana e a tabela verdade associada à porta lógica **And**.

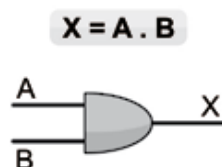


Figura 2 – Símbolo gráfico e equação booleana associados à porta lógica And

Entradas		Saída
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Figura 3 – Tabela Verdade associada à porta lógica And

Observe a tabela verdade, veja que esta reflete o que dissemos a princípio, que a porta lógica **And** fornece uma resposta verdadeira, ou “1” se, e somente se, todas as suas entradas forem verdadeiras. Muito Simples não é mesmo?

Lembre-se que nós podemos tanto utilizar a equação booleana, o diagrama esquemático ou a própria função lógica para definir que estamos querendo utilizar esta função lógica em nosso sistema, conforme podemos ver pela Figura 4 a seguir.

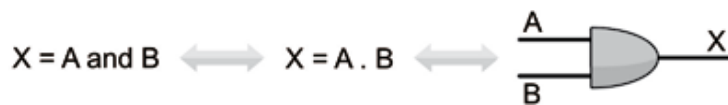


Figura 4 – Três formas equivalentes de representar a função lógica AND

Porta Lógica Or

Nas Figuras 5 e 6, a seguir, temos o símbolo gráfico, a equação booleana e a tabela verdade associadas à porta lógica **Or**.

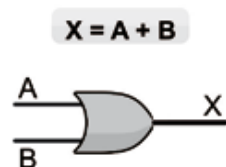


Figura 5 – Símbolo gráfico associado à porta lógica Or

Entradas		Saída
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Figura 6 – Tabela Verdade associada à porta lógica Or

Observe agora a tabela verdade da função **Or**. Veja que, conforme dissemos a princípio, a porta lógica **Or** fornece uma resposta verdadeira, ou “1”, se ao menos uma das suas entradas forem verdadeiras.

Novamente, lembre-se que nós podemos tanto utilizar a equação booleana, o diagrama esquemático ou a própria função lógica para definir que estamos querendo utilizar esta função lógica em nosso sistema, conforme podemos ver pela Figura 7 a seguir.



Figura 7 – Três formas equivalentes de representar a função lógica OR

Porta Lógica Not

Nas Figuras 8 e 9, a seguir, temos o símbolo gráfico, a equação booleana e a tabela verdade associada à porta lógica **Not**. Observe a “bolinha” utilizada na saída da porta lógica para indicar a inversão ou “negação” do sinal de entrada. Esta “bolinha” é sempre utilizada quando queremos indicar que vai ocorrer uma inversão no sinal, transformando um em zero e vice-versa.



Figura 8 – Símbolo gráfico associado à porta lógica Not, com destaque para o símbolo utilizado para indicar a inversão ou “negação” do sinal de entrada.

Entrada	Saída
A	X
0	1
1	0

Figura 9 – Tabela Verdade associada à porta lógica Not

Esta é, sem dúvida, a função lógica mais simples de todas. Se entra um sai zero e se entra zero sai um. Fácil demais, você não acha?

Diferentemente das outras funções lógicas, a função **Not** possui 4 formas de ser representada. Acompanhe na Figura 10 a seguir.

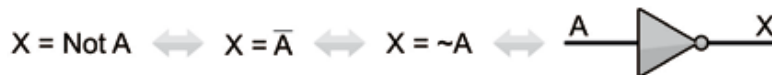


Figura 10 – Quatro formas equivalentes de representar a função lógica Not



Lembrete

Conforme dissemos a princípio, as três outras portas lógicas que apresentaremos agora foram formadas a partir de associações das três primeiras portas lógicas apresentadas. Desta forma, é importante lembrar que as suas funcionalidades são expressão da sobreposição das funcionalidades das outras portas lógicas utilizadas.

Porta Lógica Nand

Nas Figuras 11 e 12, a seguir, temos o símbolo gráfico, a equação booleana e a tabela verdade associada à porta lógica **Nand**.

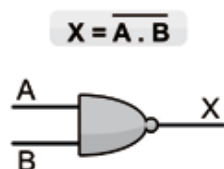


Figura 11 – Símbolo gráfico e equação booleana associados à porta lógica Nand

Entradas		Saída
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Figura 12 – Tabela Verdade associada à porta lógica Nand

É importante que percebamos que a porta **Nand** foi construída a partir de uma porta **And** associada a uma porta **Not**, conforme podemos ver na Figura 13 a seguir.

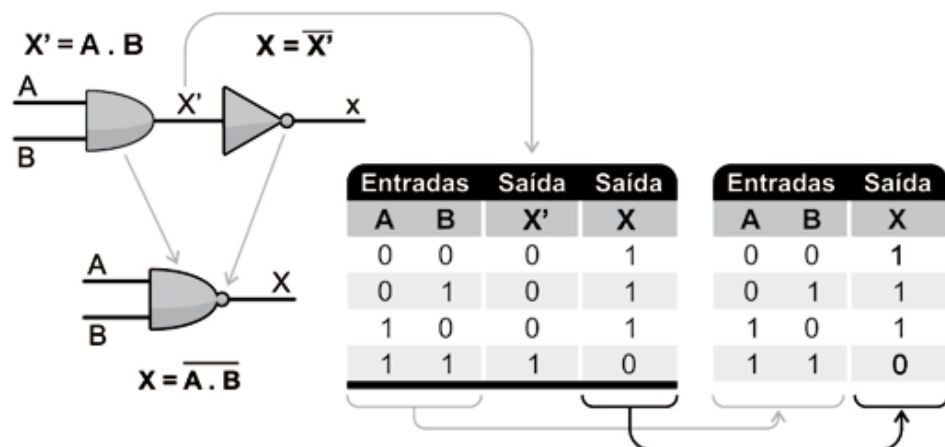


Figura 13 – Modelo esquemático de construção da Porta Nand e da sua Tabela Verdade

Observe que a tabela verdade associada à uma porta lógica indica apenas as suas entradas e saídas “visíveis”, ou seja, seguindo o exemplo da porta **Nand**, resultados intermediários obtidos por portas lógicas utilizadas internamente não são listados, conforme ocorreu com o sinal X' .

Lembre-se que nós podemos tanto utilizar a equação booleana, o diagrama esquemático ou a própria função lógica para definir que estamos querendo utilizar esta função lógica em nosso sistema, conforme podemos ver pela Figura 14 a seguir.

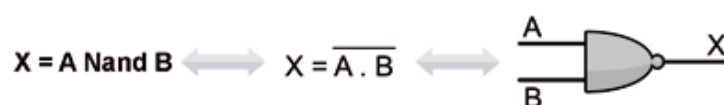


Figura 14 – Três formas equivalentes de representar a função lógica NAND

Porta Lógica Nor

Nas Figuras 15 e 16, a seguir, temos o símbolo gráfico, a equação booleana e a tabela verdade associada à porta lógica **Or**.

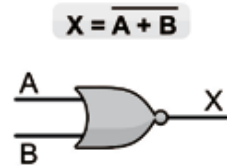


Figura 15 – Símbolo gráfico e equação booleana associados à porta lógica Nor

Entradas		Saída
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Figura 16 – Tabela Verdade associada à porta lógica Nor

Novamente, conforme fizemos com a porta **Nand**, vamos agora apresentar o esquema interno da porta **Nor**.

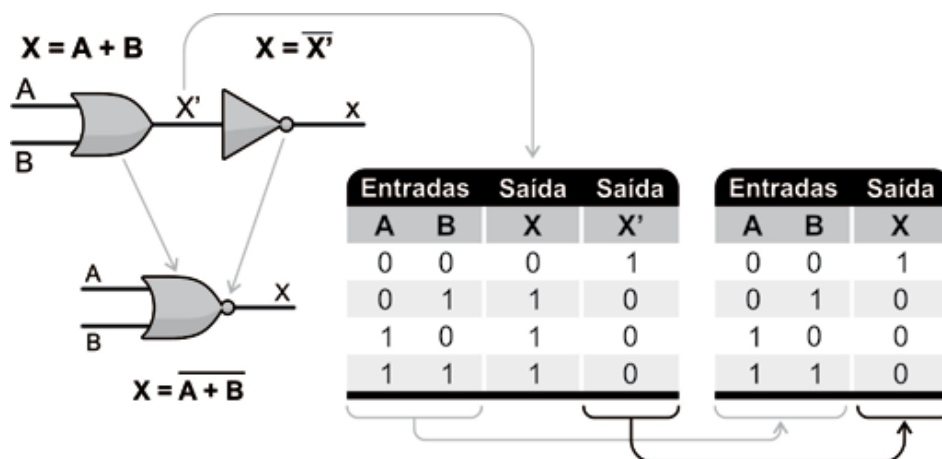


Figura 17 – Modelo esquemático de construção da Porta Nor e da sua Tabela Verdade

Por fim, temos as três formas pelas quais podemos representar uma porta lógica **Nor**.

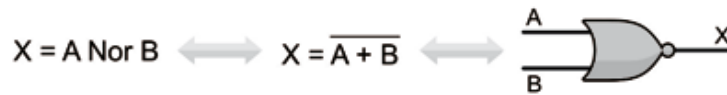


Figura 18 – Três formas equivalentes de representar a função lógica Nor

Porta Lógica Xor

Nas Figuras 19 e 20, a seguir, temos o símbolo gráfico, a equação booleana e a tabela verdade associada à porta lógica **Xor**.

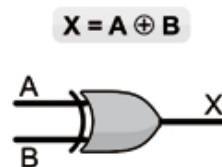


Figura 19 – Símbolo gráfico e equação booleana associados à porta lógica Xor

Entradas		Saída
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Figura 20 – Tabela Verdade associada à porta lógica Xor

Devido à sua funcionalidade, a porta **Xor** possui um esquema interno um pouco mais complexo que os das portas **Nand** e **Nor**. Observe a Figura 21 a seguir.

Como você pode perceber, a construção de uma porta **Xor** exige 2 portas **Not**, 2 portas **And** e uma porta **Or**. A tabela verdade destaca do seu lado esquerdo todos os possíveis valores de entrada e na sua última coluna, à direita, todos os valores de saída. Nas colunas marcadas como etapas intermediárias temos as possíveis entradas e saídas das portas utilizadas internamente na construção da porta **Xor**. Observe que, segundo a tabela verdade tanto da Figura 12 quanto da Figura 13, a porta **Xor** só fornece um a sua saída se, e somente se, uma das entradas for um e a outra zero.

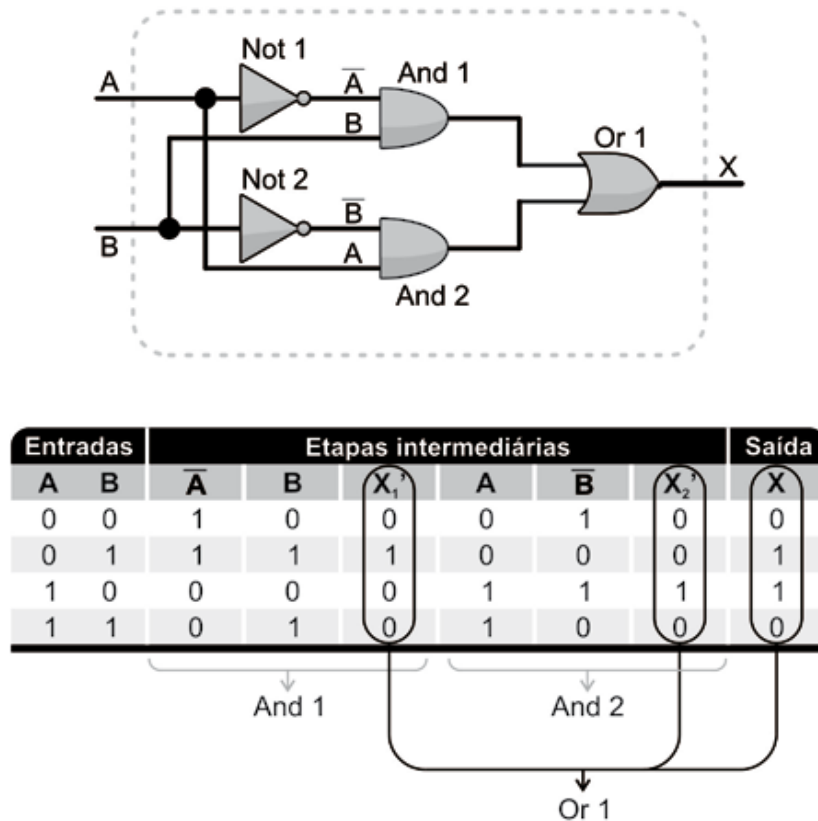


Figura 21 – Modelo esquemático de construção da Porta Xor e da sua Tabela Verdade

Por fim, temos as três formas pelas quais podemos representar uma porta lógica **Xor**.



Figura 22 – Três formas equivalentes de representar a função lógica Xor

Agora que já conhecemos as portas lógicas básicas, suas funcionalidades, seus diagramas esquemáticos e suas equações booleanas, podemos nos aventurar e ir um pouco mais longe, podemos começar a pensar em projetar nossos próprios circuitos lógicos digitais.

3.3 Circuitos Digitais

Assim como as portas lógicas, os circuitos digitais são circuitos eletrônicos projetados com o intuito de efetuar uma função, ou equação, booleana específica. Desta forma, os circuitos digitais também possuem um diagrama esquemático, uma equação booleana

e uma tabela verdade associada.

Como qualquer projeto de engenharia, o projeto de um circuito digital exige que se adote uma metodologia pré-definida de forma a poder garantir o melhor resultado dispendendo o mínimo de recursos e esforços possíveis.

Desta forma, a fim de atingir estes objetivos, em cada um dos nossos projetos, seguiremos o passo a passo listado a seguir:

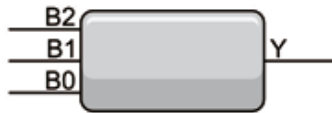
1. Analisar atentamente as funcionalidades esperadas do circuito a ser implementado, buscando identificar e eliminar possíveis redundâncias que, se removidas, possam simplificar o circuito a ser implementado.
2. Identificar e listar de maneira agrupada todos os sinais de entrada e saída do circuito
3. Definir a tabela verdade equivalente a funcionalidade esperada.
4. A partir da tabela verdade, deduzir a equação booleana geral do circuito.
5. Implementar um circuito digital que atenda a equação booleana encontrada, substituindo as expressões presentes na equação booleana por suas portas lógicas equivalentes.

Vamos começar com um pequeno exemplo. Vamos projetar um circuito com uma entrada binária de 3 bits e uma saída de 1 bit. De tal forma que este seja capaz de identificar se o valor aplicado a entrada é par, ou seja, que sinalize esta ocorrência colocando 1 a saída quando o valor aplicado à entrada for par e 0 quando for ímpar.

Aplicando a metodologia proposta temos:

1. Analisando a funcionalidade esperada do circuito, podemos constatar que não existem redundâncias e inconsistências ou incoerências.
2. Como determinado em sua especificação, nosso circuito terá 3 bits de entrada que denominaremos de B0, B1 e B2 e um bit de saída que denominaremos simplesmente de Y.
3. Na Figura 24 temos a tabela verdade do nosso circuito. Observe que nossa tabela é exatamente igual as das portas lógicas que estudamos. Temos um grupo de colunas representando os sinais de entrada e uma coluna representando o sinal de saída. Nesta tabela verdade, apenas para facilitar o acompanhamento das funcionalidades esperadas, foi incluída uma coluna a mais, a coluna N, com o valor decimal representado pelos 3 bits de

entrada(B2,B1 e B0). Verifique que, conforme definido nas funcionalidades do circuito, a coluna Y é 1 sempre que o valor indicado na coluna N é par. Como estamos querendo identificar quando o valor da entrada é par, nosso sinal Y é válido, ou seja, é 1, justamente quando o valor aplicado à entrada for 0, 2, 4 ou 6.



N	Entradas			Saída
	B2	B1	B0	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Saídas válidas (0, 2, 4, 6)

Figura 23 – Diagrama esquemático e tabela verdade do circuito digital que sinaliza quando o valor aplicado às entradas é par

Continuando, precisamos agora deduzir a equação booleana correspondente à funcionalidade esperada. Observe a Figura 24 a seguir.

N	Entradas			Saída
	B2	B1	B0	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Y = 1 quando B2 = 0 e B1 = 0 e B0 = 0, ou seja: $Y = B2 \cdot B1 \cdot B0$

Y = 1 quando B2 = 0 e B1 = 0 e B1 = 0, ou seja: $Y = B2 \cdot B1 \cdot B0$

Y = 1 quando B2 = 0 e B1 = 0 e B1 = 0, ou seja: $Y = B2 \cdot B1 \cdot B0$

Y = 1 quando B2 = 0 e B1 = 0 e B1 = 0, ou seja: $Y = B2 \cdot B1 \cdot B0$

Figura 24 – Levantamento das Equações booleanas parciais, por linhas, do circuito dado

Veja que primeiramente construímos as equações booleanas parciais, uma para cada linha da tabela verdade onde Y =1. Nosso objetivo ao descrever estas equações booleanas parciais é identificar as condições de entrada que tornam a nossa saída válida.

Observe ainda que, como cada equação booleana parcial é baseada apenas nos sinais de entrada e nos funções booleanas básicas, precisamos adotar algum artifício para sinalizar quando o sinal da entrada é válido em um ou zero, uma vez que queremos indicar claramente a condição encontrada na tabela verdade que nos gerou uma saída válida.

Desta forma, a fim de fazer esta distinção, sempre que um sinal é válido em um, ou seja, sempre que este deve ter seu valor igual a um para gerar uma saída válida, simplesmente anotamos o nome do próprio sinal na equação booleana, caso contrário, ou seja, quando este deve ter o valor igual a zero para gerar uma saída válida, anotamos o sinal com uma barra sobre o seu nome.

Observe a primeira linha da nossa tabela verdade. Veja que temos $Y=1$ quando $B2=0$, $B1=0$ e $B0=0$. Ou seja, que Y é igual a um quando a entrada $B2$ e a entrada $B1$ e a entrada $B0$ forem zero. Observe que as conjunções e que utilizamos podem ser substituídas por funções booleanas **and**, a qual nos dá uma resposta verdadeira, igual a 1, quando todas as suas entradas forem verdadeiras. Desta forma, a nossa equação booleana parcial fica conforme podemos ver na primeira linha da nossa tabela verdade.

As demais linhas válidas da tabela verdade seguem o mesmo raciocínio.

Terminado o levantamento das equações parciais, precisamos definir a equação booleana geral do nosso circuito. Para tanto, basta observarmos que o nosso circuito deve fazer $Y=1$ se ocorrer qualquer uma das quatro condições que foram identificadas. Em outras palavras, Y será igual a um se ocorrer a primeira ou a segunda ou a terceira ou a quarta condição identificada. Como as expressões **ou** aqui utilizadas equivalem a função booleana **Or**, podemos construir nossa equação booleana geral simplesmente conectando todas as nossa equações parciais já encontradas através destas funções booleanas **Or**. Observe a Figura 25 a seguir.

Por fim, nos falta apenas construir um circuito que implemente a equação que acabamos de levantar.

N	Entradas			Saída	
	B2	B1	B0	Y	
0	0	0	0	1	$Y = \overline{B2} \cdot \overline{B1} \cdot \overline{B0}$
1	0	0	1	0	
2	0	1	0	1	
3	0	1	1	0	
4	1	0	0	1	$Y = B2 \cdot \overline{B1} \cdot \overline{B0}$
5	1	0	1	0	$Y = B2 \cdot B1 \cdot \overline{B0}$
6	1	1	0	1	
7	1	1	1	0	

Equação geral do circuito

$$Y = \overline{B2} \cdot \overline{B1} \cdot \overline{B0} + \overline{B2} \cdot B1 \cdot \overline{B0} + B2 \cdot \overline{B1} \cdot \overline{B0} + B2 \cdot B1 \cdot \overline{B0}$$

Figura 25 – Construção da Equação Booleana Geral do Circuito

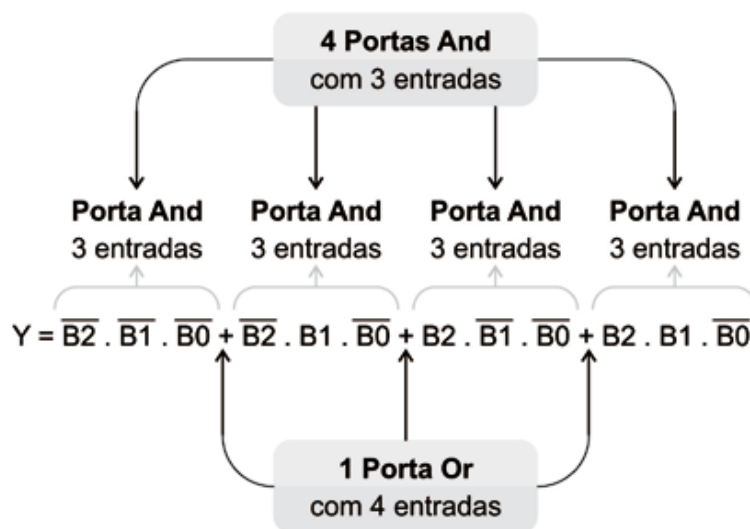


Figura 26 – Conversão da Equação Booleana Geral em portas lógicas equivalentes

Como podemos ver pela Figura 26, analisando a equação geral do circuito, fica fácil perceber que podemos converter cada parte da nossa equação geral em portas lógicas básicas. Primeiramente, cada equação parcial deve ser convertida em uma porta **And** com tantas entradas quantas forem as entradas do circuito, em seguida, deve-se interligar as saídas de cada uma destas portas **And** através de uma porta **Or** com tantas entradas quantas forem as equações parciais do circuito. A Figura 27, a seguir, nos traz o esquema elétrico final do circuito digital proposto.

Um pouco trabalhoso, mas muito interessante não é mesmo? Saiba que por simples que possa parecer esta ainda é uma das técnicas mais utilizadas para o desenvolvimento de circuitos digitais simples.

Existem técnicas mais avançadas de projeto, que permitem a partir do resultado obtido com a técnica aqui apresentada otimizar tanto a

equação booleana quanto o circuito elétrico final obtido, mas estas técnicas fogem ao escopo de nossa disciplina.

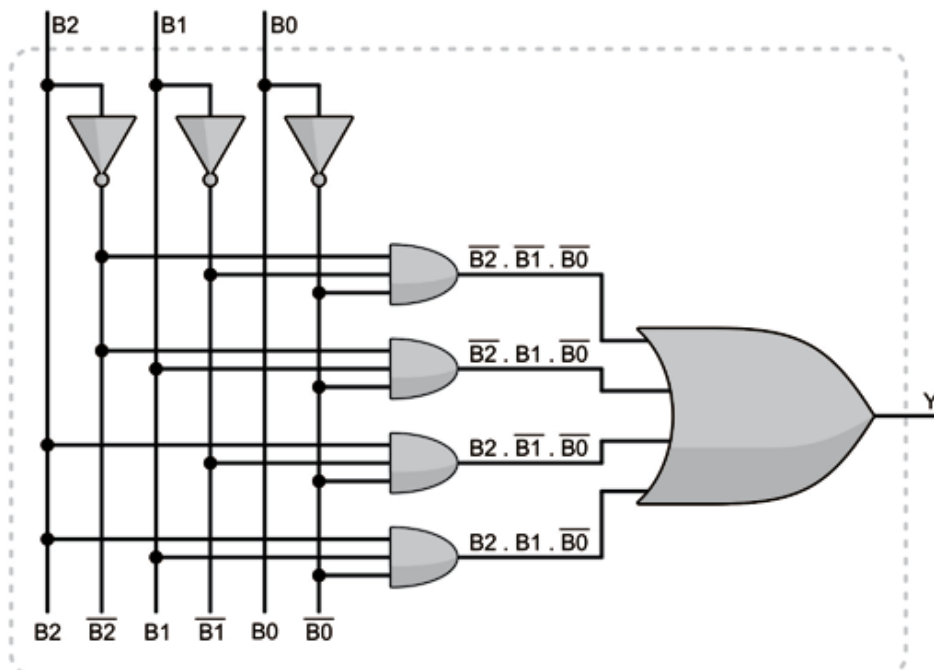


Figura 27 – Circuito digital que implementa a função de identificar se um valor binário aplicado às entradas B2, B1 e B0 é par

Que tal exercitar um pouco mais implementando um outro circuito? Desta vez vamos projetar um circuito que identifique quando o valor aplicado à entrada é divisível por 3.

Para não tomar muito tempo, vamos apenas apresentar as fases do projeto, cabe a você chegar às conclusões já apresentadas.

N	Entradas			Saída	
	B2	B1	B0	Y	
0	0	0	0	1	$Y = \overline{B2} . \overline{B1} . \overline{B0}$
1	0	0	1	0	
2	0	1	0	0	$Y = \overline{B2} . B1 . B0$
3	0	1	1	1	
4	1	0	0	0	$Y = B2 . B1 . \overline{B0}$
5	1	0	1	0	
6	1	1	0	1	
7	1	1	1	0	

$$Y = \overline{B2} . \overline{B1} . \overline{B0} + \overline{B2} . B1 . B0 + B2 . B1 . \overline{B0}$$

Figura 28 – Tabela Verdade, equações parciais e equação geral do circuito que identifica quando a entrada é divisível por 3

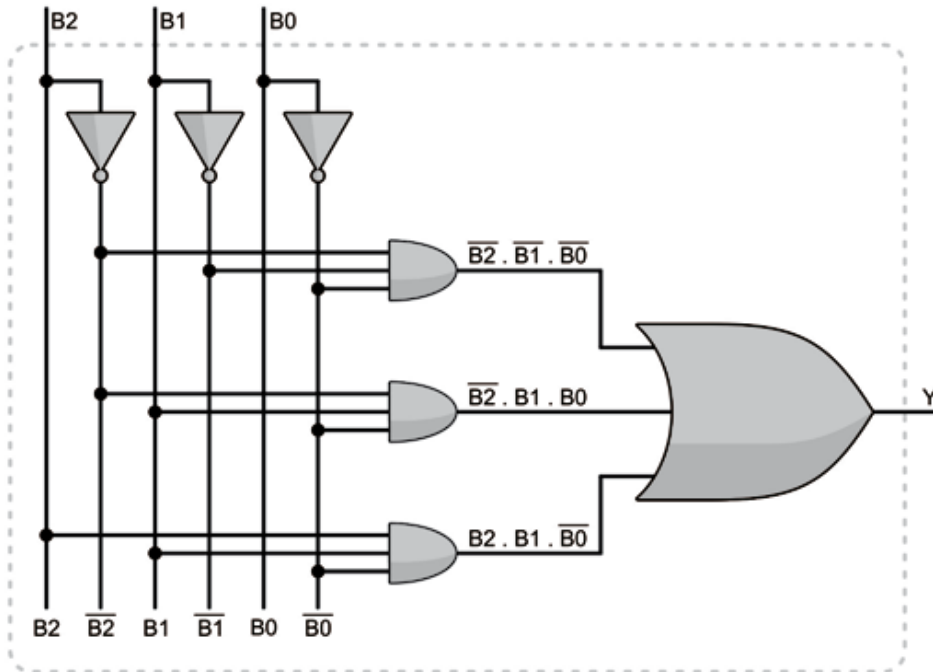


Figura 29 – Circuito digital que identifica quando a entrada é divisível por 3

Muito fácil, não é mesmo?

Estes circuitos que acabamos de implementar são conhecidos como **circuitos combinacionais**, uma vez que sua resposta depende exclusivamente de uma combinação adequada de valores de entrada e nada mais.

Por outro lado, existe também outra classe de circuitos conhecidos como **circuitos sequenciais**.

Os circuitos sequenciais são aqueles que alteram a sua resposta aos valores de entrada a depender do momento em que estes valores ocorrem.

Como sua resposta depende não apenas dos valores de entrada, mas também do momento em que estes ocorrem, estes circuitos possuem alguns sinais extras que servem para sincronizar sua operação com os outros circuitos que o cercam. Dentre estes sinais temos o sinal de Reset, que serve para inicializar a lógica interna do circuito, e o sinal de clock, ou sinal de relógio, que permite ao circuito contar o tempo decorrido entre a ocorrência de padrões de sinais de entrada de tal forma a permitir a sua correta identificação.

Podemos ver os circuitos sequenciais como uma “evolução” dos circuitos combinacionais, uma vez que estes devem não apenas responder corretamente a determinados padrões de sinais de entrada,

mas também avaliar corretamente o momento em que estes sinais ocorrem.

Os sistemas computacionais, sejam eles complexos como um computador de grande porte, ou simples, como um chip de celular, todos eles podem ser descritos como circuitos sequenciais.

Mais tarde quando estudarmos os diversos circuitos presentes na arquitetura de um computador voltaremos a nos referir aos circuitos sequenciais e suas aplicações.

Por enquanto, ficaremos por aqui.

Não deixe de verificar os exercícios resolvidos e tentar resolver os exercícios propostos no seu caderno de exercícios.



Aprenda Praticando

Agora que você já aprendeu a teoria e já viu como resolver os exercícios com os exemplos apresentados, chegou a hora de praticar resolvendo os exercícios da seção 3 do caderno de exercícios.

Lembre-se, é de suma importância que você tente resolver sozinho todos os exercícios apresentados. Se tiver alguma dúvida, volte e refaça os exemplos apresentados neste capítulo. Se ainda assim você não conseguir resolver algum exercício, peça ajuda ao tutor.



Atividades e Orientações de Estudo

Dedique, pelo menos, 5 horas de estudo para o Capítulo 3. Você precisa praticar bastante esse conteúdo apresentado no último capítulo, pois requer bastante treino. Organize uma metodologia de estudo que inicie com a leitura dos conceitos e seja seguida pela resolução de exercícios.

Você poderá esclarecer suas dúvidas com o professor e os tutores utilizando os chats e os fóruns tira-dúvidas no ambiente virtual de seu curso.

Não esqueça de ler atentamente o guia de estudo da disciplina,

pois nele você encontrará a divisão de conteúdo semanal, ajudando-o a dividir e administrar o seu tempo de estudo semanal. Lembre-se que as atividades somativas propostas pelo professor no ambiente virtual são importantes para o aprendizado e para a composição da sua nota. Observe os prazos estabelecidos pelo seu professor para essas atividades virtuais.



Vamos Revisar?

Neste capítulo nós aprendemos um pouco sobre Lógica Digital. Descobrimos que a Lógica Digital está baseada na Álgebra de Boole, a qual possui três Funções Lógicas Essenciais, as funções *And*, *Or* e *Not* e que a partir destas foram definidas mais três Funções Lógicas Básicas, as funções *Nand*, *Nor*, e *Xor*. Descobrimos também que as portas lógicas são circuitos eletrônicos que implementam as funções lógicas, e que cada porta lógica possui um símbolo esquemático, uma equação booleana e uma Tabela Verdade associada.

Neste capítulo nós aprendemos ainda também uma metodologia de projeto que nos permite construir Circuitos Lógicos Combinacionais utilizando apenas as seis portas lógicas básicas. Segundo a metodologia apresentada nosso projeto se divide em três etapas básicas: construir uma Tabela Verdade que expresse a funcionalidade esperada do circuito, deduzir as equações booleanas parciais e a equação booleana geral do circuito e por fim transformar a equação booleana geral encontrada em um circuito lógico combinacional substituindo os operadores lógicos utilizados por portas lógicas equivalentes.

Considerações Finais

Olá, Cursista!

Esperamos que você tenha aproveitado este primeiro módulo da disciplina **Infraestrutura de Hardware**. No próximo módulo, continuaremos estudando o subsistema de processamento e daremos início ao estudo das estruturas de interconexão, popularmente conhecidos por barramentos.

Aguardamos sua participação no próximo módulo.

Até lá e bons estudos!

Juliana Regueira Basto Diniz

Abner Correa Barros

Professores Autores



Referências

STALLINGS, William. **Arquitetura e Organização de Computadores**. 5. ed.

PATTERSON, D. A. e Hennessy, John L. **Organização e Projeto de Computadores**. LTC, 2000.

TANENBAUM, Andrew S. **Organização Estruturada de Computadores**. 4. ed. Tradução Helio Sobrinho. Rio de Janeiro: Prentice-Hall, 2001.

Conheça os Autores

Juliana Regueira Basto Diniz

Possui graduação em engenharia eletrônica pela Universidade Federal de Pernambuco, mestrado e doutorado em Ciência da Computação pela Universidade Federal de Pernambuco. Atualmente é professora da Universidade Federal Rural de Pernambuco (UFRPE), desenvolvendo trabalhos no grupo de Educação a Distância desta universidade. Seus temas de interesse em pesquisa são: Sistemas Distribuídos, Computação Ubíqua e Ensino a Distância.

Abner Corrêa Barros

É mestre em Ciência da Computação com foco em Engenharia de Hardware pelo Centro de Informática da Universidade Federal de Pernambuco. Possui graduação em Ciência da Computação pela mesma universidade. Atualmente é professor da disciplina de Organização e Arquitetura de Computadores da Faculdade Maurício de Nassau e Engenheiro de Hardware da Fundação de Apoio ao Desenvolvimento da UFPE (FADE), atuando em um projeto de convênio entre o Centro de Informática da UFPE e a Petrobrás. Suas áreas de interesse e pesquisa são: Hardware Reconfigurável, Arquitetura de Cores Aritméticos e Computação de Alto Desempenho em Field-Programmable Gate Array (FPGA).