Licenciatura

Computação

Introdução à Organização de Computadores

Luciano José Senger

UNIVERSIDADE ABERTA DO BRASIL

UAB

EDUCAÇÃO A DISTÂNCIA

EaD em rede

ABRUEM
Associação Brasileira dos Reitores das Universidades Estaduais e Municipais

EDUCAÇÃO A DISTÂNCIA

LICENCIATURA EM

Computação

Introdução à Organização de Computadores

Luciano José Senger

PONTA GROSSA / PARANÁ
2017

CRÉDITOS

Universidade Estadual de Ponta Grossa

Carlos Luciano Sant'ana Vargas
Reitor

Gisele Alves de Sá Quimelli
Vice - Reitor

Pró-Reitoria de Assuntos Administrativos
Amaury dos Martyres - Pró-Reitor

Pró-Reitoria de Graduação
Miguel Archanjo de Freitas Junior - Pró-Reitor

Núcleo de Tecnologia e Educação Aberta e a Distância
Eliane de Fátima Rauski- Coordenadora Geral
Marli de Fátima Rodrigues - Coordenadora Pedagógica

Sistema Universidade Aberta do Brasil
Eliane de Fátima Rauski - Coordenadora Geral
Marli de Fátima Rodrigues - Coordenadora Adjunta
Marcelo Ferrasa - Coordenador de Curso

Colaboradores em EAD
Dênia Falcão de Bittencourt
Cláudia Cristina Muller

Projeto Gráfico
Eloise Guenther

Colaboradores de Publicação
Denise Galdino - Revisão
Eloise Guenther - Diagramação

Ficha catalográfica elaborada pelo Setor Tratamento da Informação BICEN/UEPG

Senger, Luciano José
S476i Introdução à organização de computadores/ Luciano José Senger.
Ponta Grossa : UEPG/ NUTEAD, 2017.
139p. ; il.

Curso de Licenciatura em Computação. Universidade Estadual de Ponta Grossa.

1. Sistemas numéricos. 2. Representação de dados. 3. Lógica digital. 4. Componentes lógicos. 5. Circuitos. 6. Memória. I. T.

CDD : 004.14

UNIVERSIDADE ESTADUAL DE PONTA GROSSA
Núdeo de Tecnologia e Educação Aberta e a Distância - NUTEAD
Av. Gal. Carlos Cavalcanti, 4748 - CEP 84030-900 - Ponta Grossa - PR
Tel.: (42) 3220-3163
www.nutead.org
2017

APRESENTAÇÃO INSTITUCIONAL

A Universidade Estadual de Ponta Grossa é uma instituição de ensino superior estadual, democrática, pública e gratuita, que tem por missão responder aos desafios contemporâneos, articulando o global com o local, a qualidade científica e tecnológica com a qualidade social e cumprindo, assim, o seu compromisso com a produção e difusão do conhecimento, com a educação dos cidadãos e com o progresso da coletividade.

No contexto do ensino superior brasileiro, a UEPG se destaca tanto nas atividades de ensino, como na pesquisa e na extensão. Seus cursos de graduação presenciais primam pela qualidade, como comprovam os resultados do ENADE, exame nacional que avalia o desempenho dos acadêmicos e a situa entre as melhores instituições do país.

A trajetória de sucesso, iniciada há mais de 40 anos, permitiu que a UEPG se aventurasse também na educação a distância, modalidade implantada na instituição no ano de 2000 e que, crescendo rapidamente, vem conquistando uma posição de destaque no cenário nacional.

Atualmente, a UEPG é parceira do MEC/CAPES/FNDE na execução dos programas de Pró-Licenciatura e do Sistema Universidade Aberta do Brasil e atua em 40 polos de apoio presencial, ofertando, diversos cursos de graduação, extensão e pós-graduação a distância nos estados do Paraná, Santa Cantarina e São Paulo.

Desse modo, a UEPG se coloca numa posição de vanguarda, assumindo uma proposta educacional democratizante e qualitativamente diferenciada e se afirmando definitivamente no domínio e disseminação das tecnologias da informação e da comunicação.

Os nossos cursos e programas a distância apresentam a mesma carga horária e o mesmo currículo dos cursos presenciais, mas se utilizam de metodologias, mídias e materiais próprios da EaD que, além de serem mais flexíveis e facilitarem o aprendizado, permitem constante interação entre alunos, tutores, professores e coordenação.

Esperamos que você aproveite todos os recursos que oferecemos para promover a sua aprendizagem e que tenha muito sucesso no curso que está realizando.

A Coordenação

SUMÁRIO

PALAVRAS DO PROFESSOR

Caro (a) aluno (a)

Você está iniciando uma nova disciplina em seu curso de Licenciatura em Computação, chamada Introdução à Organização de Computadores. Esta disciplina é muito importante para o perfil profissional do egresso em Licenciatura em Computação e é um pré-requisito importante para as demais disciplinas da área de organização e arquitetura de computadores. Aqui, você terá a oportunidade de conhecer como um computador é estruturado e como é a organização de seus componentes básicos. Com essa disciplina, você conhecerá detalhes de sistemas de numeração binários e como o computador representa os dados. Você terá também noções de lógica digital e de álgebra Booleana, portas lógicas, circuitos combinacionais, circuitos sequenciais e de tecnologias de memória. Tais conceitos que são empregados para o projeto e construção dos componentes do computador.

Durante o curso, além deste livro, você terá acesso ao Espaço Virtual de Aprendizagem – EVA, no qual outras atividades e exercícios estarão à sua espera. Sempre que você tiver dúvidas, entre em contato com seu professor tutor, que irá ajudá-lo no que for preciso. Este livro é sua principal fonte de informação sobre a disciplina, é a que está mais à mão, e a de mais fácil consulta.

Este livro está organizado por meio de unidades. Procure entender bem os conceitos, os exemplos disponibilizados e fazer os exercícios propostos ao final de cada unidade. Estude unidade por unidade, passando para a próxima quando estiver seguro com os conceitos apresentados.

Luciano José Senger

OBJETIVOS E EMENTA

OBJETIVOS

Objetivo geral:

O objetivo da disciplina de Introdução à Organização de Computador é possibilitar que você adquira conhecimentos sobre a estrutura de um computador, pelos estudos de seus componentes básicos, dos sistemas numéricos, da representação de dados e dos sistemas digitais.

Objetivos específicos:

O aluno do Curso de Licenciatura em Computação deverá ser capaz de:

- Conhecer a história da computação e como os computadores são organizados, com enfoque nas características estruturais de computadores que seguem o modelo de programa armazenado de Von Neumann;
- identificar como o computador armazena e trata a informação em um sistema binário, assim como as representações intermediárias em hexadecimal e em octal, bem como o processo de conversão para o sistema de decimal;
- conhecer os métodos de codificação atualmente adotados para representação de dados e suas características;
- entender a importância da lógica digital para o projeto de sistemas digitais, por meio de expressões e tabelas verdade, bem como entender como é a simplificação de circuitos lógicos, por meio de postulados da lógica e de mapas de Karnaugh;
- projetar e especificar circuitos lógicos digitais básicos por meio de portas lógicas;
- conhecer os detalhes dos circuitos lógicos responsáveis pelas funções principais dos computadores, como decodificadores, multiplexadores e somadores;
- conhecer como os componentes de memória são construídos e tecnologias de memória empregadas em sistemas de computação.

EMENTA

Introdução e dados históricos. Sistemas numéricos e conversão de bases. Representação de dados. Funções e portas lógicas. Representações de circuitos lógicos: expressão booleana, diagrama lógico e linguagem de descrição. Simplificação de circuitos lógicos e equivalência entre circuitos. Circuitos combinacionais e sequenciais. Organizações de memórias com circuitos sequenciais. Tecnologias de memória.

UNIDADE I

Introdução e dados históricos

Objetivos de aprendizagem

- Entender a importância do estudo da organização de computadores.
- Entender os conceitos sobre os conceitos e sistemas de computação.
- Conhecer a estrutura básica de um computador.
- Conhecer as gerações dos computadores e os marcos do desenvolvimento nas tecnologias empregadas para a construção de computadores.

Roteiro de estudos

- SEÇÃO 1 – Motivação para o estudo da organização de computadores
- SEÇÃO 2 – Computadores e sistemas de computação
- SEÇÃO 3 – Estrutura do computador
- SEÇÃO 4 – História da Computação

UAB UNIVERSIDADE ABERTA DO BRASIL

PARA INÍCIO DE CONVERSA

Prezada (o) aluna (o)

Nesta unidade, você terá a oportunidade conhecer as motivações para o estudo da organização de computadores, a estrutura básica de um computador e a história dos computadores.

Bom estudo!

SEÇÃO 1
MOTIVAÇÃO PARA O ESTUDO SOBRE A ORGANIZAÇÃO DE COMPUTADORES

Embora a história dos computadores e da computação remonte ao ano 3.000 (AC), com a invenção e utilização do ábaco, foi a partir da década de 1940 que houve grandes avanços na computação, com a definição e implementação dos primeiros computadores eletrônicos de programa armazenado. Essa década definiu a terceira geração da computação e as bases para a construção dos processadores contemporâneos. Mais tarde e a partir da década de 70, avanços significativos ocorreram com a criação do primeiro circuito integrado e com a disseminação dos microprocessadores. Desde então, a indústria dos microprocessadores tem crescido sob a regência da Lei de Moore, a qual pressupõe que o desempenho dos microprocessadores dobra a cada 18 meses, com reflexo direto da integração e densidade de componentes por circuito integrado.

Esse crescimento permitiu que a integração de componentes, que era em torno de milhares de transistores em 1971 (com o Intel 4004, um

processador de 4 bits) passasse para a bilhões de componentes em tempos atuais com os processadores contemporâneos de 64 bits. As melhorias em termos de organização interna dos microprocessadores com a inclusão de níveis de memória cache, unidades de execução independentes, pipelines e técnicas de predição de execução permitiram, também, que a escala de integração de componentes fosse acompanhada por um acrescimento substancial de desempenho ao longo dos anos. Como reflexo, o crescimento da indústria de microprocessadores permitiu a redução de custos e, dessa forma a disseminação da computação, antes restrita a centros de computação isolados para toda a sociedade, na forma de computadores pessoais, telefones e equipamentos microprocessados com software embarcado, como modems, eletrônicos de uso geral e roupas.

O profissional da área de computação deve possuir conhecimento profundo em como o computador funciona, ou seja, como o mesmo é organizado, tendo em vista descobrir as potencialidades e limitações dos sistemas de computação. Com tal conhecimento, o profissional se torna apto para dimensionar sistemas de computação, desenvolver e implantar programas de computador eficientes, que façam bom uso dos recursos computacionais. Desempenho, confiabilidade, segurança do software e redução do consumo de energia são vitais em projetos de computação.

Desastres e prejuízos financeiros têm sido gerados pela falta do conhecimento dos detalhes do conjunto de instruções e da organização dos processadores. Dentre os erros de programação mais divulgados e relacionados com detalhes da arquitetura de computadores, destacam-se os casos de erro de software básico no controle dos mísseis Patriot e do foguete Ariane 5. No caso dos mísseis de defesa Patriot, como descreve o relatório estadunidense *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran*, o software básico apresentou um erro básico, mas grave: devido à falha no sistema de defesa Patriot, mísseis iraquianos SCUD atingiram o território do Kuwait, durante a guerra do golfo e, no mínimo 28 pessoas foram mortas em 25 de fevereiro de 1991. Em outro caso, o foguete Ariane 5 explodiu apenas 40 segundos após deixar o solo. Seria a primeira viagem do foguete, após uma década de desenvolvimento e investimentos de 7 Bilhões de dólares. O Foguete e a sua carga valiam 500 milhões de dólares. Após investigação, o comitê responsável concluiu que a causa da falha foi um erro de software, no sistema de controle inercial.

Outro desastre relacionado com o mau funcionamento do software de sistema ocorreu no Canadá, em 1985. Três pessoas morreram e mais três pessoas sofreram ferimentos graves quando uma máquina de tratamento por radiação, chamada de Therac-25, apresentou defeito e entregou doses letais de radiação aos pacientes. O erro foi causado pelo software operacional da máquina. Também na década de 1980, em 1983 toda a humanidade poderia ser extinta por um erro no sistema de defensivo soviético (Rússia). Nesse ano, o sistema de alarme de mísseis falsamente indicou que os Estados Unidos haviam lançado cinco misseis ao território soviético. Felizmente, os russos interpretaram que se os Estados Unidos atacassem eles lançariam mais que cinco mísseis e que o sistema de controle deveria apresentar comportamento anômalo, fato que foi confirmado posteriormente.

Tais fatos (e desastres) demonstram a importância do estudo da organização e arquitetura de computadores para os estudantes da área de computação: alunos de Cursos de Ciência da Computação precisam saber os detalhes da arquitetura de instruções para criar compiladores; alunos de Cursos de Sistemas de Informação e Engenharia de Software precisam saber dimensionar sistemas de computação para implantar tecnologias de informação e alunos de Cursos de Engenharia de Computação devem conhecer os detalhes da arquitetura dos computadores para projetar software embarcado, otimizar o uso de compilador, desenvolver hardware de interface e projetar novos processadores. Tal conhecimento também é vital para os alunos dos Cursos de Licenciatura de Computação, que são responsáveis por disseminar os conceitos e as tecnologias da computação na sociedade com exatidão e responsabilidade.

SEÇÃO 2
COMPUTADORES E SISTEMAS DE COMPUTAÇÃO

Os computadores e os sistemas de computação têm um papel importante na sociedade, estando presentes no dia a dia das pessoas, na educação, nas comunicações, na saúde, na gestão, na indústria, na agricultura, nas artes e na pesquisa científica. Atualmente, a maioria dos dispositivos eletrônicos incorpora um sistema de computação para controle, desde smartphones até equipamentos domésticos como televisões e geladeiras. Isso extrapola a ideia inicial que os computadores estariam presentes na sociedade apenas como computadores *desktop, notebooks* e computadores de grande porte e servidores dos centros de computação. Avanços na computação e da tecnologia dos computadores, notados principalmente a partir do século 20, estabeleceram um evento único, comparável em importância ao desenvolvimento da escrita ou da imprensa. A sociedade contemporânea depende dos computadores e de sistemas de computação, bem como de seus profissionais da área, para dar segurança como exemplo nas transações comerciais, financeiras e nas operações de transporte.

Em sua essência, o computador é uma máquina, composta de partes eletrônicas e eletromecânicas, que pode ser instruído (programado) para processar um conjunto de operações aritméticas e lógicas.

A parte eletrônica e mecânica, chamada comumente de hardware, é capaz de processar uma sequência de operações simples, chamadas de **instruções de máquina**. Um **programa de computador** é um conjunto de instruções de máquina. Essa característica dos computadores, de poderem ser programados, os torna muito flexíveis e úteis para a sociedade.

Programas de computador são criados pelos programadores, que escrevem programas em uma linguagem de programação de alto nível. Com o uso de ferramentas de software, os programas em linguagens de programação são traduzidas em instruções de máquina. Tais ferramentas e programas, chamados comumente de **software**, podem ser unidos ao

hardware computacional para resolver problemas de várias áreas da sociedade.

Como o computador funciona internamente? Quais são as partes de um computador? Quantas unidades operacionais e conexões existem no computador? Como o computador representa e processa as informações? Tais perguntas são respondidas pelo estudo da **organização dos computadores.** A organização interna de um computador estabelece como as instruções são representadas e processadas, como a informação é representada, quantas unidades funcionais existem internamente, bem como os componentes internos são projetados e construídos.

SEÇÃO 3
ESTRUTURA DE UM COMPUTADOR

A Figura 1.1 ilustra uma representação simples de um computador (Stallings, 2010). O computador interage com o ambiente externo, com as pessoas e outros dispositivos, por meio de seus periféricos. Exemplos de periféricos são o teclado e mouse. Através das linhas de comunicação, como, por exemplo, uma rede de comunicação, o computador troca informação com dispositivos e outros computadores.

Internamente, o computador contém a sua **unidade central de processamento** ou **processador.** A unidade central de processamento, abreviada como UCP ou CPU (*Central Processing Unit*) realiza o processamento das instruções e dos dados dos programas. Interligado diretamente com a UCP, o computador tem uma **memória principal**, cuja função é o armazenamento dos programas e dados. Todo programa que é executado pelo computador é armazenado na memória. Essa forma de organização, chamada desde a década de 1940 como **arquitetura de programa armazenado** permite que o computador tenha flexibilidade no processamento, desde que distintos programas podem ser armazenados nessa memória e processados a critério do usuário e/ou programador.

Além desses componentes, a estrutura básica de um computador contém dispositivos de entrada e saída (E/S), que englobam os dispositivos periféricos e dispositivos de armazenamento permanente da informação, com discos magnéticos e unidades de armazenamento de estado sólido. Todos os componentes desta estrutura trocam informação por meio de linhas de comunicação ou barramentos de entrada e saída.

A Figura 1.2 ilustra os componentes principais da unidade central de processamento. A unidade de controle é responsável por controlar os componentes dedicados ao processamento das instruções. Tais componentes são implementados na unidade lógica e aritmética, chamada de ULA ou de ALU (*Aritmethic and logic unit*) e são responsáveis pelas funções de processamento do computador. Os registradores atuam como "bloco de rascunho" armazenando em uma memória de acesso rápido instruções e dados de programas. Todos os componentes internos à UCP são interligadas por um barramento que viabiliza a troca de informações. Cabe salientar que um computador contemporâneo pode ter mais de uma unidade central de processamento ou processador.

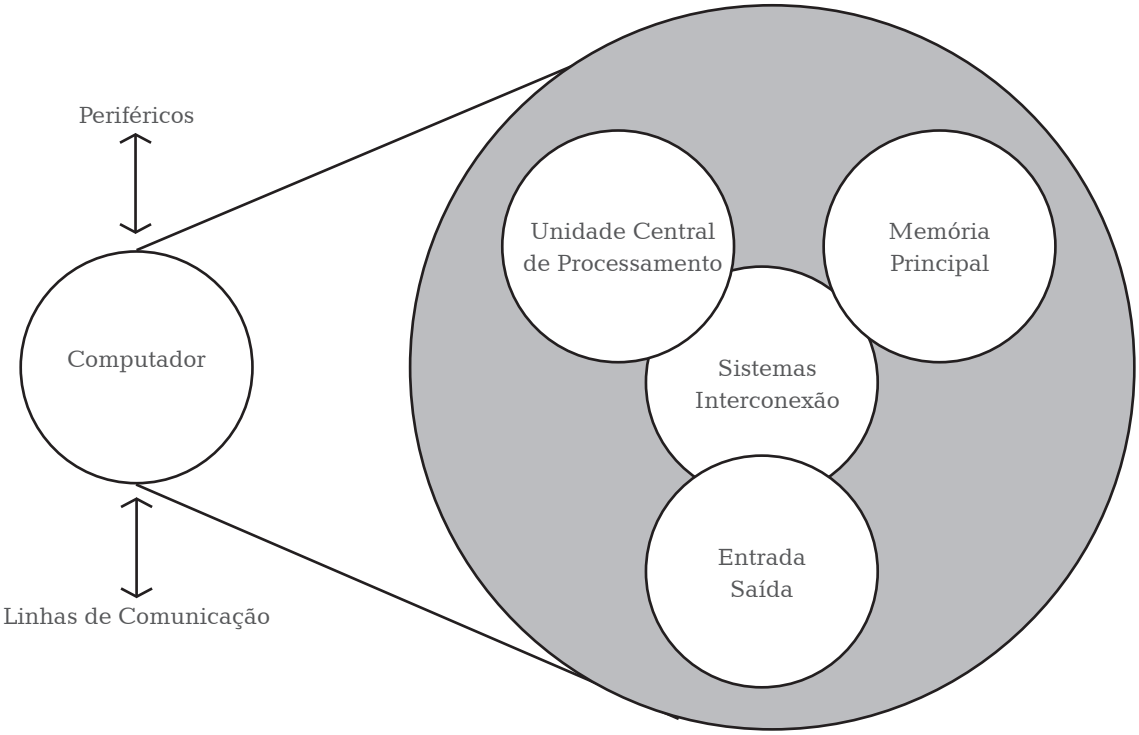


Figura 1.1. Estrutura de um computador (Fonte: Stallings, 2010)

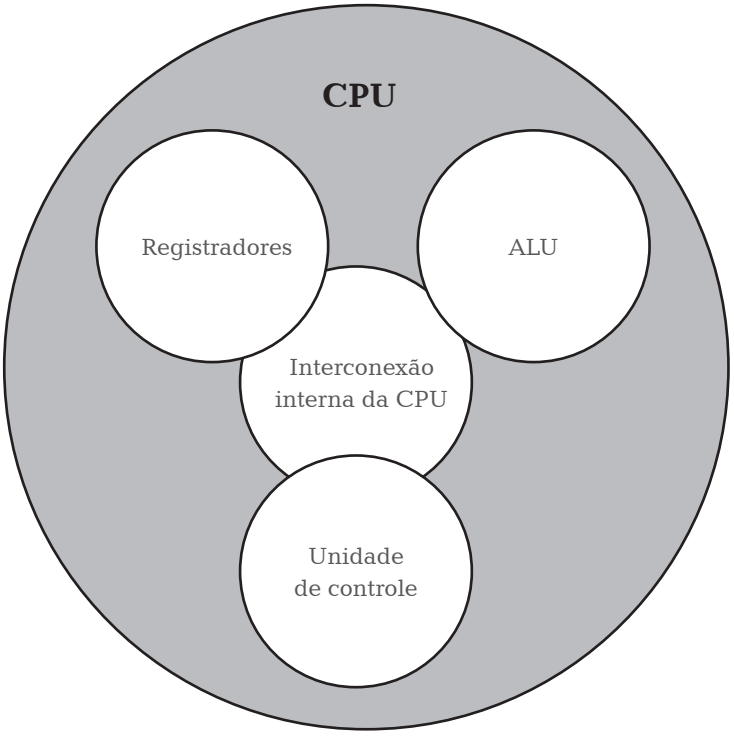


Figura 1.2. Estrutura da Unidade Central de Processamento (Adaptado de Stallings, 2010).

SEÇÃO 4
HISTÓRIA DOS COMPUTADORES

É comum, quando se trata da história do computador, organizar o estudo por meio de gerações. Considera-se que, além da chamada geração zero ou geração dos dispositivos mecânicos, há quatro gerações na história dos computadores. A tecnologia adotada em cada geração corresponde ao marco que a identifica.

Geração zero ou geração dos dispositivos mecânicos

A invenção e o desenvolvimento dos computadores vêm da necessidade da sociedade humana de realizar cálculos de forma eficiente, ou seja, de forma rápida e com precisão. Apesar do grande

desenvolvimento na indústria de computadores e da ciência da computação ter sido mais pronunciada no século 20, desde os tempos remotos, os povos babilônicos já empregavam o ábaco para tais tarefas. O ábaco é considerado o precursor do computador atual (Figura 1.3) e considerado o primeiro dispositivo mecânico para cálculos. A primeira evolução após o ábaco foi a *Pascalina* ou calculador de Pascal (Figura 1.4), dispositivo mecânico criado pelo matemático Blaise Pascal em 1642, que empregava engrenagens e rodas para realizar cálculos de soma e subtração. O ábaco e a calculadora de pascal não podem ser automatizados (ou programados) realizando sempre operações fixas, como soma e subtrações.

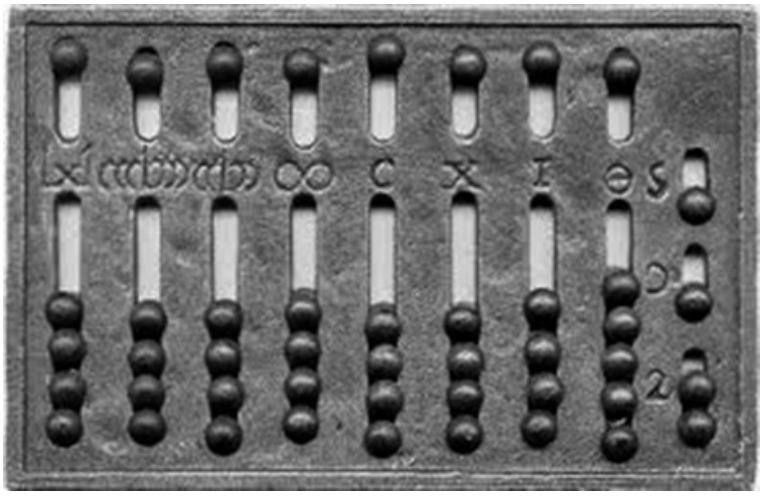


Figura 1.3. Ábaco mesopotâmico

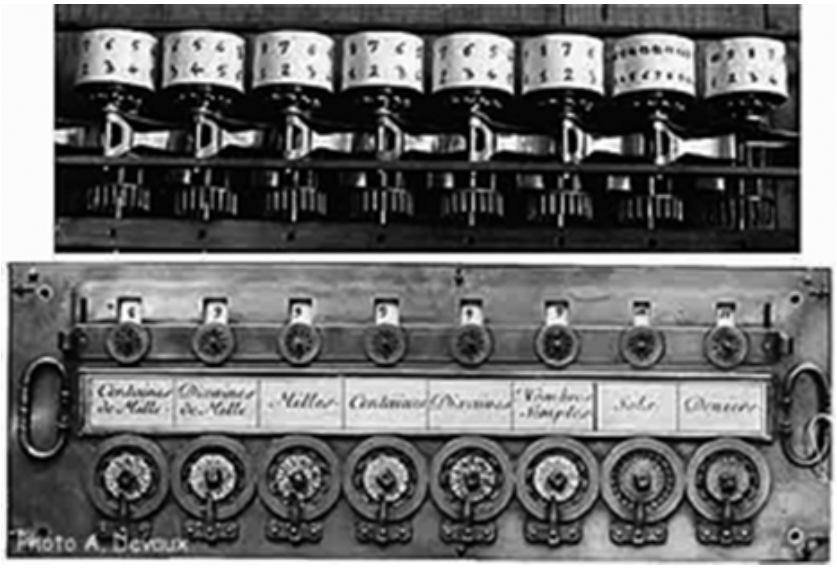


Figura 1.4. Calculadora de Blaise Pascal (Foto de A. Deavaux).

O primeiro dispositivo mecânico que podia ser programado surgiu em meados de 1801, com o desenvolvimento de um equipamento para tecelagem por Joseph Jacquard, uma máquina que podia ser programada por meio de cartões perfurados para produzir tapetes. Como curiosidade, para tecer o retrato de seu criador a máquina precisou de um programa composto de vinte e quatro mil cartões perfurados.

Ainda na era dos dispositivos mecânicos, trabalhos importantes foram realizados por Charles Babbage, que em 1823, sob encomenda da Marinha Real Inglesa, desenvolveu um dispositivo mecânico para auxiliar o cálculo repetitivo de tabelas de navegação. O dispositivo criado por Babbage foi chamado de máquina de diferenças (Figura 1.5). Mais tarde, Babbage se dedicou a criar um dispositivo que pudesse ser programado. Para isso, projetou a chamada máquina analítica, que era um computador mecânico, o qual podia ser programado por meio de cartões e tinha uma memória composta por engrenagens mecânicas. Anos mais tarde, Herman Hollerith, em 1880, construiu uma máquina para realizar as operações de recenseamento dos Estados Unidos da América.

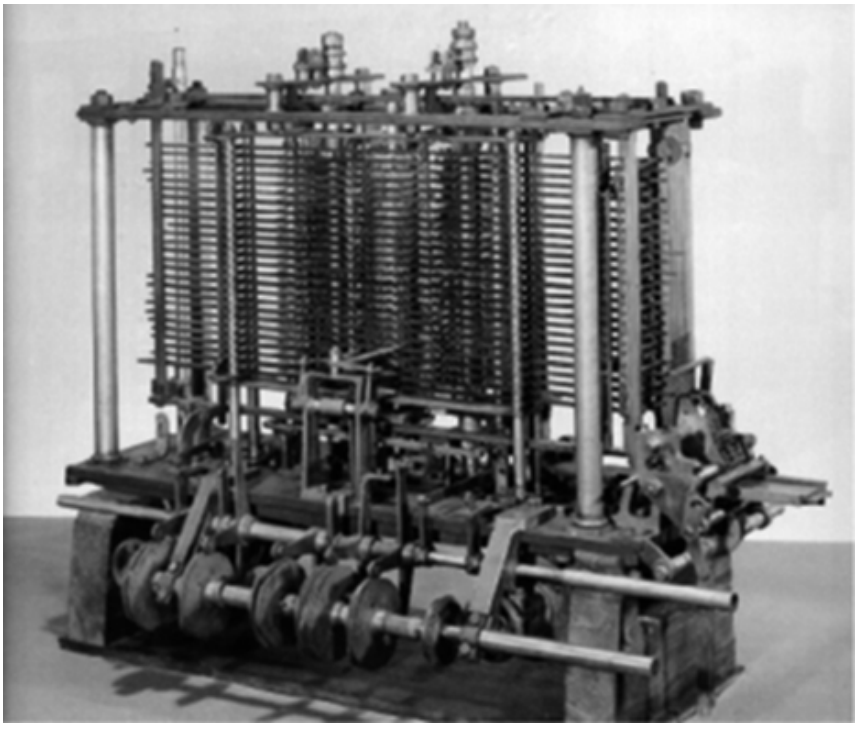


Figura 1.5. Máquina analítica de Babbage (Fonte: Wikipédia Commons).

A primeira contribuição com o uso de dispositivos eletromecânicos foi o trabalho de Konrad Zuse, que em 1930, na Alemanha, construiu uma série de calculadoras automáticas usando relés eletromagnéticos. Teve seu trabalho destruído durante a segunda guerra mundial, de forma que seu trabalho não teve influência em trabalhos posteriores (Tanenbaum & Austin, 2013).

Nos Estados Unidos, na década de 1940, os pesquisadores John Atanasoff e George Stibbitz projetaram calculadoras, sendo a máquina de Atanasoff avançada para época por empregar aritmética binária e capacitores. Capacitores são dispositivos eletrônicos e foram usados por Atanasoff para construir elementos de memória. Circuitos integrados de memória principal dos computadores atuais empregam o mesmo componente, capacitores para armazenar a informação, que torna o trabalho de Atanasoff precursor sobre a construção da unidade de memória dos computadores.

Por fim, cabe salientar o trabalho de Howard Aiken, que inspirado no trabalho de Babbage, decidiu construir um computador empregando relés. Seu computador foi chamado de Mark I e foi finalizado em 1944. O computador usava fitas de papel perfuradas para entrada dos programas e dados.

Os pesquisadores da geração zero eram visionários que, embora tenham inspirado demais pesquisadores para a construção de computadores e para o avanço da computação, foram derrotados pela tecnologia da época, composta primariamente de dispositivos mecânicos e eletromagnéticos. Avanços significativos na computação iniciaram com o desenvolvimento da válvula, dando origem a primeira geração dos computadores.

A primeira geração dos computadores

A geração zero foi marcada pelo incentivo da segunda guerra mundial e dos militares para a construção de computadores. Logo no início do conflito, os alemães estavam tendo grandes avanços com seus submarinos, causando baixas em embarcações britânicas. Os submarinos contavam com um sistema de comunicação pelos quais os almirantes alemães enviavam informações via rádio. As informações eram codificadas, sendo que, mesmo que recuperadas (ouvidas) pelos ingleses, não podiam ser interpretadas. O dispositivo empregado pelos alemães para a codificação era chamado de máquina ENIGMA (Figura 1.6).

Figura 1.6. Máquina de codificação de mensagens alemã ENIGMA.

Após a inteligência britânica ter obtido uma máquina ENIGMA, pesquisadores começaram a tentar a decifrar as mensagens dos alemães. Para isso, era necessária uma quantidade muito grande de cálculos, que deveriam ser processados em um tempo hábil, de forma que a mensagem ainda fosse útil (ou seja, pudesse interpretar uma ordem de comando alemã com a localização dos ataques). Para tentar decodificar tais mensagens o governo britânico financiou um projeto para construir um computador eletrônico, chamado de COLOSSUS. O computador é classificado como eletrônico por usar válvulas eletrônicas para processar a informação. O matemático Alan Turing, pioneiro da ciência da computação, trabalhou nesse projeto. O computador funcionou desde 1943, mas o projeto ficou escondido como segredo militar por trinta anos. O COLOSSUS é considerado o primeiro computador digital do mundo (Tanenbaum & Austin, 2013). A Figura 1.7 ilustra uma parte do computador COLOSSUS, na qual é possível visualizar as válvulas eletrônicas do computador.

Por motivos parecidos dos militares britânicos, os militares estadunidenses necessitavam realizar cálculos de balística com maior velocidade e precisão. Em 1943, John Mauchley e sua equipe começaram a construir um computador eletrônico, chamado de ENIAC (*Electronic Numerical Integrator and Computer*). O ENIAC era composto por 18 mil válvulas e 1500 relés, pesava 30 toneladas e consumia 140 kW de energia (Como comparação, um computador atual consome em média 500 W para funcionar). O projeto foi finalizado apenas em 1946, após o término da segunda guerra mundial.

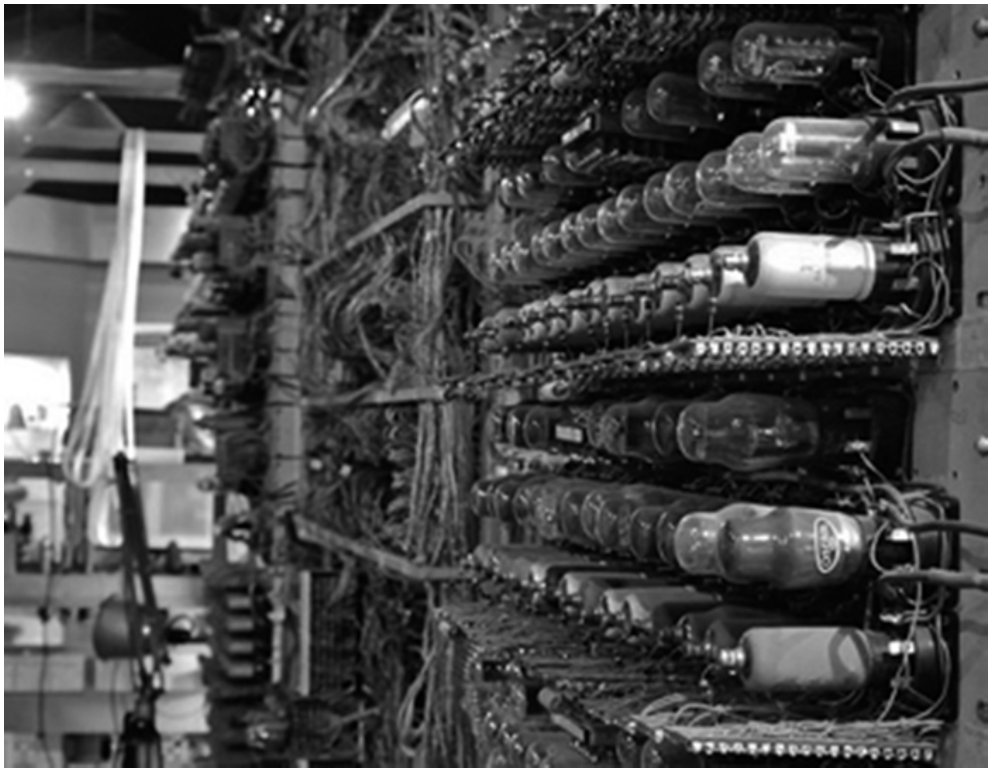


Figura 1.7. Computador COLOSSUS (Fonte: Wikipédia).

Contemporâneo de Mauchley e também participante do projeto do ENIAC, o pesquisador John Von Neumann, considerado o mais eminente matemático da época e gênio por muitos pesquisadores, trabalhou no desenvolvimento do computador IAS. Von Neumann percebeu que os projetos da época tinham deficiências importantes. Uma delas era o fato do uso de cabos e interruptores para realizar a programação do computador. Outra era a forma com que o ENIAC processava a

informação, empregando o sistema decimal, como os humanos. Von Neumann substituiu a aritmética decimal pela aritmética binária, mais adequada para ser usada com os dispositivos eletrônicos.

O seu projeto, conhecido como **máquina de Von Neumann**, é considerado como a base dos computadores digitais atuais (Figura 1.8). O computador de Von Neumann também é chamado de computador de programa armazenado ou arquitetura de programa armazenado, pelo fato que os programas são armazenados na memória principal para serem processados, em vez de existirem nas conexões de cabos e interruptores. Isso trouxe grande flexibilidade para os computadores.

Outra contribuição de Von Neumann foi a separação do conceito de **arquitetura** lógica do computador da sua implementação física. A arquitetura de um computador define a quantidade de instruções, tamanho de bits da palavra que será processada e outros aspectos de mais alto nível em relação à implementação física do hardware. O projeto de Von Neumann foi feito em termos de blocos lógicos e suas interconexões. A forma que Von Neumann idealizou como um computador deveria ser, independente de *como* ele fosse construído, deu origem ao termo **arquitetura de Von Neumann.**

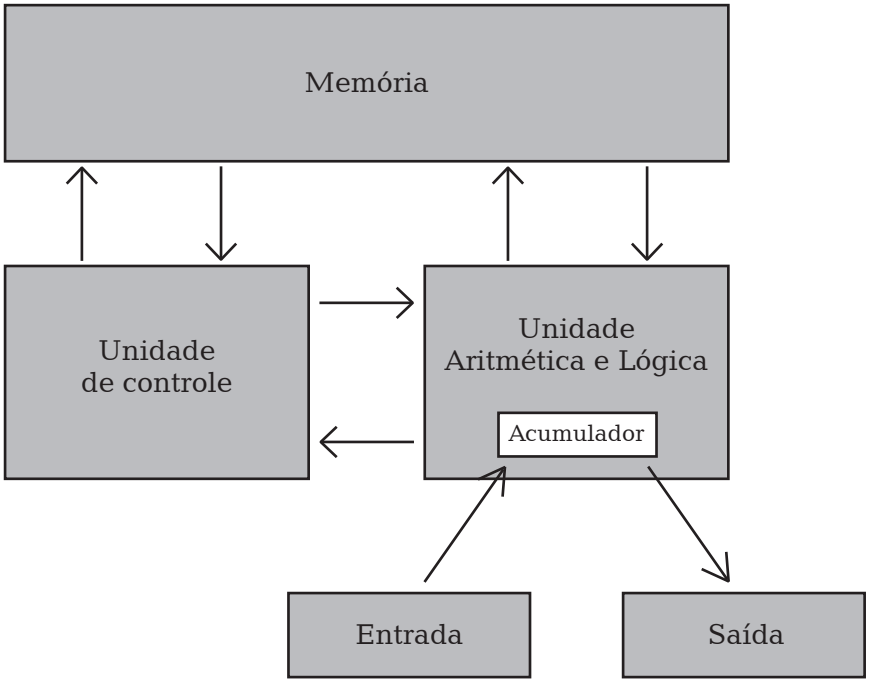


Figura 1.8. Máquina de programa armazenado de Von Neumann (Fonte: Wikipédia).

A segunda geração dos computadores

A segunda geração dos computadores é marcada pela invenção do transistor. O transistor é um dispositivo que realiza basicamente a mesma função da válvula eletrônica, mas, quando comparado com a válvula eletrônica tem dimensões reduzidas, gasta menos energia em seu funcionamento, ocupa uma área menor e é muito mais confiável.

O transistor foi inventado em 1948 e deu origem ao primeiro computador transistorizado, chamado de TX-0. Os engenheiros que construíram o TX-0, dentre eles Kenneth Olsen, fundaram a companhia DEC (*Digital Equipament Corporation* em 1957) para fabricar computadores digitais. Em meio à incerteza de investidores, que não acreditavam que computadores eram importantes, o primeiro projeto demorou quatro anos para ficar pronto e foi chamado de computador PDP-1. O PDP-1 foi vendido a partir de 1961 pelo valor de 120 mil dólares. O PDP-1 é considerando como um marco na origem a indústria de computadores.

Mais tarde a companhia DEC lançou o PDP-8, precursor da ideia de *barramentos*. **Barramento** é um componente eletrônico que serve para interligar unidades do computador forma estruturada e padronizada. Cerca de cinquenta mil unidades do PDP-8 consolidando a relevância da indústria de computadores na sociedade.

A terceira geração dos computadores

A terceira geração dos computadores é marcada pela invenção por Jack Kilby e Robert Noyce, em 1958, do circuito integrado. Essa invenção permitiu que vários transistores fossem integrados (colocados e interligados) em um único circuito, chamado de pastilha ou *chip*. A integração nessa época, que é a quantidade de transistores que podiam ser integrados em uma pastilha, era de pequena escala (cerca de poucas dezenas transistores) e de média escala (poucas centenas de transistores). A invenção do circuito integrado foi importante, pois permitiu reduzir dimensão e consumo dos computadores trazendo ganhos no desempenho e na confiabilidade. Os primeiros computadores a usarem circuitos integrados foram os B3500 e o B3600, fabricados pela Burroughs Corporation e os primeiros computadores da companhia IBM em 1964. A

IBM produziu nessa época os computadores da linha IBM System/360, compatíveis entre si em nível de software.

Com a evolução dos circuitos integrados, em 1965 surgiu a **Lei de Moore**, criada por Gordon Moore, considerado um visionário da área de eletrônica, publicou uma teoria evolucionista da indústria de computadores que estabelecia que o número de transistores dos chips dobraria a cada 18 meses, mantendo o mesmo custo. Desde a sua concepção, a Lei de Moore ditou o crescimento da indústria ao longo dos próximos anos até os dias atuais.

Ainda nos anos 1960, período marcado pela chamada *guerra fria,* os computadores foram empregados extensivamente pelos governos estadunidense e soviético para projetos militares. Na área de pesquisa científica, o homem chegou a Lua com o projeto Apollo, que tinha um computador (sistema de navegação) chamado de AGC (*Apollo guidance computer*), composto de 4100 circuitos integrados e uma velocidade de processamento de 1024 Mhz (como comparação, um computador atual tem um processador completo em apenas um circuito integrado e trabalha em uma frequência na ordem de 3Ghz). Os circuitos integrados do computador (Figura 1.9) empregavam portas lógicas do tipo NOR (ver Unidade 4) de 3 entradas (Figura 1.10). O AGC foi inventado pelo laboratório de instrumentação do MIT, nos Estados Unidos e fabricado pela Raytheon a partir de 1966. O AGC é considerado o primeiro sistema computacional embarcado.

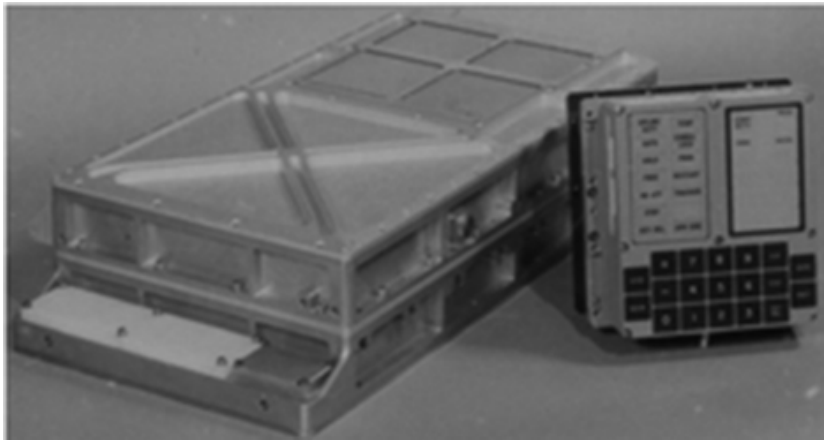


Figura 1.9. Computador AGC do projeto Apollo

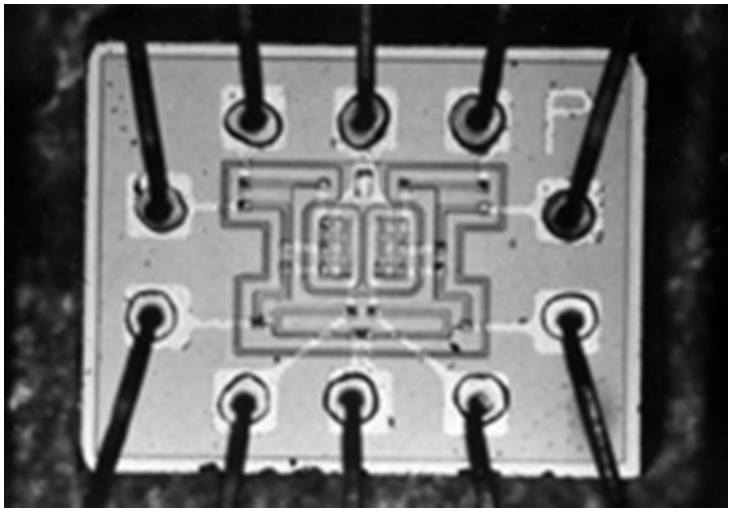


Figura 1.10. Detalhe do circuito integrado com portas NOR de três entradas.

A quarta geração dos computadores

A quarta geração iniciou na década de 1980, com a utilização em larga escala dos microprocessadores, bem como uma integração maior de componentes nas pastilhas. Essa integração, chamada de VLSI (*Very large scale integration*), foi possível pelo desenvolvimento da indústria de circuitos integrados que começou a colocar inicialmente centenas e, ao longo dos anos, tem integrado milhões de transistores em um único circuito integrado. Tal avanço foi notado a partir da criação do microprocessador, em 1971, pela companhia Intel. Nesse ano, a Intel produziu o microprocessador Intel 4004, que foi vendido e utilizado em calculadoras (Figura 1.11).

A partir da integração VLSI, os computadores tornaram-se mais rápidos e mais baratos e foi possível o surgimento do conceito do computador pessoal. Essa época é marcada pelo lançamento do computador pessoal pela IBM, com o processador Intel 8080 (Figura 1.12). O computador pessoal contribuiu para a disseminação do computador na sociedade: antes restrito às grandes empresas e indústrias, a partir do computador pessoal as pessoas poderiam ter o seu próprio computador. A popularização do computador permitiu que o software evoluísse, pelas mãos e cérebros de vários programadores pelo mundo.

Figura 1.11. O primeiro microprocessador Intel 4004.

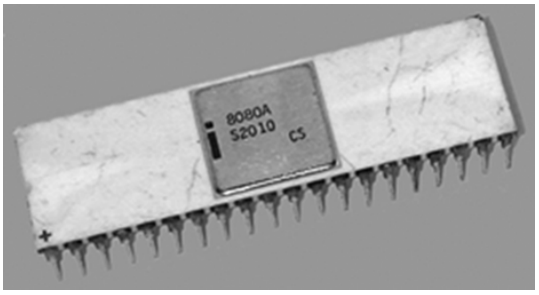


Figura 1.12. Microprocessador Intel 8080 (Fonte: Wikipédia).

Ainda em relação ao hardware, houve também o desenvolvimento da computação reconfigurável e do projeto de sistemas digitais por meio de ferramentas de software. Dentre as tecnologias para desenvolvimento de sistemas digitais, destacam-se as matrizes do tipo FPGA (*Field Programmed Gate Array*) e das linguagens de descrição de hardware, como a linguagem VHDL. Com tais ferramentas é possível o projeto e síntese de hardware de forma rápida e com a possibilidade de reconfiguração.

A quarta geração permanece até os dias atuais. Cabe salientar que devido ao desenvolvimento dos microprocessadores, muitas outras áreas foram beneficiadas. Por exemplo, o desenvolvimento dos circuitos integrados permitiu a criação e consolidação de tecnologias de redes de computadores e da Internet. Além disso, trouxe um grande desenvolvimento do software, principalmente dos sistemas operacionais e das ferramentas de desenvolvimento de maneira geral.

ARTICULANDO

Leve, para sala de aula, componentes básicos de um computador para que o aluno perceba na prática esses componentes. Obtenha componentes usados, como circuitos integrados, e demonstre para o aluno para que o mesmo tenha ideia de como são fisicamente os componentes de um computador.

SAIBA MAIS

- Visite o site Museu da História da Computação Computer History Museum (http://bit.do/iocuni1a).

- Visite o site História da Computação (http://bit.do/iocuni1b).

- Visite o site (http://bit.do/iocuni1c) e verifique a integração dos componentes eletrônicos ao longo do tempo.

- Assista ao filme **"O Jogo da Imitação (2014)",** que trata do ENIGMA e dos pesquisadores britânicos (entre eles Alan Turing) que pesquisavam como quebrar a criptografia empregada pelos nazistas na segunda guerra mundial (http://bit.do/iocuni1d).

Após ter estudado o conteúdo desta unidade, é hora de você realizar alguns exercícios para fixação.

1. Como o computador é organizado?

2. Estabeleça uma linha do tempo em relação às gerações dos computadores, ressaltando as tecnologias que foram o marco em cada uma das gerações.

3. Qual o grande avanço na computação que foi estabelecido por John Von Neumann?

4. Visite http://www.wow.com/wiki/Transistor_count . Qual a integração, em número de transistores do processador 8088? Como classificar essa integração?

5. Visite http://www.wow.com/wiki/Transistor_count . Qual a integração, em número de transistores do processador Core i7 Ivy Bridge? Como classificar essa integração?

ANOTAÇÕES

UNIDADE II

Sistemas Numéricos

Objetivos de aprendizagem

- Entender o conceito de notação posicional em sistemas numéricos.
- Conhecer o sistema binário de numeração.
- Conhecer como é feita a conversão entre as bases decimal e binária.
- Entender a aritmética básica no sistema de numeração binário.

Roteiro de estudos

- SEÇÃO 1 – Sistemas de numeração
- SEÇÃO 2 – Conversão de bases de numeração
- SEÇÃO 3 – Aritmética binária

UAB UNIVERSIDADE ABERTA DO BRASIL

PARA INÍCIO DE CONVERSA

Prezada (o) aluna (o)

Nesta unidade, você terá a oportunidade conhecer o sistema de numeração binário, empregado pelo computador para processar digitalmente a informação e como é a conversão entre a base decimal e a binária. Serão demonstrados também como é realizada a aritmética básica em sistemas binários e conceitos importantes.

Bom estudo!

SEÇÃO 1
SISTEMAS DE NUMERAÇÃO

A necessidade de representar quantidades e de realizar operações aritméticas, como soma e subtração, é algo que acompanha o ser humano desde que o homem habitava as cavernas. Muitas cavernas registram nas rochas símbolos, na forma de riscos colocados um ao lado do outro, que se acredita representar contagem de animais ou de eventos. Além das pinturas em cavernas, acredita-se que o homem pré-histórico empregava ossos, cordas e pedras para representar quantidades (Figura 2.1).

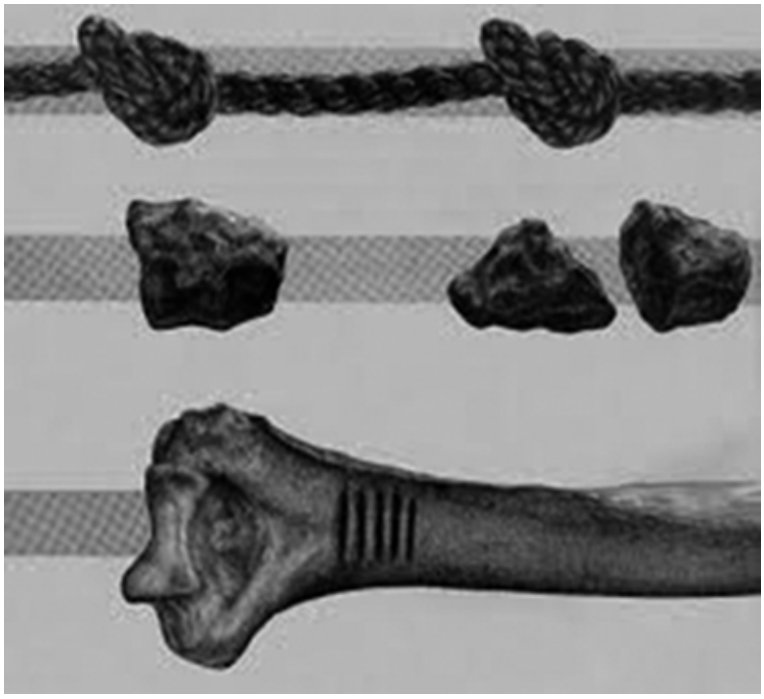


Figura 2.1: Representação de quantidades pelo homem pré-histórico.

Há diferentes formas de representar números. Tradicionalmente aprendemos, desde crianças, a contar nos dedos das duas mãos de um até dez. Assim, no nosso dia a dia empregamos dez símbolos diferentes (de 0 a 9) para a contagem de valores, representação de datas e para operações aritméticas.

A contagem de valores empregando os símbolos de 0 a 9 caracteriza o sistema decimal, ou sistema de numeração decimal. Esse sistema tem origem na Índia, 600 a.C. Os hindus, que viviam no vale do Rio Indo, onde hoje é o Paquistão, conseguiram desenvolver um sistema de numeração que reunia as diferentes características dos antigos sistemas. O conhecimento sobre esse sistema de numeração foi passado aos persas, transcrito em arábico e introduzido na Europa pelas invasões bárbaras, recebendo assim o nome de algarismos arábicos, que são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. A Figura 2.2 ilustra a evolução dos símbolos até a definição dos algarismos empregados atualmente.

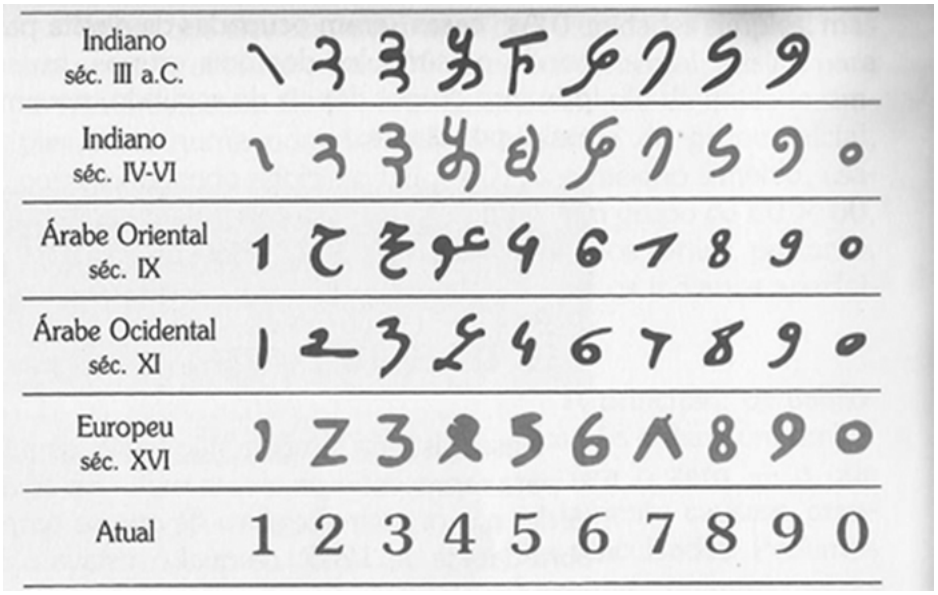


Figura 2.2: Evolução dos algarismos arábicos.

O homem contemporâneo está familiarizado com a base 10 (decimal), no dia a dia, já os computadores atuais trabalham exclusivamente com a base 2 (binário). O processamento de dados pelos circuitos eletrônicos dos computadores é realizado empregando o sistema numérico binário, que tem apenas dois símbolos: o 0 e o 1. Esses símbolos são representados internamente pelo computador por dois níveis de voltagem (por exemplo 0 = 0 volt e 1 = 5 volts). Na realidade, tais níveis não são valores únicos, mas sim faixas de representação. Isso permite que o sistema se torne imune a interferências, diferenças de características entre componentes e variações de tensão.

Os Sistemas de Numeração permitem representar dados numéricos através de números, caracteres ou símbolos, dependendo da forma de escrita utilizada. Os sistemas mais comuns de numeração são chamados de posicionais, onde o valor absoluto é composto pela posição dos algarismos em um número. Em um sistema posicional a **base** representa a quantidade de algarismos (símbolos) permitidos. Estamos familiarizados com algumas bases, por exemplo: ao medir o tempo, têm-se os segundos (60 s) que totalizam 1 minuto, minutos (60m) que totalizam uma hora. Nesse exemplo, estamos trabalhando com base 60, ou seja, a cada 60 unidades de tempo passamos para uma unidade maior.

No nosso dia a dia utilizamos a base 10, base decimal, tanto para a maioria dos cálculos, quanto para uma representação numérica qualquer. No entanto, o computador utiliza apenas dois estados possíveis para representar internamente as informações, ou seja, ele utiliza o **bit** (*binary digit*) que constitui a base binária, a qual é representada pelos elementos (0, 1). Outras bases são a Octal, representada pelos elementos (0, 1, 2, 3, 4, 5, 6, 7) e a Hexadecimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), que são representações intermediárias empregadas para visualização rápida e menos sujeita a erros dos valores binários. Estas duas bases são bastante utilizadas devido à velocidade de conversão dos valores em binários, uma vez que elas são múltiplas entre si.

A Tabela 2.1 ilustra exemplos de números, nos sistemas de numeração binário, octal, decimal e hexadecimal. No sistema de numeração de base 2, apenas são usados dois símbolos: 0 e 1. No sistema de numeração de base 8 são empregados apenas os algarismos de 0 a 7. Observe que, a partir do 7 não há mais coincidência com a representação decimal. Na base 16, hexadecimal, ocorre o contrário. A base é maior que a quantidade de algarismos do sistema decimal e dessa forma emprega-se letras de A a F para representar os valores maiores que 9.

Tabela 2.1 Números nas bases binária, octal, decimal e hexadecimal.

Base binária	Base Octal	Base decimal	Base hexadecimal
0	0	*0*	*0*
1	1	*1*	*1*
10	2	*2*	*2*
11	3	*3*	*3*
100	4	*4*	*4*
101	*5*	*5*	*5*
110	6	*6*	*6*
111	7	*7*	*7*
1000	10	*8*	*8*
1001	11	*9*	*9*
1010	12	*10*	*A*
1011	13	*11*	*B*
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10

Se os computadores empregam números binários para realizar operações aritméticas e outras de processamento, para que servem as bases octal e hexadecimal? Servem apenas para "encurtar" a representação dos números e facilitar a entrada e saída de informações. Assim, um número binário como 10101111 é simplesmente representado como AF em hexadecimal. Mais simples e menos sujeito a erros de digitação e de interpretação.

SEÇÃO 2
CONVERSÃO ENTRE BASES

Como realizar a conversão entre diferentes bases de numeração, por exemplo, da base binária para a decimal? A **notação posicional** e o teorema fundamental da numeração permitem que os números possam ser convertidos de uma base para outra, usando um método simples que emprega a base e o valor relativo de cada algarismo do número para obter um valor absoluto. Por exemplo, considere o seguinte número, na base decimal (a notação indica que *a* é um número expresso na base decimal). O número é composto de três algarismos. Cada número tem o seu valor absoluto, iguais respectivamente a 1, 2 e 5, e seus valores relativos iguais a 1×100, 2×10 e 5×1. A soma desses valores relativos à posição do algarismo produz o número em questão: $1\times100+2\times10+5\times1=125$.

Pode-se representar esse número utilizando a notação posicional da seguinte forma: substituindo os valores 100, 10 e 1 por potências relativas à posição do algarismo do número, de forma que o 5 está na posição 0, o 2 na posição 1 e o 1 na posição 2.

Para converter um número em uma base diferente de 10, para a base decimal, emprega-se a fórmula da notação posicional. Por exemplo, considere a conversão de um número da base 8 para a base 10: . Assim, a conversão no número pode ser realizada da seguinte forma:

O método inverso, ou seja, para converter um número da base 10 para uma determinada base *b* é feito por um conjunto de divisões sucessivas do número pela base até que o resultado da divisão seja 0. Esse método é chamado de método das divisões sucessivas. Com esse método, os algarismos na base *b* vão sendo obtidos na ordem inversa, do menos significativo para o mais significativo. Por exemplo, considere a seguinte conversão no número para a base binária, expressa da seguinte forma:

$$\frac{49}{2} = 24, r_0 = 1;$$
$$\frac{24}{2} = 12, r_1 = 0;$$
$$\frac{12}{2} = 6, r_2 = 0;$$
$$\frac{6}{2} = 3, r_3 = 0;$$
$$\frac{3}{2} = 1, r_4 = 1;$$
$$\frac{1}{2} = 0, r_5 = 1;$$

Figura 2.3. Exemplo de conversão de base por meio de divisões sucessivas.

Onde r_i corresponde ao resto da divisão obtida na fase *i*. Assim, o número 49_{10} na base 2 (binária) é igual a 110001_2. Mais um exemplo, em outra forma de ilustrar a conversão, do método das divisões sucessivas, representando os quocientes na primeira coluna, nesse caso a conversão 13_{10}=0_2:

Quociente	*Resto*
$(13)_{10}$	–
$(6)_{10}$	1
$(3)_{10}$	0
$(1)_{10}$	1
$(0)_{10}$	1

$$13_{10} \rightarrow 1101_2$$

Figura 2.4. Exemplo de conversão de base por meio de divisões sucessivas.

Conversões entre as bases 2, 8 e 16

As conversões nas bases 2, 8 e 16 são mais simples por serem potências entre si: base binária é 2^0, base 8 é 2^3 e a base 16 é 2^4. Como exemplo, vamos realizar a conversão entre a base 2 e a base oito, $(23)_8=0_2$. O método para conversão é agrupar os algarismos em grupos de tamanho igual ao tamanho da base fonte, convertidos para a base alvo. A base fonte é a base 8, ou seja $2^3=8$, grupos de tamanho de 3 bits. A base alvo é a binária, então o algarismo 2 é equivalente a 010 e o algarismo 3 é igual a 011, de forma que o resultado final é igual a $(23)_8=(010011)_2$.

E o processo inverso, como fica? No processo inverso, da base 2 para a base 8, deve-se organizar o número em grupos de 3 bits, da direita para a esquerda: 010 011 e para cada grupo atribuir o número em base 8 equivalente. Mais um exemplo: o número 110101011011 equivale a qual número na base hexadecimal? Começamos agrupando da direita para a esquerda, em grupos de 4 bits (a base alvo é a hexadecimal): 1101 0101 1011. Para cada grupo, associa-se o algarismo hexadecimal equivalente, tendo como resultado D5B.

Mais alguns exemplos a seguir (Figura 2.5).

Base Binária	→	Base Hexadecimal
$(1001\ 1100)_2$	→	$9\,C_{16}$
$(0101\ 1010)_2$	→	$5\,A_{16}$
$(1010\ 1011\ 0111)_2$	→	$A\,B\,7_{16}$

Figura 2.5. Exemplo de conversão de bases potência de 2 por meio de agrupamento.

SEÇÃO 3
ARITMÉTICA EM BINÁRIO

Quando realizamos a seguinte operação de soma em decimal: 9 + 1, de forma automática sabemos que o resultado é 10. Isso deve ao fato que 9 é último algarismo que temos na base decimal; se somarmos mais 1, significa que teremos na posição 0 do número (lembre-se da notação posicional) o valor relativo igual a zero e haverá um início de contagem na posição 1 (ou potência 10^1). Em binário, tal raciocínio deve também ser empregado, lembrando que a base é 2 e que a quantidade de símbolos (apenas 0 e 1) é bem inferior à base decimal. Assim, seguem as operações em binário:

a. 0 + 0 = 0
b. 0 + 1 = 1
c. 1 + 0 = 1
d. 1 + 1 = 0, e *carry* 1 (vai-um) para o bit mais significante.

Um exemplo: **00011010 + 00001100 = 00100110**

```
          1  1 vai-um
    0  0  0  1  1  0  1  0  =   26(base 10)
+   0  0  0  0  1  1  0  0  =   12(base 10)
---------------------
    0  0  1  0  0  1  1  0  =   38(base 10)
```

Mais um exemplo: **00010011 + 00111110 = 1010001**

```
       1  1  1  1  1          vai-um
    0  0  0  1  0  0  1  1  = 19(base 10)
+   0  0  1  1  1  1  1  0  = 62(base 10)
_____________________________
 0  1  0  1  0  0  0  1  = 81(base 10)
```

Na subtração, quando em decimal fazemos a seguinte operação 11 – 2, fazemos mentalmente o empréstimo de um valor mais à esquerda, pois não é possível realizar a subtração 1 - 2. O "empréstimo" é , então emprestamos "10" para o "1" que fica com 1+10=11 e subtraímos 1 do 1 mais à esquerda.

Em binário, o mesmo método se aplica: caso não seja possível realizar a operação de subtração para o algarismo corrente, "emprestamos" do vizinho mais à esquerda. Segue um exemplo:

```
00100101 - 00010001 = 00010100
           2
  0  0  1  0  0  1  0  1  = 37(base 10)
- 0  0  0  1  0  0  0  1  = 17(base 10)
---------------------------------------
  1  0  0  1  0  1  0  0  = 20(base 10)
```

Na subtração, quando em decimal fazemos a seguinte operação 11 – 2, fazemos mentalmente o empréstimo de um valor mais à esquerda, pois não é possível realizar a subtração 1 - 2. O "empréstimo" é , então emprestamos "10" para o "1" que fica com 1+10=11 e subtraímos 1 do 1 mais à esquerda.

Em binário, o mesmo método se aplica: caso não seja possível realizar a operação de subtração para o algarismo corrente, "emprestamos" do vizinho mais à esquerda. Segue um exemplo:

```
00100101 - 00010001 = 00010100
           2
  0  0  1  0  0  1  0  1  = 37(base 10)
- 0  0  0  1  0  0  0  1  = 17(base 10)
---------------------------------------
  1  0  0  1  0  1  0  0  = 20(base 10)
```

Use o relógio com ponteiros, ábaco e outras ferramentas para demonstrar a ideia de sistemas de numeração e suas bases.

Visite o site sobre o sistema de numeração binário na Wikipédia: http://bit.do/iocuni2

Após ter estudado o conteúdo desta unidade, é hora de você realizar alguns exercícios para fixação.

1. Execute as seguintes conversões de base:

(a) $(10010010101)_2=0_8$; $(10010010101)_2=0_{16}$; $(10010010101)_2=0_{10}$

(b) $(11110101)_2=0_8$; $(11110101)_2=0_{16}$; $(11110101)_2=0_{10}$

(c) $(3AB08)_{16}=0_2$; $(3AB08)_{16}=0_{10}$; $(3AB08)_{16}=0_8$

(d) $(724)_8=0_2$; $(724)_8=0_{10}$; $(724)_8=0_{16}$

2. Crie um sistema de numeração de base 4, e relacione os 20 primeiros algarismos desse sistema de numeração.

3. Nesse novo sistema de numeração, realize as seguintes conversões:

(a) $(3120)_4=0_2$; $(3120)_4=0_8$; $(3120)_4=0_{10}$
(b) $(3120)_4=0_{16}$

4. Durante uma exploração, a arqueóloga Lar Acroft encontrou numa escavação uma pedra gravada com os seguintes caracteres:

% @ # %

&

– – – – – – –

% & # & %

Concluindo brilhantemente (e com uma boa dose de adivinhação) que os símbolos correspondiam a uma operação de adição entre dois números positivos e que todos os algarismos usados pela antiga civilização estão presentes na gravação, determine a base de numeração utilizada, o algarismo arábico correspondente a cada símbolo e a representação das parcelas e do resultado da adição, convertidas para a base 10.

5. Execute as seguintes conversões de base:

(a) $(2173)_{10}=0_{16}$
(b) $(2173)_{10}=0_8$
(c) $(2173)_{10}=0_2$
(d) $(2317)_8=0_2$
(e) $(1A45B)_{16}=0_8$
(f) $(3651)_{16}=0_2$
(g) $(11001011011011)_2=0_8$

6. Efetue as seguintes operações de soma:

(a) $(31752)_8+(6735)_8$
(b) $(2A5BEF)_{16}+(9C829)_{16}$
(c) $(37742)_8+(26573)_8$
(d) $(1100111101)_2+(101110110)_2$
(e) $(110011110)_2+(11011111)_2$

Efetue as seguintes operações de subtração:

(a) $(64B2E)_{16}-(27EBA)_{16}$
(b) $(2351)_8-(1763)_8$
(c) $(11001000010)_2-(11011111)_2$
(d) $(100010)_2-(11101)_2$

8. Quantos números inteiros positivos podem ser representados em uma base **b**, cada um com **n** algarismos?

9. A partir do valor binário 110011, escreva os cinco números que seguem esta sequência.

10. Quantos números binários diferentes podem ser armazenados em memórias com espaço de armazenamento de 6 dígitos cada uma?

11. Qual é o valor decimal equivalente ao maior número de 7 algarismos que pode existir na base 2?

12. Um odômetro hexadecimal mostra o número 5*ECFC*. Quais são as próximas seis leituras?

13. Um odômetro hexadecimal mostra o número *A*3*FF*. Qual é a leitura seguinte? Após rodar alguns quilômetros, o odômetro apresenta a seguinte leitura: *A*83*C*. Quantos quilômetros foram percorridos? (Dê a resposta em hexadecimal e decimal).


ANOTAÇÕES

UNIDADE III

Representação de dados

Objetivos de aprendizagem

- Entender o conceito de palavra binária.
- Conhecer as representações de números positivos e negativos em binário.
- Conhecer a forma de representação de números fracionários pelo computador.
- Conhecer a forma de representação de alfanuméricos por meio de tabelas de códigos.

Roteiro de estudos

- SEÇÃO 1 – Representação de dados numéricos
- SEÇÃO 2 – Representação de alfanuméricos

UAB UNIVERSIDADE ABERTA DO BRASIL

PARA INÍCIO DE CONVERSA

Prezada(o) aluna(o)

Nesta unidade, você terá a oportunidade conhecer como os números em binário são empregados pelo computador para representar valores numéricos e alfanuméricos. É muito importante você entender o conceito de palavra binária e como são as representações mais comuns para dados numéricos inteiros e fracionários (ponto flutuante).

Bom estudo!

SEÇÃO 1
REPRESENTAÇÃO DE DADOS NUMÉRICOS

Um computador executa operações sobre dados numéricos (os números) ou alfabéticos (letras e símbolos). Na unidade anterior, você aprendeu que o computador trabalha com dados no sistema de numeração binário. Além dos conceitos das bases binárias, um conceito importante é o conceito de palavra. Computacionalmente, a palavra significa o lote de bits que é processado de cada vez pelo computador. Por exemplo, um computador que tenha a palavra definida com o tamanho igual a 4 bits vai processar os dados em lotes de 4 bits. Atualmente é comum o uso de computadores de palavras de 32 e 64 bits. Quando os primeiros computadores comerciais foram lançados, era comum o uso de palavras com quantidade de bits menores, como 8, 12 ou 16 bits.

Então para que os dados sejam processados pelo computador, se faz necessário definir formas pelos quais os números e caracteres serão representados. A forma de representação define a quantidade de bits, que geralmente tem relação com a palavra e o método para

codificar a informação. Tanto o método para codificar a informação quanto à quantidade de bits empregada são escolhidos tendo em vista o desempenho (para que as operações sejam executadas mais rápidas pelo computador) e o consumo de memória (de forma que a memória seja utilizada racionalmente, sem desperdício no armazenamento das instruções e dados).

A representação dos dados define como os diferentes tipos de dados serão armazenados. É comum classificar os dados como numéricos ou alfanuméricos. Os dados numéricos podem ser representados em ponto fixo (números inteiros) ou em ponto flutuante (números reais e fracionários). Os alfanuméricos são representados por meio de tabelas, sendo as mais comuns ASCII, EBCDIC e UNICODE. Mais detalhes serão vistos a seguir.

A base de numeração adotada pelo computador é a binária. Assim, a forma mais comum de representar números é exatamente o seu correspondente em binário. Mas, antes de continuarmos com a representação de dados numéricos, precisamos conversar sobre o conceito de **palavra binária**. Antes de representarmos qualquer valor em binário, temos que definir o tamanho da palavra. Então, considere a seguinte questão exemplo: Como o computador armazena e processa o número 2 internamente? Bem, antes de tudo, precisamos definir qual o tamanho da palavra. Tomando como exemplo um tamanho de palavra de 8 bits (chamado de Byte), temos o valor 2 representado internamento pelo computador pelo lote de bits igual a 00000010. Tal forma de representação é bastante lógica e fácil de ser interpretada pelo software. Atualmente para representar valores numéricos inteiros, os computadores usam lotes (palavras) de 32 bits. Inclusive, o tipo *integer* da linguagem C ocupa 32 bits (em computadores de 64 ou de 32 bits).

Uma complicação adicional é a representação de números negativos. Como representar o valor -2? Não há milagres aqui. Vamos ter que gastar 1 bit da palavra para lidar com o sinal do número. Assim, poderíamos ter o valor binário 00000010 representando o número 2 e o valor 10000010 representando o valor -2. Note que o bit mais à esquerda (chamado de mais significativo) está ligado, ou seja, é igual a 1, indicando que o número é negativo. Simples, não? Note que é mais adequado que o bit de sinal seja o bit mais significativo, de forma que demais bits da palavra podem ser

empregados para operações aritméticas. Dessa forma, uma representação em binário com *n* bits disponibiliza para a representação de um número *n-1* bits mais à direita e o bit mais significativo representa o sinal (positivo ou negativo). Essa representação tem o nome de **representação em sinal e magnitude**. A tabela 3.1 apresenta mais exemplos de números representados em sinal e magnitude.

Tabela 3.1. Exemplos de números representados em sinal e magnitude.

Valor decimal	Valor binário com palavra de 8 bits (7 + bit de sinal)
+9	00001001 (bit inicial 0 significa positivo)
-9	10001001 (bit inicial 1 significa negativo)
+127	01111111 (bit inicial 0 significa positivo)
-127	11111111 (bit inicial 1 significa negativo)

É importante agora discutir o conceito de **faixa de representação**. A faixa de representação estabelece quantos valores (números) podem ser representados. Para a representação em sinal e magnitude, considerando uma base **b** com **n** bits, tem-se que há b^n representações e pode representar b^n -1 valores. Uma característica da representação é que existem duas representações para o número zero. Assim, considerando a base binária há 2^n representações, representando os valores entre - (2^{n-1}-1) e + (2^{n-1}-1). O maior valor inteiro positivo será então + (2^{n-1}-1) e o menor valor inteiro negativo será - (2^{n-1}-1) (Murdocca, 2001).

A representação em sinal e magnitude apresenta duas desvantagens: representação dupla para o zero e lentidão nas para operações aritméticas do computador (Murdocca & Heuring, 2000). Devido a esse fato, a representação em **complemento de 2** (abreviada como C2) é frequentemente usada em sistemas de computação para representar valores em ponto fixo, por permitir apenas uma representação para o zero e ser muito adequada para realizar operações aritméticas.

A representação C2 é para números negativos em ponto fixo e obtida **invertendo todos os bits,** após **somando 1** ao resultado. Como exemplo, vamos calcular o complemento a 2 (C2) de um número binário 0010 com 4 dígitos. Após inverter todos os bits, tempos 1101. Após a inversão,

soma-se mais 1 ao resultado: 1101+1 = 1110. Então, 1110 é equivalente ao valor -2 representado em binário, usando a representação C2. Note que, assim como sinal e magnitude, o bit mais à esquerda é igual a 1, indicando que o número é negativo. Como há apenas uma representação para o número zero, a faixa de representação é assimétrica, ou seja, há um número negativo a mais que pode ser representado em C2. A Tabela 3.3 apresenta mais exemplos de números representados em C2. Note que todos os números negativos têm o bit mais à esquerda igual a 1.

Tabela 3.3. Números em Complemento a 2 (C2) considerando uma palavra de 4 bits.

Decimal (positivo)	Em binário	Decimal (negativo)	Em binário
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

A faixa de representação da notação C2 é igual a 2^n valores (entre - 2^{n-1} e + 2^{n-1}-1), sendo 2 a base. Assim, o maior valor inteiro positivo será então + (2^{n-1}-1) e o menor valor inteiro negativo será - (2^{n-1}). Por que a representação em C2 é empregada? O primeiro motivo já foi discutido: é a representação única para o número zero. Considerando uma palavra de 4 bits como exemplo, o código 1000 representa o número -8, ao invés do zero redundante em sinal e magnitude. A segunda razão pode ser demonstrada com o seguinte exemplo: suponha que o computador (cuja palavra é 4 bits) necessita realizar a seguinte operação de subtração x = 5 – 2. Sabemos que essa operação é equivalente a x = 5+(-2). O computador realiza então a seguinte tarefa, o valor 5 tem seu binário equivalente a 0101 e o valor -2, em C2, é igual a 1110. Realizando a soma (e desprezando o último "vai

um") temos 0101+1110 = 0011, que é o valor 3. Dessa forma, empregando a representação em C2, o computador pode realizar operações de soma e de subtração usando o mesmo circuito (hardware) somador, que gera economia no projeto dos circuitos.

Mais uma questão que pode surgir é a seguinte: em um computador (considerando ainda a palavra de 4 bits de tamanho) o seguinte número negativo 1001 representado em C2 equivale a qual valor em decimal? Bem, olhando para o número 1001 notamos que o mesmo é negativo, desde que o bit mais à esquerda é igual a 1. Então, podemos fazer a operação de inverter os bits e somar 1 ao resultado, tendo 1001 = 0110+1 = 0111, que equivale ao número 7 na base decimal. Isso demonstra outra característica interessante e prática da representação C2, a sequência de passos (inverter os bits e somar 1 ao resultado) é a mesma, seja para obter o número negativo equivalente (p.e., para o número 7 em binário 0111->1000+1 = 1001) ou para recuperar o valor absoluto (não negativo) de um número representado em C2.

Para números inteiros (não fracionários), o computador emprega a representação C2. E para números reais, também chamados de fracionários, como, por exemplo, o valor 3,141516, como é a representação? Certamente aqui, as mesmas premissas devem ser obedecidas: a representação deve agilizar o processamento do computador com economia de memória na representação. A representação empregada para os números reais pelo computador é chamada de representação em ponto flutuante.

Por exemplo, considere o número 3,141516 acima, Este número pode ser expresso como $3{,}141516 \times 10^{0}$ ou ainda 3141516×10^{-6}. Assim, qualquer número pode ser representado na forma **número x base** $^{\text{expoente}}$, na qual se pode variar dois fatores: a posição da vírgula (que delimita a parte fracionária) e a potência à qual elevamos a base. Como a origem da palavra é inglesa (*floating point*), os livros traduzem tal representação como ponto flutuante. No Brasil, o ideal seria chamar de vírgula flutuante, pois empregamos a vírgula para separar a parte inteira da fracionária.

Para representar os números reais, o computador emprega a representação normalizada, na qual o número é normalizado movimentando a vírgula para a direita ou para a esquerda de forma que o número seja menor que 1 e o mais próximo possível de 1. Isso é feito multiplicando o número por uma potência da base de forma a manter o valor inicial do número. Alguns exemplos:

a) $17{,}583_{10}$ --> normalizando ==> $0{,}17583 \times 10^{2}$

b) $0{,}0004028_{10}$ --> normalizando ==> $0{,}4028 \times 10^{-3}$

c) $0{,}00001001_{2}$ --> normalizando ==> $0{,}1001 \times 2^{-4}$

Assim, os números reais são representados da seguinte forma: $\pm$ a x b $^{\pm expoente}$, onde **a** é o número a ser representado e **b** é base do sistema, que no computador é igual a 2. A parte do número (**a**) é chamada de mantissa (**M)**, onde $\pm$ 0,M x b $^{\pm e}$. Note que quanto maior a quantidade de bits empregada para representar a mantissa, maior a precisão da faixa de representação; quanto maior a quantidade de bits dedicados para representar o expoente, maior a magnitude dos números que podem ser representados. Os computadores geralmente empregam o padrão IEEE 754 para representar números em ponto flutuante (Murdocca e Heuring, 2000).

A norma IEEE 754, publicada em 1985, procurou uniformizar a maneira como as diferentes máquinas representam e processam os números em ponto flutuante (Murdocca & Heuring, 2000). Essa norma define dois formatos básicos para os números em ponto flutuante: o formato simples, com 32 bits e o duplo com 64 bits. O primeiro bit é para o sinal: 0 representa número positivo e 1 representa número negativo. No formato simples, o expoente tem 8 bits e a mantissa tem 23 bits; no formato duplo, o expoente tem 11 bits e a mantissa 52 bits.

SEÇÃO 2
REPRESENTAÇÃO DE ALFANUMÉRICOS

O armazenamento de alfanuméricos, chamados também de caracteres (letras, números e outros símbolos) é feito através de um esquema de representação onde se convenciona que cada carácter tem um código (representado em binário) associado. Três códigos de representação de caracteres são comumente empregados: ASCII, EBCDIC e UNICODE. Os códigos ASCII (*American Standard Code for Information Interchange*) e EBCDIC (*Extended Binary Coded Decimal Interchange Code*) são empregados pela maioria dos computadores e equipamentos. Esses códigos estabelecem que os caracteres devem ser representados por uma palavra de 8 bits (que dá ao todo uma quantidade de caracteres que podem ser representados igual a 2^8=256). A Tabela 3.4 apresenta os caracteres A e Z representados em EBCDIC e ASCII. Em relação ao código ASCII, cabe salientar que há duas versões: o código ASCII e o código ASCII estendido, que é o considerado aqui, de 8 bits de tamanho.

Tabela 3.4. Exemplos de alfanuméricos representados em EBCDIC e ASCII.

Alfanumérico	EBCDIC	ASCII
A	11000001	10100001
Z	11101001	10111010
&	01010000	00100110

Com o avanço dos computadores, tornou-se necessário representar não apenas os caracteres latinos, e sim caracteres oriundos de outros países e idiomas. O UNICODE é um padrão que serve para esse propósito. Criado em 1993, a versão corrente deste código permite representar aproximadamente 99.000 caracteres, abrangendo assim quase todos os sistemas de escrita empregados no planeta (Pedroni, 2010). O UNICODE inicialmente tinha 16 bits de tamanho. Atualmente há 3 esquemas

principais de codificação: UTF-8, UTF-16 e UTF-32. Dentre eles, o UTF-8 tem sido adotado por usar palavras de tamanho variável de 1, 2, 3 ou 4 Bytes. Visando economizar espaço de armazenamento, o padrão UTF-8 emprega 8 bits quando codifica os primeiros símbolos da tabela ASCII (índices 0 a 127) e 16 bits para os símbolos de índice 128 a 2047 (Pedroni, 2010).

Após ler e entender todos os conceitos apresentados nesta unidade siga em frente e resolva os seguintes problemas.

1. Monte um esquema de representação através de códigos binários para os pontos cardeais (N,S, E e O).

2. Inclua no esquema de representação os pontos secundários (p.e. NE).

3. Qual é a quantidade de números ou códigos diferentes que podem ser criados utilizando:

- 4 bits
- 8 bits
- 16 bits
- 32 bits

6. Qual é o maior inteiro positivo que pode ser representando em uma palavra de 4 bits, em sinal e magnitude?

7. Qual é o maior valor em hexadecimal que pode ser representado com 4 algarismos em hexadecimal?

8. Considerando a representação em complemento a 2 (C2), qual é a faixa de representação considerando uma palavra de 16 bits?


ANOTAÇÕES

UNIDADE IV

Lógica Digital

Objetivos de aprendizagem

- Conhecer conceitos da álgebra booleana e da lógica digital tendo em vista o projeto de sistemas digitais.
- Conhecer como é realizada a simplificação de expressões por meio de propriedades da lógica tradicional e por meio de diagramas.
- Entender como os níveis lógicos são representados no hardware do computador.
- Entender como é realizada a representação de expressões lógicas por meio de portas lógicas e de linguagens de descrição de hardware.

Roteiro de estudos

- SEÇÃO 1 – Lógica Digital
- SEÇÃO 2 – Simplificação de expressões lógicas com diagramas de Veitch-Karnaugh
- SEÇÃO 3 – Portas Lógicas e Hardware digital
- SEÇÃO 4 – Linguagens de descrição hardware

UAB UNIVERSIDADE ABERTA DO BRASIL

PARA INÍCIO DE CONVERSA

Prezada(o) aluna(o)

O computador é um representante de sistema digital. Nesta unidade, você terá a oportunidade conhecer os fundamentos de projeto de sistemas digitais pelo estudo da álgebra booleana e da lógica digital. Serão demonstrados como os conceitos e propriedades da álgebra de Boole são aplicados para o projeto de circuitos lógicos e como os diagramas de Veitch- Karnaugh são usados para simplificar expressões e circuitos lógicos. Após, você conhecerá como é a representação de circuitos lógicos por meio de portas lógicas e uma introdução sobre a representação por meio de linguagem de descrição de hardware.

Bom estudo!

SEÇÃO 1
LÓGICA DIGITAL

A álgebra de Boole ou booleana (1850) serve como base para a lógica digital e para a construção de computadores. A álgebra de Boole tem esse nome em retribuição da comunidade científica ao matemático inglês George Boole (1815-1864), que desenvolveu uma análise matemática sobre Lógica. A álgebra de Boole é formalismo para o tratamento da lógica tradicional, no qual o valor das variáveis podem ser iguais ou a *true,* ou *false*, ou ainda no contexto de circuitos digitais, 1 ou 0. Em 1938, C. E. Shannon aplicou pela primeira vez a álgebra booleana para demonstrar que circuitos elétricos de chaveamento podem ser representados por uma álgebra Booleana com dois valores (Uyemura, 2002).

Por exemplo, considere as seguintes proposições lógicas, que podem ser verdadeiras ou falsas:

- Está chovendo;
- A previsão do tempo indica chuva.

E a seguinte proposição combinada:

- *Está chovendo* **OR** *a previsão do tempo indica chuva.*

Podemos estabelecer no exemplo, que:

- Eu levarei guarda-chuva = Está chovendo **OR** a previsão do tempo indica chuva.

Pode-se pensar que a proposição *guarda-chuva* como um resultado que deve ser calculado pela combinação dos resultados das proposições *chovendo* e *previsão do tempo* (Figura 4.1). Desde que as proposições só podem assumir os valores verdadeiro ou falso, todas as possibilidades de entradas e de saídas podem ser descritas em uma tabela verdade (Tabela 4.1).

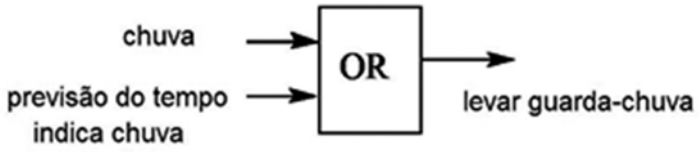


Figura 4.1. Proposições de lógica para o problema de levar o guarda-chuva.

Tabela 4.1. Tabela verdade do problema de decisão *levar guarda-chuva.*

chuva	previsão	guarda-chuva
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Na álgebra de Boole um formalismo para a lógica tradicional é empregado. Nesse formalismo, tem-se que: os valores são representados por 1 e 0, de forma que 1 = TRUE; 0 = FALSE; as proposições da lógica tradicional são substituídas por variáveis: por exemplo C = *está chovendo*

e P = *previsão do tempo indica chuva* e os operadores da lógica tradicional são substituídos por símbolos: ' = NOT, + = OR e • = AND. Assim, empregando ainda o exemplo do problema de levar o guarda-chuva, tem-se que G = P+C, o qual indica que devemos levar o guarda-chuva (G será verdade, igual a 1) se ou P ou C forem verdade (igual a 1). E se formos de carro? Vamos considerar que devemos levar o guarda-chuva (G) se não formos de carro. Considerando que a proposição *ir de carro* corresponde a C, a nossa expressão lógica revisitada (Figura 4.2) ficaria da seguinte forma: G = NOT(C) . (P+C).

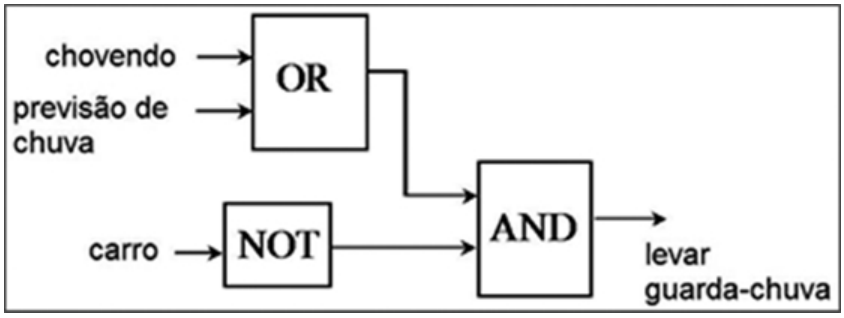


Figura 4.2. Proposições de lógica para o problema revisitado de *levar o guarda-chuva.*

Por que a álgebra booleana é importante no estudo sobre a organização de computadores? Um computador é um exemplo de sistema digital. Em sistemas digitais pode-se trabalhar os conceitos da álgebra booleana sobre bits, de forma a construir relações básicas que servem para criar componentes maiores (Uyemura, 2002). Assim, pode-se trabalhar sobre grupos de bits (palavras). Por exemplo: Sendo A=1, NOT(A)=0; Sendo A=0101, NOT(A)=1010. Assim, o processamento dos dados em nível de bits é possível por meio de células que executam operações matemáticas, resultando em uma saída com valor 0 ou 1 (Figura 4.3).

Figura 4.3. Circuito lógico digital simples com três entradas e uma saída.

Na álgebra booleana, a **Tabela Verdade** de uma função binária apresenta todas as combinações possíveis de entrada e as saídas respectivas. Assim, para **n** variáveis de entrada tem-se $\mathbf{2^n}$ possibilidades. A Figura 4.4 apresenta uma tabela verdade para uma função de três entradas e de uma saída.

A	*B*	*C*	*f*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Figura 4.4. Um exemplo de tabela verdade empregando valores binários.

As propriedades da álgebra boolena permitem, a partir de um problema especificado por meio de uma expressão lógica, obter uma expressão lógica mais simples (simplificação lógica ou simplificação de circuitos). As Figuras 4.5 e 4.6 apresentam as propriedades e teoremas empregados para a simplificação de expressões lógicas. Por exemplo, considere a seguinte função F = A.B + A.B′. Tal função pode ser simplificada passo a passo empregando as propriedades e teoremas da lógica:

F = A.B + A.B′ /* 1-propriedade distributiva */
F= A. (B + B′) /* 2- expressão entre parênteses =1 */
F= A. 1
F = A

No contexto de circuitos digitais, a simplificação é importante, pois leva à economia de componentes físicos eletrônicos nos projetos e

também a uma melhor eficiência do circuito (tempo de processamento e gasto de energia). Mais um exemplo:

G = ((A + B′ + C)+ (B + C′))′ /* Morgan */
G = (A + B′+ C)′ . (B + C′)′ /* Morgan */
G = A′. B′′. C′ . B′ . C′′e
G = A′. B . C′ . B′. C
G = A′. (B. B′). (C.C′)
G= 0

Notavelmente, a equação simplificada para G é muito mais simples que a expressão original. Um terceiro exemplo:

H = (A+B+C). (A+B) /*propriedade distributiva */
H = A.A + A.B + A.C + A.B + B.B + B.C
H = A + A.B + B + A.C + A.B + B.C /*A +A.B = A*/
H = A + B + C.A + B.C
H = (A + C.A) + (B + B.C)
H = A + B

AND	OR	Propriedade
A . B = B . A	A + B = B + A	Comutativa
A . (B + C) = A .B + A . C	A + (B . C) = (A +B) . (A + C)	Distributiva
1 . A = A	0 + A = A	Identidade
A . ~A = 0	A + ~A = 1	Complemento

Figura 4.5. Propriedades da álgebra booleana.

0 . A = 0	1 + A = 1	Teorema de 0 e 1
A . A = A	A + A = A	Idem Potência
A . (B . C) = (A .B) . C	A + (B + C) = (A + B) + C	Associativa
~~A = A		Involução
~(A . B) = ~A + ~B	~(A + B) = ~A . ~B	Teorema de Morgan
A . B + ~A . C + B . C = A . B + ~A . C	(A + B) . (~A + C) . (B + C) = (A + B) . (~A + C)	Teorema do Concenso
A . (A + B) = A	A + A . B = A	Teorema da Absorção

Figura 4.6. Propriedades da álgebra booleana.

O projeto de um circuito lógico usualmente inicia pelo conjunto de especificações das variáveis de entrada, utilizadas para produzir uma ou mais saídas. Comumente, adota-se a **formas canônica** ou lógica estruturada. A forma canônica é empregada para escrever equações booleanas de maneira que ela utilize vários tipos de formas regulares e repetidas (Uyemura, 2002). Formas canônicas são expressas por meio das representações soma de produtos (SDP) ou Produto de somas (PDS). Qualquer função lógica pode ser expressa na forma de uma SDP ou PDS.

Uma expressão lógica representada canonicamente por meio da soma de produtos (SDP) consiste em efetuar operações OR sobre termos contendo operações AND. A terminologia SDP vem do fato que operações AND, tais como A.B, são semelhantes a produtos (multiplicações) e que a operação OR (A+B) é similar a soma. Para a função ser considerada como na forma canônica SDP, todas as variáveis devem aparecer em cada um dos termos, em sua forma normal ou complementar.

Considere os seguintes exemplos:

Exemplo 1: suponha que tenhamos as variáveis A, B e C. As seguintes funções estão em suas formas canônicas de SDP:

F = A.B.C + A′.B.C + A.B′.C + A.B.C′

G = A′.B′.C′ + A′.B.C′ + A.B′.C′

A estrutura canônica é devida ao fato que todos os termos A, B e C estão contidos na equação.

Exemplo 2: considere a expressão booleana F(A,B,C) escrita como:

F = A.B + A′.C + B.C′

A equação está na forma soma de produtos, mas não em sua estrutura canônica, pois cada termo tem apenas duas de três possíveis variáveis.

Exemplo 3: Considere a expressão:

G(a,b,c) = a. b′.c + a′.b. c + a.c

Apesar dos dois primeiros termos satisfazerem o critério para a forma canônica, o último tem apenas as variáveis a e c, portanto G não está na forma canônica (embora esteja na forma SDP).

Uma equação pode ser formatada para a forma canônica aplicando propriedades e teoremas da lógica. Por exemplo, considere a função:

$H(x, y, z) = x.y + y.z$

Apesar de ser uma estrutura em SDP, ela não é classificada como sendo a forma canônica. Esta equação pode ser colocada em sua forma canônica usando as identidades: $(x+x') = 1$; $(y + y') = 1$ e aplicando a propriedade distributiva nos termos:

$H(x,y,z) = x.y.1 + 1.y.z$
$H(x,y,z) = x.y.(z+z')+ (x+x').y.z$
$H(x,y,z) = x.y.z + x.y.z' + x.y.z + x'.y.z$
$H(x,y,z) = x.y.z + x.y.z' + x'.y.z$

Esta técnica pode ser aplicada em qualquer expressão em SDP para formatá-la em sua forma canônica. Cabe salientar, que a formatação canônica insere complexidade (mais variáveis ou operações) na expressão lógica, por meio de um número maior de termos. Por outro lado, tem a vantagem de representar o problema com uma abordagem estruturada, auxiliando o projeto de sistemas digitais grandes e complexos (Uyemura, 2002).

A representação canônica em produto de somas (PDS) consiste em efetuar operações AND sobre termos contendo operações OR, como no exemplo: $F(x,y) = (x'+y) . (x+y')$. Comumente, a representação SDP é mais empregada para representação de circuitos lógicos, principalmente por ser mais simples de ser obtida tendo como entrada a tabela verdade do circuito lógico. Por exemplo, A Figura 4.7 demonstra como a expressão lógica de um circuito na forma canônica pode ser obtida a partir de uma tabela verdade. Cada linha que tem como saída o valor 1 dá origem a um termo na expressão lógica. Essa operação, de obter uma expressão lógica a partir de uma tabela verdade, é também chamada, na lógica digital, de derivação.

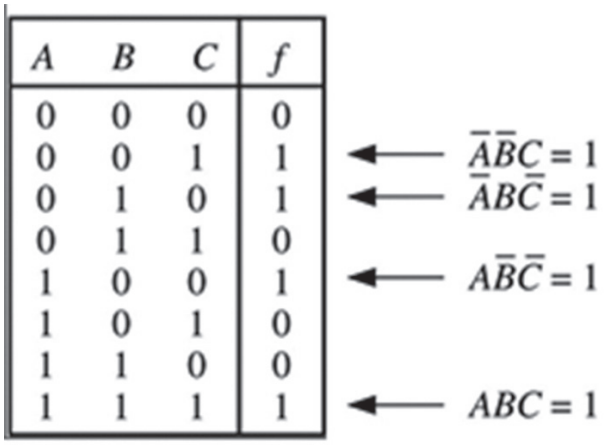


Figura 4.7. Extração da forma canônica a partir de uma tabela verdade (Fonte: Uyemura, 2002).

Outra vantagem da representação SDP é a facilidade na utilização da técnica de simplificação por meio de diagramas de Veitch-Karnaugh, ou simplesmente, mapas de Karnaugh.

SEÇÃO 2
SIMPLIFICAÇÃO DE EQUAÇÕES COM DIAGRAMAS DE VEITCH-KARNAUGH

Inventada em 1953 por Maurice Karnaugh (Harris & Harris, 2013), a técnica de mapas de Karnaugh representa tabelas verdade de uma forma que se pode rapidamente encontrar regiões na qual a identidade x+x´= 1 é reconhecida e empregada para simplificar a expressão lógica. Por exemplo, considere a seguinte sequência de simplificações:

G(a,b,c) = a.b.c + a.b.c′ + a.b′.c
G(a,b,c) = a.b(c + c′) + a.b′.c
G(a,b,c) = a.b + a.b′.c

Note que entre os dois primeiros termos (**a.b.c + a.b.c'**), a identidade x+x´= 1 pode ser empregada para reduzir para o termo **a.b.** Verifique que a *diferença* ou *distância* entre os dois termos em questão é de um bit apenas. Para isso, considere que a variável em sua forma normal equivale ao bit 1 e a variável em sua forma complementar equivale ao bit 0. Essa forma de interpretação, também chamada de **mintermos,** torna-se útil aqui. Assim, teríamos toda a expressão lógica completa poderia ser representada da seguinte forma:

G(a,b,c) = a.b.c + a.b.c′ + a.b′.c
G(a,b,c) = a.b′.c +a.b.c′ + a.b.c
G(a,b,c) = 101 + 110 + 111

Obtendo assim a expressão representada em mintermos como G(a,b,c)=∑(5,6,7), onde os valores em decimal 5, 6 e 7 correspondem aos termos 101, 110 e 111. Essa representação em mintermos evidencia que há termos que tem apenas um bit de distância (no caso dos mintermos 5 e 6, por exemplo), nos quais a identidade x+x´=1 pode ser empregada para realizar a simplificação. A Figura 4.8 apresenta mais um exemplo de derivação de mintermos a partir de uma tabela verdade de 3 entradas e duas saídas.

Entradas			Saídas	
X	Y	Z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$D = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ = m_1 + m_2 + m_4 + m_7 = \Sigma(1, 2, 4, 7)$$

$$B = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ + XYZ = m_1 + m_2 + m_3 + m_7 = \Sigma(1, 2, 3, 7)$$

Figura 4.8. Extração da forma canônica a partir de uma tabela verdade.

A utilização da técnica de mapas de Karnaugh emprega o conceito de mintermos e se dá da seguinte dos seguintes passos principais (Uyemura, 2002):

- Começando com a tabela verdade, são mapeadas as combinações de entradas e saída em uma tabela retangular.
- A estrutura da tabela e as regras associadas nos permitem facilmente localizar termos na qual a identidade (X+ X′)=1 pode ser utilizada para simplificar a função.

No passo 1, Mapas de Karnaugh utilizam a propriedade de que funções podem ser expressas em uma tabela verdade quadricular. Para obter a tabela verdade quadricular (mapa) deve-se começar com uma lista de todos os possíveis mintermos de entrada e os resultados da saída da função para cada mintermo. Se o resultado (saída) para o mintermo é igual a 1, preenche-se o quadro com o valor 1; se 0, preenche-se com o valor zero. Como exemplo, considere o caso de duas variáveis, no qual existem quatro mintermos: a′.b′ + a′.b + a.b′ + a.b. A Figura 4.9 ilustra essa situação.

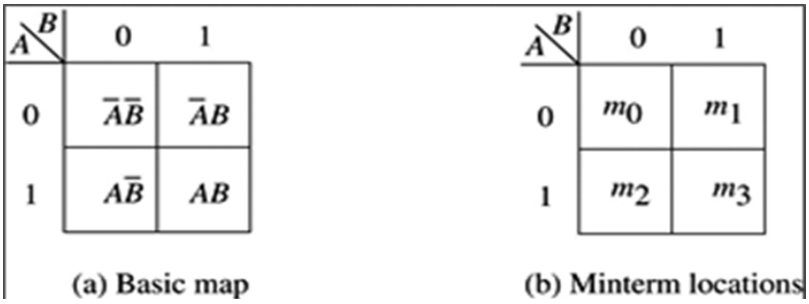


Figura 4.9. Mapa de Karnaugh para duas variáveis de entrada (Fonte: Uyemura, 2002).

As Figuras 4.10 e 4.11 apresentam mapas para duas variáveis, para as funções AND e NAND. Note como é o preenchimento dos quadros com os valores obtidos da tabela verdade. Para a função AND, apenas a posição do mintermo 3 é preenchida com 1. O quadro evidencia que não há mintermos próximos iguais a 1 que poderiam ser empregados para realizar a simplificação por meio da identidade x+x´=1.

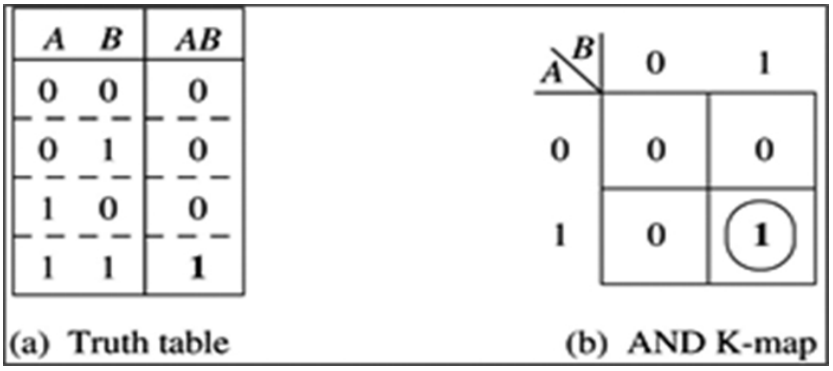


Figura 4.10. Mapa de Karnaugh para a função AND (Fonte: Uyemura, 2002).

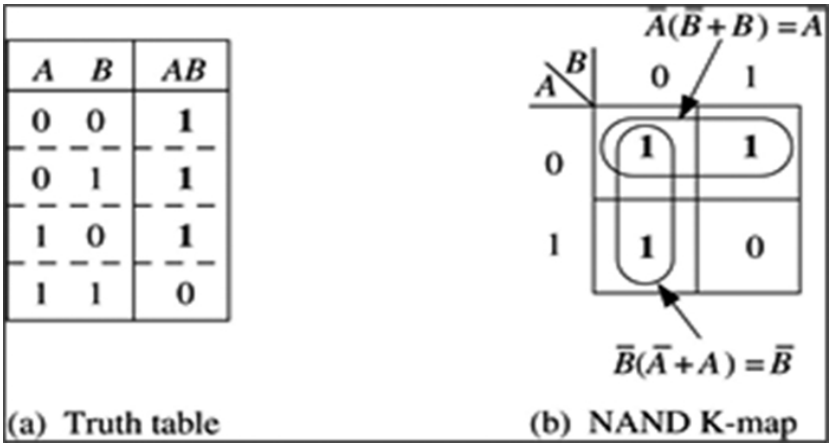


Figura 4.11. Mapa de Karnaugh para a função NAND (Fonte: Uyemura, 2002).

Já na Figura 4.11, o mapa evidencia que os mintermos 0, 1 e 2 estão próximos. Busca-se então circular esses termos formando duplas. Nesse caso, temos duas duplas. A técnica pode ser empregada aqui para simplificar da seguinte forma: para cada dupla formada, observa-se a variável que trocou de valor. Observe que no par da primeira coluna, temos que a variável A trocou de valor (alternou entre 0 e 1 no par). Essa variável é eliminada do termo, ficando apenas com a variável B´ (a variável na forma complementar é inserida na expressão final, pois o par em questão está na região de valor igual a 0). No par da linha, temos que a variável B alternou o seu valor nessa região, de forma que ficamos apenas com o termo A´. Dessa forma, a expressão final fica reduzida

para f (A, B) = A´+ B´. A Figura 4.12 mostra mais um exemplo de simplificação para duas variáveis, considerando as funções OR e NOR.

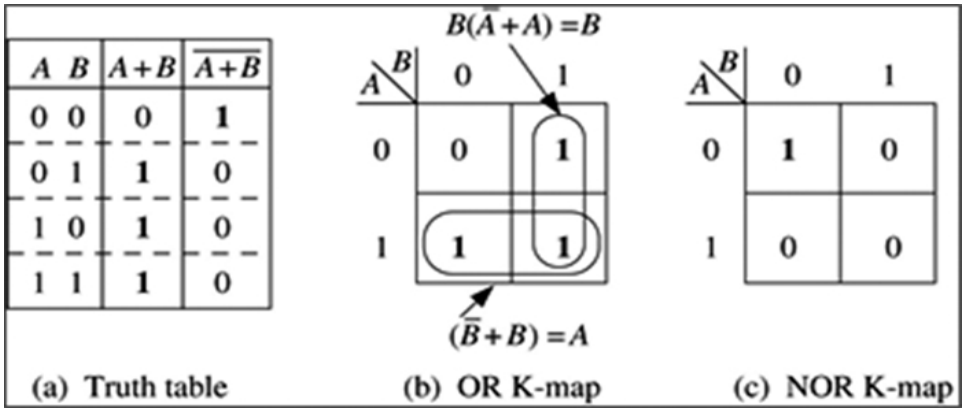


Figura 4.12. Mapa de Karnaugh para as funções OR e NOR (Fonte: Uyemura, 2002).

A Figura 4.13 mostra uma representação (não é a única) de um mapa para funções de três variáveis de entrada e, a Figura 4.14 apresenta um exemplo completo de simplificação para uma função de exemplo. Note que um mintermo pode ser empregado mais de uma vez para a simplificação. No caso de três variáveis, deve-se tentar localizar inicialmente *quadras* e depois partir para os *pares* (haveria uma quadra nesse exemplo se houvesse o mintermo 6 na expressão, que geraria a simplificação f(A,B,C)= B).

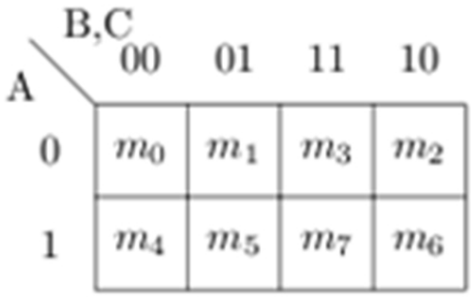


Figura 4.13. Construção do Mapa de Karnaugh para três variáveis de entrada.

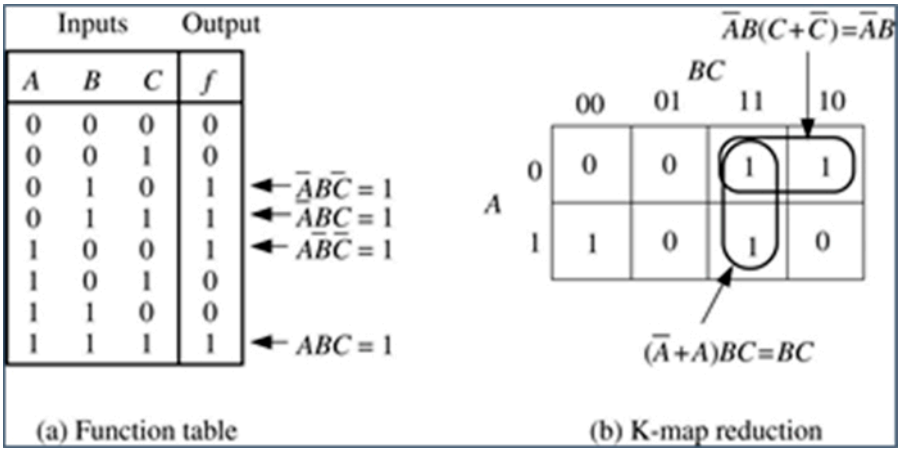


Figura 4.14. Simplificação com Mapa de Karnaugh para três variáveis de entrada (Fonte: Uyemura, 2002).

As Figuras 4.15 e 4.16 apresentam exemplos para 4 variáveis de entrada. Note que o mapa pode ser considerado de uma forma em que as extremidades podem ser conectadas para realizar a simplificação.

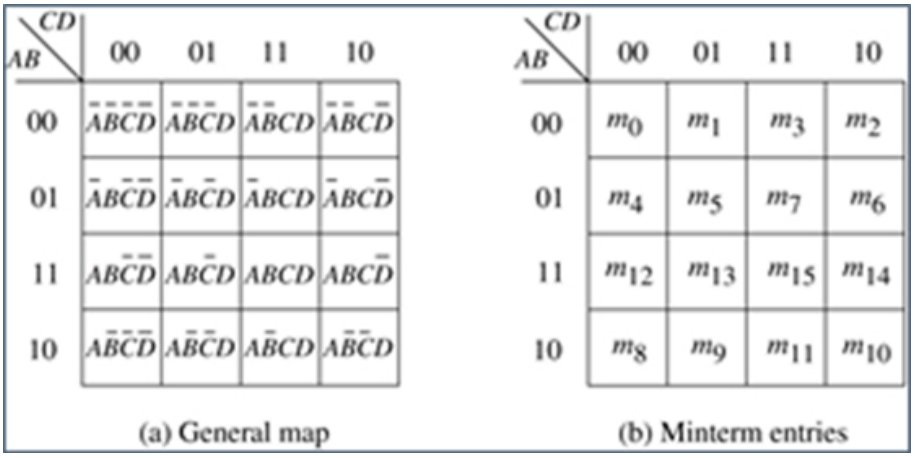


Figura 4.15. Mapa de Karnaugh para quatro variáveis de entrada (Fonte: Uyemura, 2002).

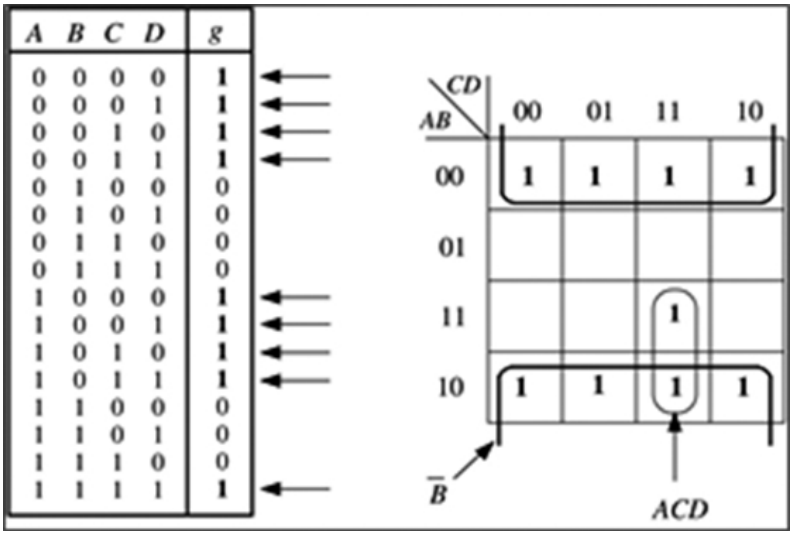
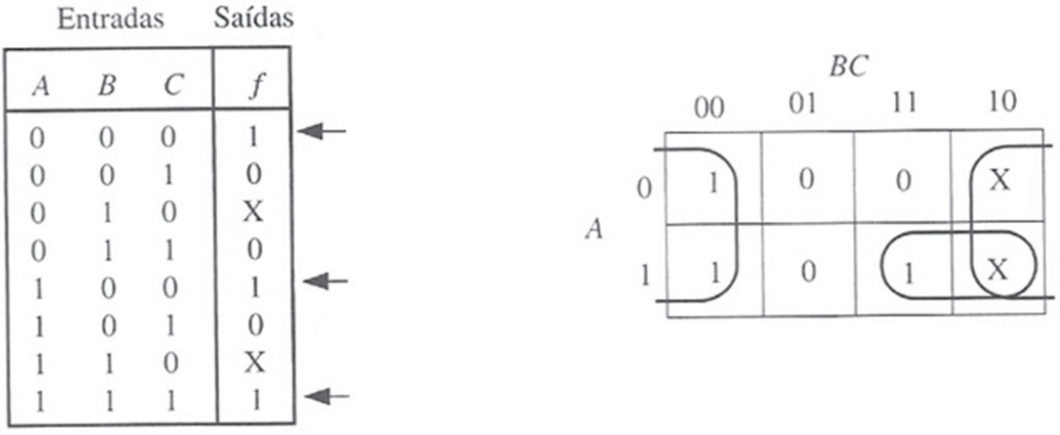


Figura 4.16. Simplificação com Mapa de Karnaugh para quatro variáveis de entrada (Fonte: Uyemura, 2002).

A técnica de mapas de Karnaugh pode ser utilizada em funções com qualquer número de variáveis. Apesar disso, a sua utilização torna-se mais complicada à medida que o número de variáveis aumenta. Na prática, mapas de karnaugh são empregados para até quatro entradas (tabelas verdade de 16 linhas). Uma característica importante de mapas de Karnaugh é o fato da utilização de *don´t care*.

Uma saída ***don´t care*** ou **condição irrelevante**, representada no mapa de Karnaugh com a letra "X", especifica uma saída do circuito que pode ser igual a 1 ou 0. Alguns circuitos podem ter tal característica. Quando o "X" é encontrado no mapa ele pode ser empregado assim como o "1" para criar pares, quádruplas, etc., de forma a melhorar a simplificação. Apesar disso, um "X" no mapa de Karnaugh não precisa gerar um termo na expressão final simplificada, de forma obrigatória, como é o caso de uma saída igual a "1". Podemos interpretar o "X" como uma ajuda ou um "coringa" no mapa.

A Figura 4.17 ilustra um exemplo completo de mapa de Karnaugh com o uso de condições irrelevantes. Perceba que há duas condições irrelevantes que são usadas no mapa para criar uma quadra e também para criar um par. O aproveitamento das condições irrelevantes no mapa permite obter expressões menores para a tabela verdade.



Entradas			Saídas
A	*B*	*C*	*f*
0	0	0	1
0	0	1	0
0	1	0	X
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	X
1	1	1	1

A \ *BC*	00	01	11	10
0	1	0	0	X
1	1	0	1	X

Figura 4.17. Tabela verdade com condições irrelevantes para os mintermos 2 e 6. A simplificação gera a expressão **f(a,b,c) = AB + C´ .**

Comumente, além do uso de expressões booleanas, adota-se a representação gráfica para a especificação de circuitos lógicos. Essa representação gráfica, chamada de portas lógicas, é discutida a seguir.

SEÇÃO 3
PORTAS LÓGICAS

Na unidade anterior (3), vimos como os bits podem ser empregados para representar os dados, sejam dados numéricos ou alfanuméricos. O assunto desta unidade é explorar os sistemas digitais que executam operações nessas variáveis binárias. Por exemplo, como é possível construir um circuito somador que realize a soma de duas palavras de 4 bits? O tijolo básico para a construção de circuitos digitais são as **portas lógicas** (*logic gates*).

Portas lógicas são circuitos digitais simples que recebem uma ou mais entradas e produzem uma saída binária. Saída binária significa que a saída pode ser igual a 0 ou igual 1, dependendo do tipo da porta lógica.

As portas lógicas têm uma representação gráfica associada, de forma que as entradas são colocadas à esquerda e as saídas à direita do diagrama. A relação entre as entradas e a saída das portas lógicas pode ser relacionada em uma tabela verdade ou descrita por uma equação conforme a álgebra de Boole.

Portas lógicas são implementadas fisicamente no hardware do computador, principalmente por meio de transistores. Transistores é um componente eletrônico que se popularizou a partir da década de 1950, revolucionou a eletrônica e permitiu todo o desenvolvimento do hardware e da computação. O transistor funciona como uma chave que pode permitir que a corrente (bit 1) passe ou não pelo circuito. O tempo de chaveamento é muito reduzido, de forma que transistores, dependendo da tecnologia adotada podem "chavear" a corrente em uma frequência na ordem de GigaHertz.

A Figura 4.18 ilustra a porta lógica para a operação NOT, simplesmente chamada como porta lógica NOT ou também chamada de inversora. Essa porta tem uma entrada e uma saída. A tabela verdade demonstra o comportamento da porta. Caso a entrada seja igual a 0, a saída é igual a 1; caso contrário, quando a entrada é igual a 1, a saída é igual a zero.

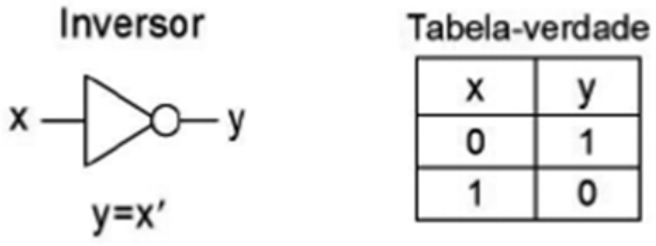


Figura 4.18. Porta Inversora (Fonte: Pedroni, 2010).

A maior parte das portas, ao contrário da porta inversora (NOT), emprega duas entradas. A Figura 4.19 ilustra a porta AND ou "e" lógico. Essa porta realiza a operação lógica "e" entre duas variáveis binárias, de forma que a saída será igual a 1 se e somente se as duas entradas forem iguais a 1.

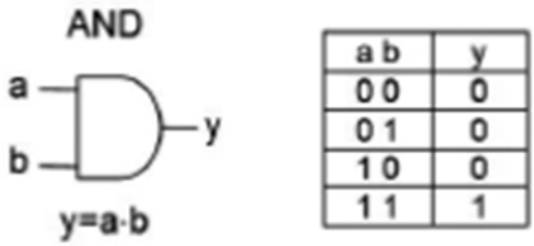


Figura 4.19. Porta AND (Fonte: Pedroni, 2010).

Outro exemplo de porta de duas entradas é a porta OR, ou "ou" lógico (Figura 4.20):

Figura 4.20. Porta OR (Fonte: Pedroni, 2010).

As portas lógicas AND e OR têm variantes nas quais a sua saída é invertida, realizando as operações NAND e NOR, respectivamente (Figuras 4.21 e 4.22). Note que para os dois casos, as saídas das tabelas verdade são ao contrário da função elementar associada (AND ou OR). Portas NAND ou portas NOR formam um **conjunto lógico completo ou universalidade**. Um conjunto lógico completo é um conjunto de portas que pode implementar qualquer função lógica. Além disso, a tecnologia que permite implementar portas lógicas fisicamente tem seus elementos eletrônicos mais simples se comportando como portas NAND ou como portas NOR. A Figura 4.23 ilustra a universalidade das portas NOR e NAND, demonstrando como as demais portas podem ser obtidas por meio de manipulações da álgebra de Boole.

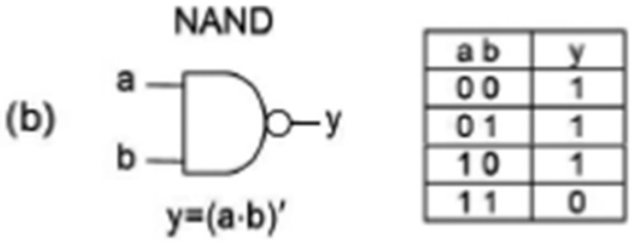


Figura 4.21. Porta NAND (Fonte: Pedroni, 2010).

Figura 4.22. Porta NOR (Fonte: Pedroni, 2010).

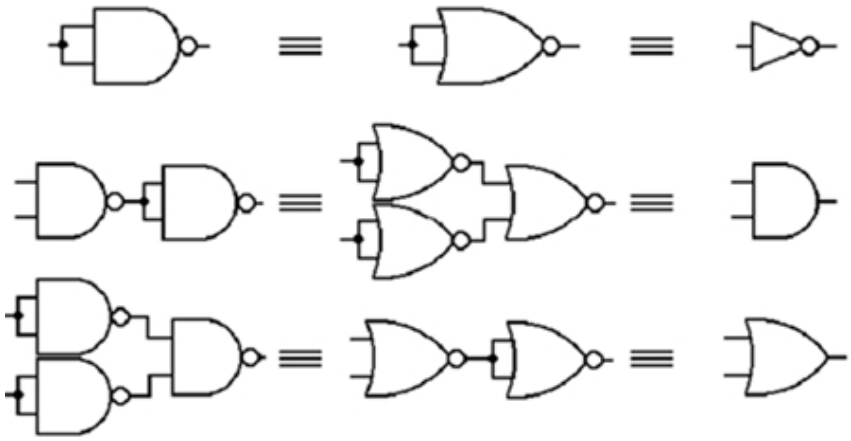


Figura 4.23. Universalidade das portas NAND e NOR

Outra porta lógica frequentemente empregada para a implementação de circuitos digitais é a porta XOR. A porta XOR realiza a operação "ou exclusivo". Tal porta é ilustrada pela Figura 4.24. Sua porta complementar é a porta XNOR, que é ilustrada pela Figura 4.25.

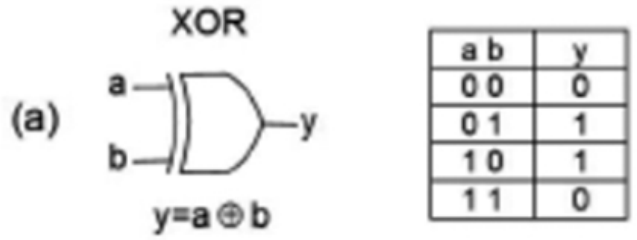


Figura 4.24. Porta XOR (Fonte: Pedroni, 2010).

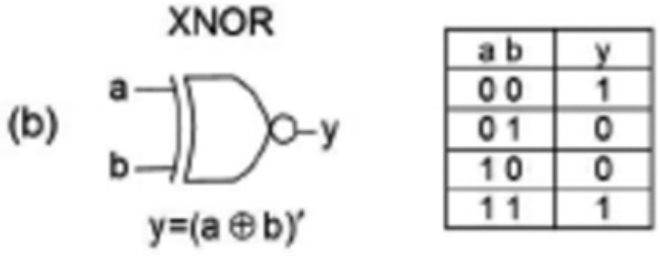


Figura 4.25. Porta XNOR (Fonte: Pedroni, 2010).

Com o uso das portas lógicas, circuitos digitais podem ser especificados e construídos. Para isso, as portas lógicas são interligadas para que o comportamento do circuito obedeça o estabelecido por uma tabela verdade ou por uma expressão lógica. Por exemplo, a Figura 4.26 ilustra um circuito digital chamado de somador parcial (maiores detalhes sobre circuitos somadores serão vistos na próxima Unidade de Estudo).

O circuito tem duas entradas, A e B, que são as parcelas a serem somadas. Note que a saída S recebe a operação XOR das entradas. Ou seja, caso uma ou outra entrada seja igual a um, o resultado S é igual a 1 (representando ou a soma 0+1, ou a soma 1+0). Caso contrário, a saída S é igual a zero (representando ou a soma 0+0, ou a soma 1+1). Quando ambos os valores de A e B forem iguais a 1, a saída C será ativada, indicando que houve uma soma de 1+1 e que o "vai-um", representado pela variável C, é obrigatório para ser considerado em na próxima parcela da soma. Perceba os círculos fechados nas entradas indicando uma ligação entre as linhas (que representam os fios elétricos).

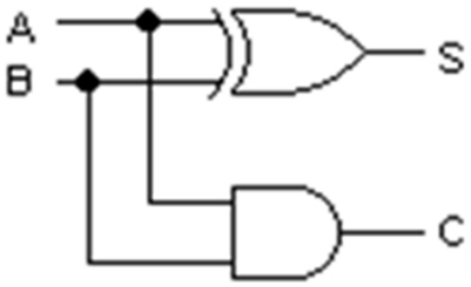


Figura 4.26. Exemplo de circuito digital especificado com portas lógicas XOR e AND.

Apesar das portas lógicas oferecerem uma visão prática, tais representações correspondem a uma visão simplificada de fenômenos físicos do mundo real. Por exemplo, olhando para os diagramas das Figuras 5.13 a 5.20 podemos estabelecer a relação do valor de saída pelas entradas, mas os diagramas não nos dizem em quanto tempo a saída y estará disponível após a aplicação das entradas a e b. Ou seja, o tempo de processamento dos dados de entrada. Fenômenos físicos, comumente estudados na eletrônica, estabelecem os tempos e parâmetros de corrente e de voltagem para o funcionamento das portas.

Cada tecnologia da área da eletrônica tem valores distintos entre si para esses parâmetros. Embora então os sistemas digitais (e seus tijolos básicos, as portas lógicas) empregarem variáveis discretas, na realidade as variáveis são representadas por sinais contínuos como valores de voltagem em fios. O mapeamento de variáveis contínuas para variáveis discretas é feito por meio da definição de **níveis lógicos.**

A Figura 4.27 ilustra um exemplo de mapeamento de valores de voltagem para sinais discretos em duas tecnologias comuns para a construção de circuitos integrados: TTL (*Transistor-transistor logic*) e CMOS (*Complementary Metal Oxide Semicondutor Logic*). Dessa forma, considerando como exemplo a tecnologia CMOS, se o circuito receber como entrada valores no intervalo 0 a 1,5 V, o sistema digital considera tal valor como sendo o 0 (zero) lógico; se o sistema digital recebe como entrada valores de voltagem entre 3.5 e 5 V, o sistema considera como sendo o 1 (um) lógico. Se por algum motivo o sistema digital receber como entrada um valor de voltagem entre 1.5 e 3.5 V, o comportamento

do circuito não será previsível. Assim, o sistema digital deve garantir que o sistema não opere no nível indeterminado. Felizmente, cada tecnologia tem uma margem de ruído na qual o valor é considerado aceitável (por exemplo, um valor de voltagem igual a 1.6V ainda seria considerado igual a nível lógico zero).

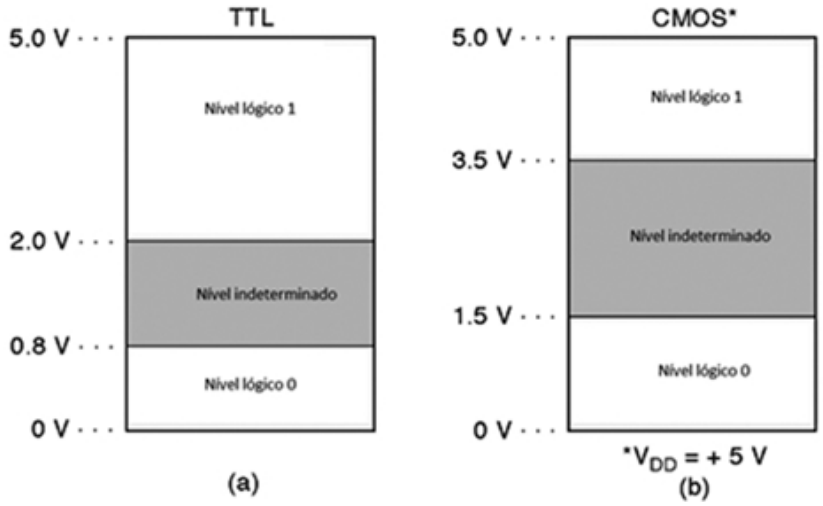


Figura 4.27. Tecnologias e níveis lógicos (Fonte: o Autor).

Cabe salientar que: a) para a mesma tecnologia, os níveis lógicos de entrada podem ser diferentes dos níveis lógicos de saída e; b) a tecnologia empregada como exemplo tem outras possibilidades de níveis lógicos quando utiliza níveis de voltagem de alimentação do circuito diferentes (Harris & Harris, 2013).

SEÇÃO 4
LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Além das portas lógicas, circuitos lógicos podem ser descritos com o uso de **linguagens de descrição de hardware.** Linguagens de descrição de hardware (HDLs) fornecem uma forma eficiente para o projeto auxiliado por computador (CAD) de sistemas lógicos digitais. Por meio de HDLs, o projetista pode descrever circuitos digitais da mesma forma que os programadores escrevem programas, ou seja, por meio de uma linguagem. A descrição do circuito em uma linguagem pode ser analisada por meio de software e através de ferramentas de software, o circuito pode ser sintetizado (gravado) em hardware. Uma das linguagens de descrição de hardware mais empregadas é a VHDL (*Very high speed integrated circuit*), que foi uma demanda do governo estadunidense, na década de 1980, para acelerar o projeto de hardware. Outra linguagem comumente adotada é a linguagem Verilog (Pedroni, 2010).

VHDL é uma linguagem padronizada e de alto nível (ou seja, mais facilmente entendida pelo projetista) que permite a descrição do comportamento de circuitos lógicos mais complexos por meio componentes mais simples. A descrição do circuito em VHDL é sintetizado através de um conjunto de portas lógicas e elementos de memória.

A Figura 4.28 ilustra o código fonte da descrição da função lógica OR. Da mesma forma que as demais formas de projeto de circuitos lógicos, inicia-se relacionando as entradas e as saídas, por meio da palavra chave **entity.** A palavra reservada **port** permite estabelecer as entradas e as saídas. Nesse exemplo, temos as entradas **a** e **b,** e a saída **f.** O tipo das entradas e da saída é **bit**, que indica que a função trabalha com valores de 1 bit.

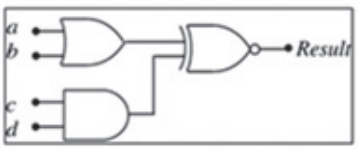
A palavra reservada **entity** declara uma *caixa preta*, ou seja, temos as entradas e as saídas, mas não definimos ainda *oque* o circuito faz. Note a semelhança entre a palavra reservada **entity** com o diagrama de bloco. Para definirmos *oque* o circuito vai realizar, usamos a palavra reserva **arquitecture**. No exemplo, estamos criando uma **arquitecture** chamada

de Lógica para a entidade que criamos previamente, chamada de primeiro exemplo. Entre o **Begin** e o **end** declaramos o comportamento do circuito. Nesse caso, a linha f <= a or b, define que a porta de saída (ou linha de saída) **f** recebe a operação **or** entre as entradas **a** e **b**.

```
-- função OR
entity primeiro_exemplo is
        port(a,b: in bit;
             f: out bit );
end primeiro_exemplo;
arquitecture Logica of primeiro_exemplo is
Begin
        f <= a or b;
end Logica;
```

Figura 4.28. Exemplo de descrição de hardware em VHDL (Adaptado de Uyemura (2002)).

A Figura 4.29 ilustra um exemplo um pouco mais complexo de circuito lógico descrito na linguagem VHDL. Mais a esquerda, tem-se a representação do circuito por meio de portas lógicas. Note que a saída do circuito, representado pela variável **Result** é uma expressão lógica com o uso de parênteses para definir a prioridade entre as portas. A linguagem VHDL é muita rica em relação às construções e as formas de especificar os circuitos lógicos. Embora seja um assunto mais relacionado com a área de sistemas digitais e de Engenharia de Computação, alunos de licenciatura que tenha interesse no assunto podem consultar o livro *VHDL cookbook*, disponível gratuitamente na Internet (veja na seção *Saiba Mais*).




```
-- segundo exemplo
entity segundo_exemplo is
        port(a,b,c,d: in bit;
             Result: out bit );
end segundo_exemplo;
arquitecture Logica of segundo_exemplo
is
Begin
Result <= (a or b) xnor (c and d);
end Logica;
```

Figura 4.29. Exemplo de descrição de hardware em VHDL de um circuito lógico (Adaptado de Uyemura (2002)).

Para quem tiver mais interesse sobre VHDL, sugere-se a leitura do VHDL Cookbook: http://bit.do/iocuni4.

Para fixar os conceitos apresentados nesta Unidade, realize as atividades abaixo:

1. Escreva a tabela verdade da função AND.

2. Escreva a tabela verdade da função NOT.

3. Sendo A=1 e B=0, calcule X=AB.

4. Sendo A=01101101 e B=11101001, calcule X=A.B. (Como calcular? A operação lógica deve ser realizada bit a bit entre as duas palavras.).

5. Sendo A=10111 e B=11001, calcule X=A.B.

6. Considerando os valores obtidos no item anterior, calcule Y = XAB.

7. Sendo A=02CD e B=3FA0, calcule X = A.B.

8. Considerando os mesmos valores para A e B, dois números em hexadecimal, calcule X = NOT A.B.

9. Sendo A=11CD e B=AA01, dois números em hexadecimal, calcule X = NOT AB.

10. Sendo e , dois números em octal, calcule X = NOT AB.

11. Calcule na expressão: ALFA = ALFA AND 0, o valor final de ALFA (ALFA é um valor binário).

12. Construa a tabela verdade para a expressão lógica Y = XAB.

13. Um sistema de alarme residencial (A) deve ser disparado quando o sinal do detector de presença (C) for ativado e quando não existirem moradores presentes (M) na casa. Sendo A, C e M sinais binários que representam respectivamente o disparo do alarme, o sinal de presença e o sinal de moradores na casa, construa um circuito lógico para representar tal sistema de alarme.

14. Sendo A=1 e B=0, resolva X = NOT AB.

15. Um cachorro se coça quando existem pulgas e o cachorro não foi tratado com um remédio anti-pulgas. Sendo C o sinal que indica a coceira do cachorro, R que indica que o cachorro foi tratado com remédio anti-pulgas e P o sinal que indica que o cachorro tem pulgas, construa um circuito para obter o sinal C a partir das regras criadas para indicar a coceira.

16. Considerando uma palavra de 16 bits, defina o valor de B, em hexadecimal, na expressão: A = B.C para que a variável A receba apenas os 4 bits mais significativos do valor de C e tenha os demais bits menos significativos instanciados sempre com o valor zero.

17. Desenvolva a expressão lógica que represente a seguinte afirmação: "O alarme soará se for recebido um sinal de falha juntamente com um sinal de parada ou um sinal de alerta".

18. Desenhe, através de portas lógicas, o circuito correspondente à afirmação anterior.

19. Desenvolva a expressão lógica que decida o que Bruno deve fazer em relação ao seguinte problema:

(a) Bruno deverá ir ao jogo de futebol se e somente se Felipe for à praia e Roberta for estudar.

(b) Roberta estudará se Renata ou Patrícia trouxerem o livro.

(c) Renata concorda em trazer o livro, mas Felipe não quer ir à praia.

O que Bruno deverá fazer?

20. Desenhe, através de portas lógicas, o circuito correspondente à expressão anterior.

21. Mostre como uma porta NAND pode ser usada como uma porta inversora (NOT).

22. Mostre como uma porta AND pode ser montada a partir de duas portas NAND.

23. Qual é a operação lógica é realizada pelo circuito abaixo?

24. Considere as seguintes palavras binárias:

(a) `A=1011`

(b) `B=1110`

(c) `C=0011`

(d) `D=1010`

Obtenha o valor de X nas seguintes expressões lógicas:

(a) `X = A . ( B + C )`

(b) `X = ( A + B ) . ( C + ( A + D ) )`

(c) `X = B . C . A + ( C + D )`

(d) `X = A + B + C . B + A`

25. Desenhe os diagramas com portas lógicas das expressões da questão anterior.

26. Implemente um circuito combinacional de 3 entradas para a função maioria: essa função retorna 1 quando a maioria das entradas tem valor igual a 1. O projeto do circuito deverá conter:

- *Tabela verdade da função*
- *Descrição da expressão lógica na forma de soma de produtos (SDP)*
- *Simplificação através do mapa de Karnaugh*
- *Especificação do projeto com o uso de portas lógicas*

1. Idem, para a função minoria.


ANOTAÇÕES

UNIDADE V

Componentes Lógicos Combinacionais

Objetivos de aprendizagem

- Entender como é o projeto de circuitos lógicos digitais.
- Entender como é o projeto hierárquico de componentes digitais.
- Conhecer componentes básicos de circuitos lógicos digitais.

Roteiro de estudos

- SEÇÃO 1 – Projeto de circuitos lógicos
- SEÇÃO 2 – Circuito somador
- SEÇÃO 3 – Circuito multiplexador
- SEÇÃO 4 – Circuito decodificador
- SEÇÃO 5 – Circuito comparador

UAB UNIVERSIDADE ABERTA DO BRASIL

PARA INÍCIO DE CONVERSA

Prezada(o) aluna(o)

Nesta unidade, você terá a oportunidade de exercitar os conceitos de lógica digital, vistos até aqui, para projetar circuitos combinacionais. Você conhecerá também os circuitos combinacionais mais comuns em sistemas digitais e na organização dos computadores.

Bom estudo!

SEÇÃO 1
PROJETO DE CIRCUITOS LÓGICOS

Nas áreas da eletrônica digital e da computação, um **circuito lógico** é uma rede composta de portas lógicas que processa variáveis binárias. Um projeto de circuito lógico (também chamado de lógica digital ou como circuito digital) pode ser resumido através dos seguintes passos: a) Especificação do problema; b) Criação da lógica que implementa as funções desejadas; c) construção do circuito; d) Teste e verificação do circuito (Uyemura, 2002).

Um circuito lógico pode ser visto como uma caixa preta, com: a) uma ou mais entradas, uma ou mais saídas; b) uma ou mais saídas; c) uma especificação funcional que define o comportamento, ou seja, a relação entre as entradas e as saídas; d) e, como comentado na seção anterior, uma especificação temporal que define o atraso para que as saídas estejam disponíveis a partir do momento que as entradas estão disponíveis (Harris e Harris, 2013).

A Figura 5.1 ilustra um exemplo de somador parcial que realiza a soma entre as entradas (bits) A e B, produzindo como resultado a soma S e o *vai-um* para a próxima parcela. Em um primeiro momento, é interessante relacionar tais entradas. A especificação do circuito digital interno (à caixa preta) pode ser realizada por meio de expressões booleanas, portas lógicas ou por uma linguagem de descrição de hardware.

Figura 5.1. Um circuito lógico como uma caixa preta.

Circuitos lógicos podem ser classificados como combinacionais e sequenciais. A saída de um **circuito combinacional** depende única e exclusivamente de suas entradas, ou seja, o circuito implementa uma função que pode ser descrita por uma tabela verdade. Por exemplo, uma simples porta lógica pode ser vista com um circuito digital combinacional, de forma que a sua saída depende apenas das entradas. Já os circuitos sequencias produzem saídas que dependem das entradas e de seu estado corrente: circuitos sequenciais implementam uma memória interna que armazena um **estado interno**. O nome *sequencial* vem desse fato: a saída produzida em um determinado instante de tempo depende da sequência de entradas aplicada no circuito, e não apenas na entrada atual (corrente). Pense em um exemplo de circuito sequencial: um cofre eletrônico que abrirá a porta após receber uma *sequência* correta de valores numéricos. Por exemplo, se a senha tem 4 dígitos e é igual a 1234, quando o usuário digitar o número 4 (entrada corrente), não será apenas o número quatro que abriu a porta do cofre e sim a sequência 1234 completa. Nesta unidade, trataremos de circuitos combinacionais. Circuitos sequenciais serão discutidos na próxima unidade.

Em computação, os circuitos combinacionais permitem realizar operações aritméticas como soma e subtração, assim como atuam para selecionar e controlar os elementos internos de uma unidade lógica e aritmética. Esta unidade apresenta os circuitos lógicos mais comuns para o projeto de computadores digitais.

Como podemos projetar um circuito combinacional? Para responder essa pergunta, vamos analisar o seguinte exemplo:

Exemplo: Crie um circuito digital que considerando uma entrada de 3 bits que representa um valor numérico inteiro sem sinal, gere a saída igual a 1 quando o valor de entrada for maior que 4 (decimal). Começamos o nosso projeto com a tabela verdade, relacionando as entradas **a**, **b**, **c** com a saída **f**. Como a função tem 3 entradas, temos que a tabela tem $2^3 = 8$ linhas:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Note que para os valores de entrada 101, 110 e 111, que são maiores que 4 em decimal, temos a saída igual a 1. O próximo passo é a simplificação por meio de mapa de Karnaugh (Figura 5.2). Como o circuito tem só uma saída, empregamos um único mapa (o número de saídas do circuito é igual à quantidade de mapas a serem empregados).

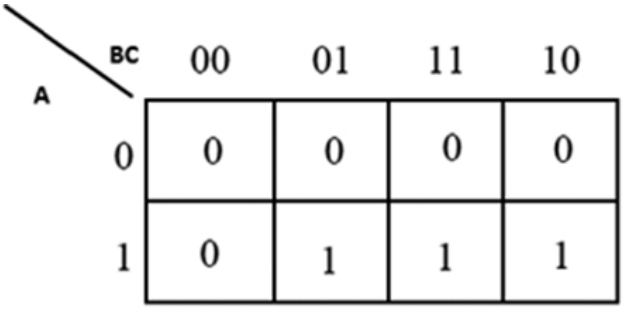


Figura 5.2. Mapa de Karnaugh para o exemplo.

A partir do mapa, procuramos quadras inicialmente; como não há quadras, procuramos pares. Nesse caso, temos o par em azul e o par em vermelho (Figuras 5.3 e 5.4).

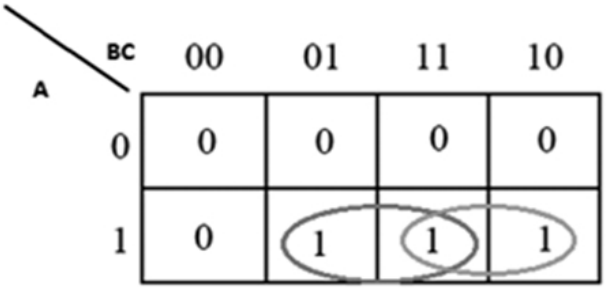


Figura 5.3. Pares circulados no Mapa de Karnaugh do exemplo.

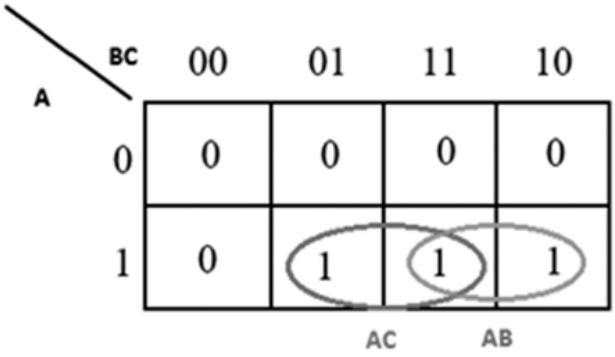


Figura 5.4. Simplificação final do mapa de karnaugh para o exemplo.

Nossa expressão final, que representa a função **f,** é **F(A,B,C) = AB + AC**. A partir deste ponto, podemos especificar o circuito por meio de portas lógicas. É usual representar as entradas nas linhas verticais e o círculo fechado nas linhas para indicar a ligação dos fios (Figura 5.5).

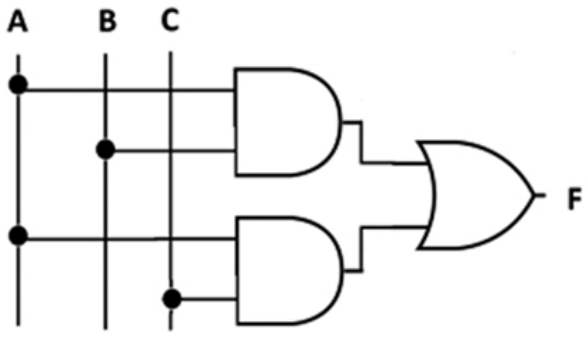


Figura 5.5. Circuito lógico simplificado para o exemplo

Para expressões maiores, que necessitem também das entradas na forma complementar (negadas), pode-se adotar a representação com linhas adicionais já com as portas inversoras (Figura 5.6):

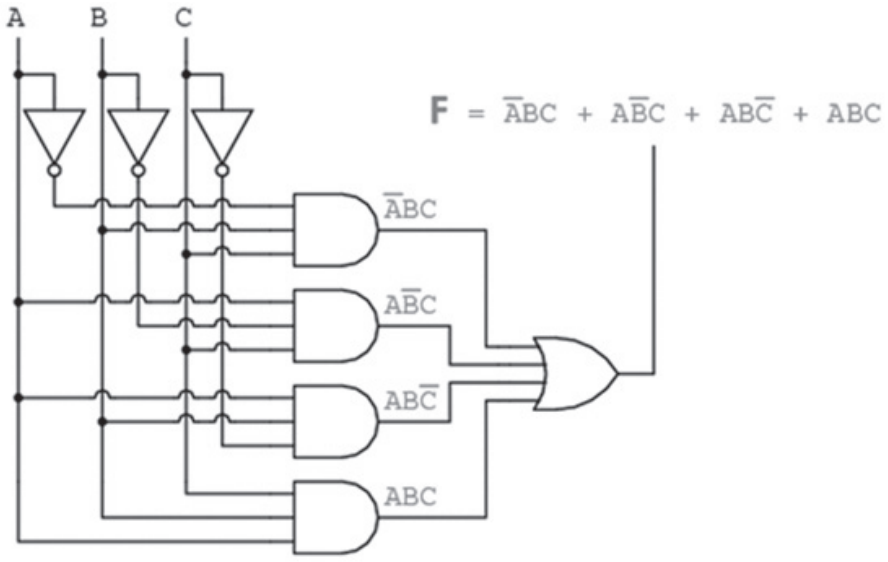


Figura 5.6. Exemplo de circuito lógico com as linhas normais e complementares de entrada.

SEÇÃO 2
CIRCUITO SOMADOR

Um circuito somador em computação realiza a soma de duas palavras. Por exemplo, um computador de 16 bits frequentemente realiza soma entre duas palavras de 16 bits. Para a definição de um circuito somador, são necessários dois componentes: um circuito somador parcial e um conjunto de circuitos somadores completos. Por exemplo, para um computador realizar a soma de duas palavras de 16 bits, serão necessários 1 circuito somador parcial e 15 circuitos do tipo somador completo. Então, qual a diferença entre os circuitos somador parcial e completo?

Um circuito somador parcial realiza a soma entre dois bits, ou seja, o circuito tem duas entradas. Operando de maneira similar a soma decimal, o circuito tem duas saídas: o resultado da parcela e o *vai-um.* O *vai-um* tem como saída o valor 1 quando a soma ultrapassa o valor 2 em decimal (10 em binário). Assim, abaixo tem o resultado de algumas somas em binário:

```
  0     0     1
+ 0   + 1   + 1
-----------------
0 0   0 1   1 0
```

O circuito somador parcial gera *vai-um*, mas não considera o *vai-um* de uma parcela anterior. O circuito somador completo calcula a soma, tendo como entradas não somente os dois bits das palavras e sim o *vem-um*, que corresponde ao *vai-um* de uma parcela a direita.

Bem vamos às tabelas verdade. Na Figura 5.7., segue a tabela verdade do circuito somador parcial. Note que temos as entradas A e B e as saídas S e C0.

A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figura 5.7. Tabela verdade do circuito somador parcial,

A representação do circuito por meio de portas lógicas está ilustrada na Figura 5.8. Conforme comentado anteriormente, o circuito somador completo realiza a soma de considerando três entradas: as duas mais óbvias A e B, as quais são os bits da palavra e o *vem*-um (C_i), que representa o *vai-um* da parcela mais a direita. A Figura 5.9 apresenta a tabela verdade para o circuito somador completo.

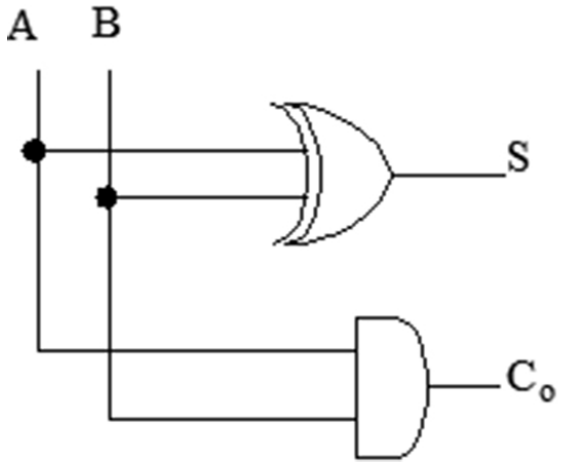
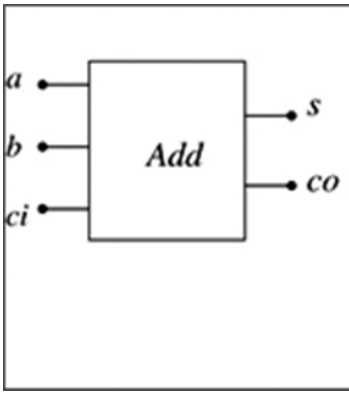


Figura 5.8. Circuito somador parcial especificado com o uso de portas lógicas.



a	*b*	*ci*	*s*	*co*
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Tabela 5.9. Diagrama de bloco e Tabela verdade do circuito somador completo.

A Figura 5.10 o mapa de Karnaugh e o circuito por meio do uso de portas lógicas. Note que o mapa de Karnaugh não pode ser simplificado com o uso de pares. Algebricamente, pode-se obter uma simplificação por meio de uma porta XOR (considere aqui o símbolo "#' para representar a operação XOR):

S = A'B'Ci + A'BCi'+ AB'Ci' + ABCi

S = A'B'Ci + ABCi + A'BCi'+ AB'Ci' // associativa

S = Ci(A'B'+AB) + Ci'(A'B+ AB') // distributiva

S = Ci(A # B)' + Ci'(A # B) // Consulte as tabelas verdade XNOR e XNOR

S = Ci # A# B

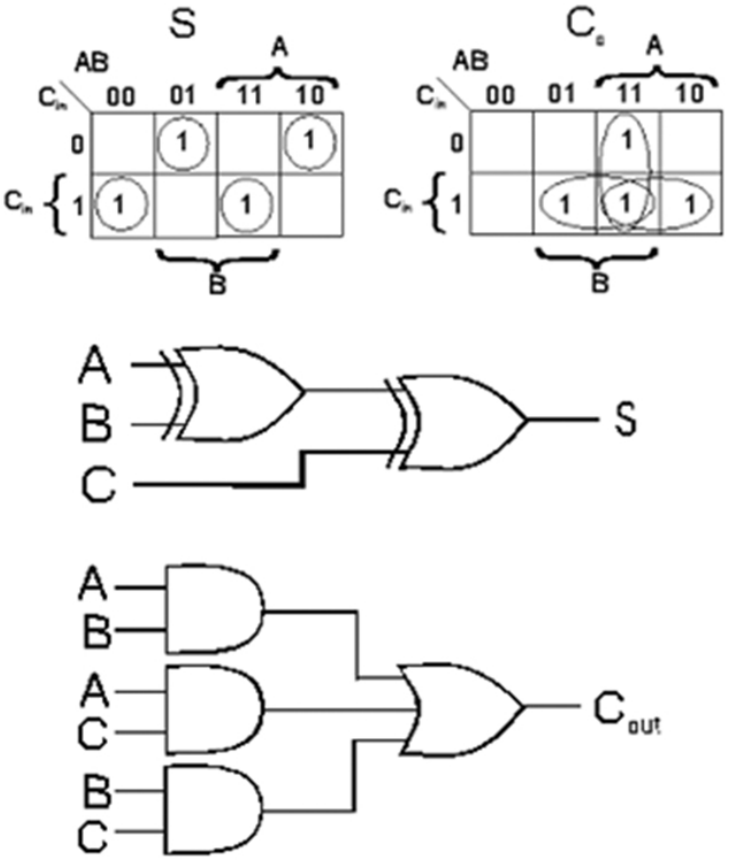


Figura 5.10. Circuito somador completo: mapa de Karnaugh e circuito com portas lógicas.

Conforme comentado previamente, para realizar uma soma se faz necessário associar blocos de somadores em série. Por exemplo, para obtermos um somador de 3 bits, deve ser empregados um somador parcial para os dois bits (A e B) menos significativos (mais a direita da palavra) e dois somadores completos para os demais bits. A Figura 5.11 ilustra o resultado dessa associação de circuitos somadores. Essa associação é comumente chamada de **projeto hierárquico**. No projeto hierárquico elementos mais simples são unidos para criar componentes mais complexos.

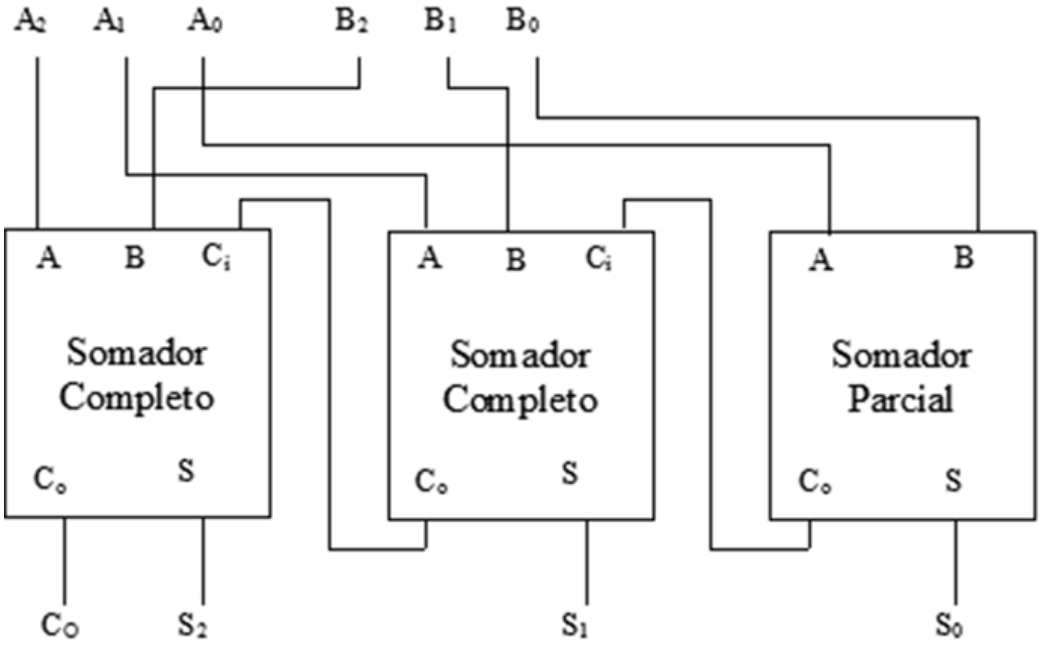


Figura 5.11. Circuito somador de três bits obtido pela associação de 2 somadores completos e 1 somador parcial.

Cabe salientar que existem outras formas de realizar as somas, como por exemplo a predição de vai-um (Murdocca, 2000). Empregando a técnica de predição de vai-um, o tempo para o processamento é reduzido, por meio de expressões algébricas que permitem obter os bits de *vem-um* dos somadores sem a necessidade de esperar o processamento em cadeia da sequência de somadores. Lembre-se que todo circuito digital tem um atraso, ocasionado pelo funcionamento dos componentes eletrônicos, de forma que se tivermos um somador de 32 bits, o último somador completo, da parcela mais significativa, deverá aguardar o processamento dos demais somadores para obter o seu valor de *vai-um*. O somador com predição atua dessa forma, eliminando a ligação em cadeia entre os somadores. Outra forma de realizar a soma é por meio de vários circuitos somadores em paralelo. Para o leitor que quiser se aprofundar nesse assunto, sugere-se a leitura de Patterson e Henessy (2012).

SEÇÃO 3
CIRCUITO MULTIPLEXADOR

O multiplexador ou MUX é um circuito combinacional que tem a finalidade de *selecionar*, através das *variáveis de seleção*, *uma* de suas *entradas*, conectando-a eletronicamente à sua *única saída*. Essa operação é denominada *multiplex* ou *multiplexação* que significa *seleção* e, tanto suas entradas como sua saída, são denominadas também, *canais* de entrada e saída.

Embora os exemplos a seguir empreguem canais de 1 bit de tamanho, internamente ao computador os canais por exemplo 32 ou 64 bits de tamanho, internamente no caminho de dados do processador (Patterson & Henessy, 2014). A Figura 5.4 ilustra o diagrama de bloco de um multiplexador de **n** entradas e de **m** linhas de seleção. A relação entre as variáveis **n** e **m** é a seguinte: $\mathbf{n = 2^m}$. Por exemplo, para um multiplexador de 16 entradas, serão necessárias 4 linhas de seleção (n=16 e m=4). É comum denotar um multiplexador na forma **MUX n:1**. Por exemplo, um multiplexador de 16 entradas é representado como MUX 16:1, indicando que a partir de 16 entradas (ou canais) haverá apenas uma saída (Figura 5.12).

Figura 5.12. Circuito Multiplexador como um diagrama de bloco.

Inicialmente, vamos considerar um multiplexador simples de duas entradas (E0 e E1) e com uma linha de seleção A. Como a seleção de entradas (A) não depende do valor das entradas, a tabela-verdade é abreviada, de forma que a coluna saída (S) apresenta os nomes das variáveis de entrada:

A	S
0	E0
1	E1

A Figura 5.13 ilustra o circuito multiplexador por meio de portas lógicas. Multiplexadores com mais entradas podem ser obtidos da mesma forma, aumentando o número de portas AND e inversoras, mantendo a porta OR mais próxima à saída. Por exemplo, considere agora um multiplexador de 4 entradas, de forma que **n**=4 e **m=**2. A Figura 5.14 ilustra o circuito, obtido a partir da expressão em SDP: F = S1′.S0′.P0 + S1′.S0.P1 + S1.S0′.P2 + S1.S0.P3.

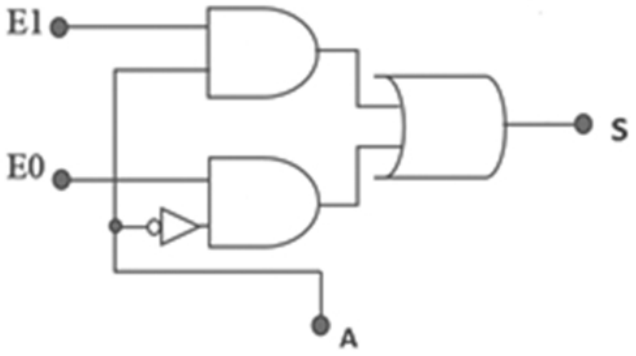


Figura 5.13. Circuito Multiplexador especificado por meio de portas lógicas (Fonte: o Autor).

Multiplexadores podem ser usados como elementos lógicos, para implementar funções genéricas: se os bits de seleção forem substituídos por variáveis, pode-se realizar o **and** lógico com os valores fixados em 0 e 1 para implementar funções lógicas. Como exemplo, a Figura 5.15 apresenta a função OR implementada com o uso de um multiplexador. De forma similar, a Figura 5.16 apresenta as funções lógicas OR e XNOR implementadas com multiplexadores.

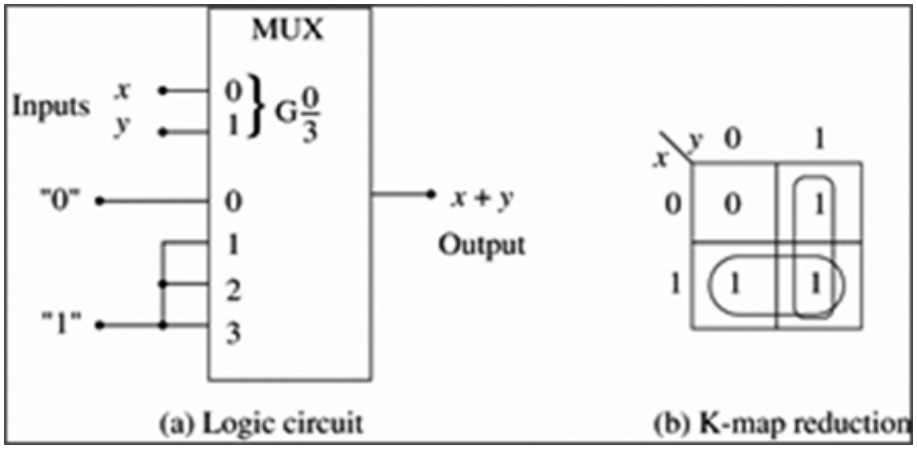


Figura 5.15. Circuito Multiplexador sendo usado para implementar a função **OR** (Fonte: Uyemura, 2002).

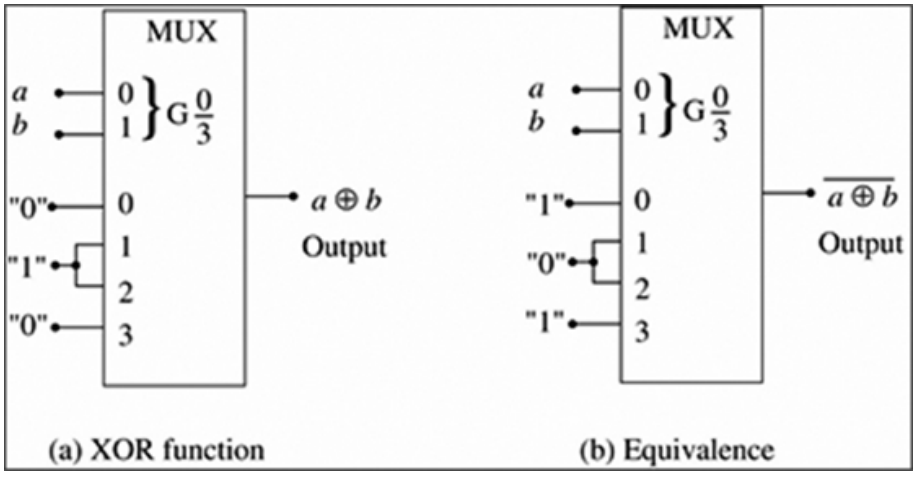


Figura 5.16. Circuito Multiplexador sendo usado para implementar as funções **XOR** e **XNOR** (Fonte: Uyemura, 2002).

Multiplexadores maiores (com mais entradas) podem ser obtidos com a associação de um conjunto de multiplexadores. A Figura 5.17 apresenta um multiplexador de 8 entradas construindo a partir de dois multiplexadores de 4 entradas e um multiplexador de duas entradas, observe que o bit mais alto da linha de seleção (representado pela letra A) controla entre os dois multiplexadores ligados as entradas, de forma que bit mais significativo da linha de seleção alterna entre os dois multiplexadores. As linhas B e C, que correspondem aos bits 0 e 1 da palavra de seleção são compartilhados entre os dois multiplexadores 4:1.

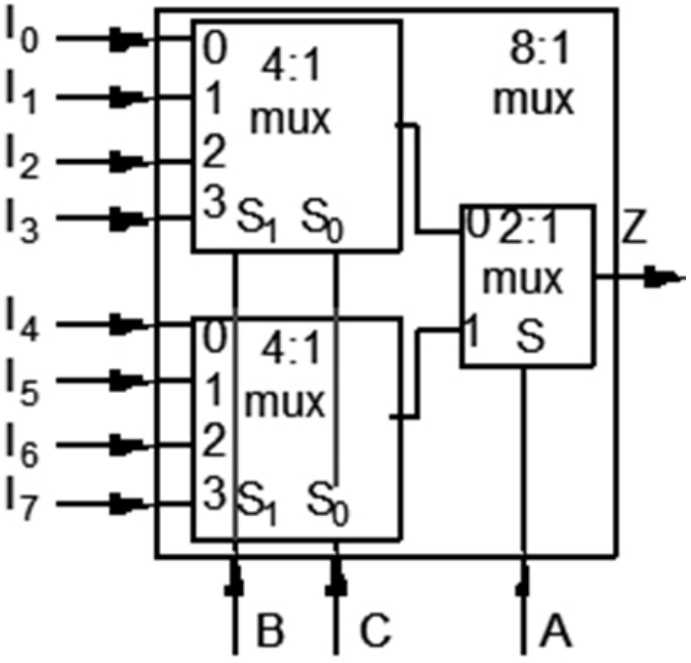


Figura 5.17. Circuito Multiplexador de 8 entradas obtido por meio da associação de multiplexadores (Fonte: o Autor).

SEÇÃO 4
CIRCUITO DECODIFICADOR

Em circuitos de controle de memória é comum a utilização de decodificadores. Decodificador é um circuito que composto de n entradas e 2^n saídas. Para cada combinação de entrada, apenas uma das saídas será ativada (bit 1), enquanto que as outras linhas de saída serão inativas (bit 0). Utilizado para selecionar um conjunto de circuitos específicos através de uma combinação de bits (p.e. pastilhas de memória).

A Figura 5.18 apresenta o diagrama de bloco de um circuito decodificador, ressaltando a quantidade de entradas e saídas. Um decodificador com n entradas tem 2^m saídas. A Figura 5.19 apresenta o circuito lógico de um decodificador de 2 entradas e 4 saídas, com as entradas nomeadas de A0 e A1 e com as saídas nomeadas como D0 a D7. Um circuito decodificador com 2 entradas e 4 saídas é comumente chamado como decodificador 2x4.

Um circuito decodificador pode ser empregado para implementar funções lógicas. Por exemplo, considere as três funções abaixo:

F1 = A′.B.C′ + A′.B′.C.D + A.B.C.D
F2 = A.B.C′.D+ A.B.C
F3 = (A′+B′+C′+D′)

A Figura 5.20 ilustra como tais funções podem ser implementadas com um decodificador 4x16.

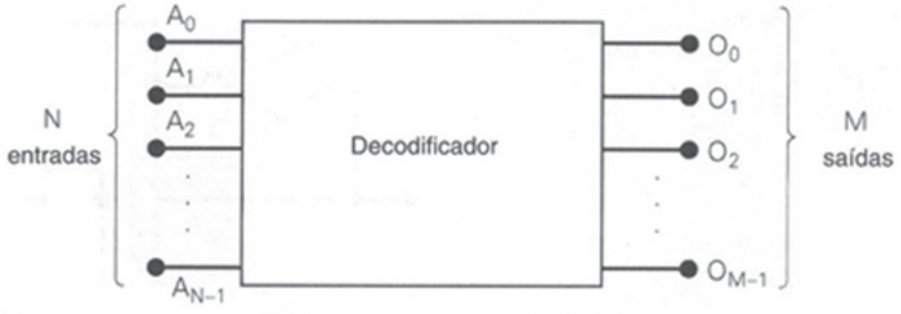


Figura 5.18. Circuito Decodificador (Fonte: o Autor).

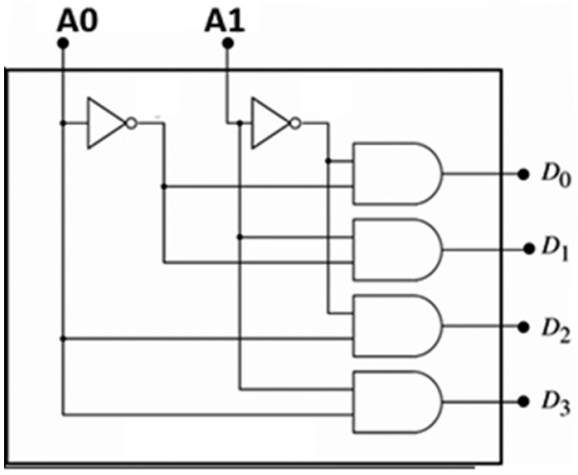


Figura 5.19. Circuito Decodificador 2x4 (Fonte: o Autor).

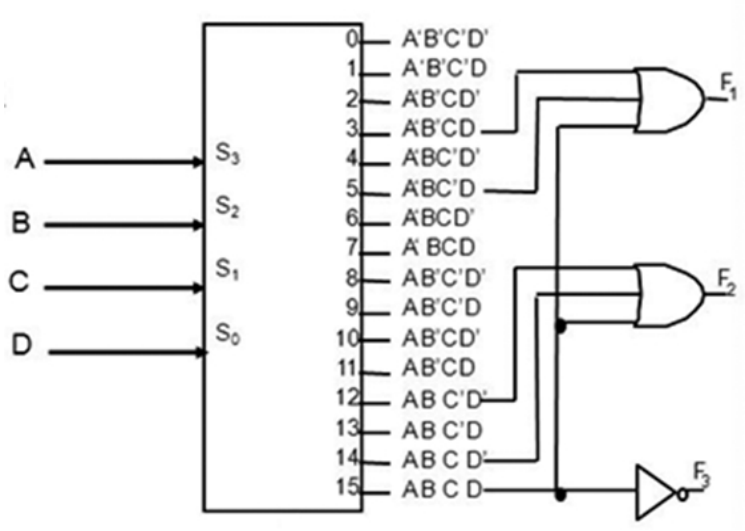


Figura 5.20. Circuito Decodificador para implementar funções lógicas (Fonte: o Autor).

SEÇÃO 5
CIRCUITO COMPARADOR

O circuito comparador compara duas palavras binárias e retorna um valor igual a 1 se as duas palavras são exatamente iguais. A Figura 5.21 ilustra o diagrama de bloco do circuito comparador, considerando a operação de comparação para duas palavras de 4 bits de tamanho, nomeadas de **a** e **b**, sendo **f** a saída do circuito.

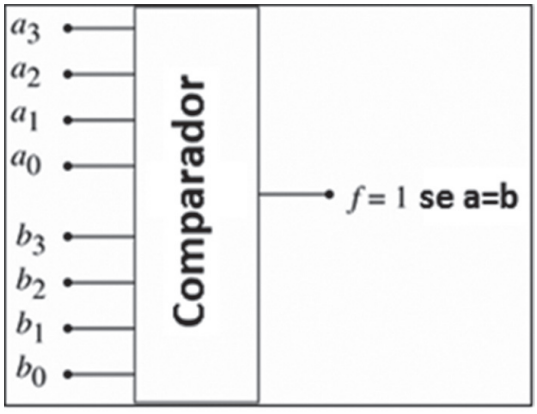


Figura 5.21. Circuito comparador (Fonte: o Autor).

O Circuito comparador é implementado realizando a operação XNOR entre cada um dos bits das palavras e realizando a união dos sinais em uma porta AND. Para que a saída da porta AND seja igual a 1, todas as entradas devem ser iguais, ou seja, todas as comparações, bit a bit, resultaram em valores iguais. A Figura 5.22 apresenta o circuito comparador para duas palavras de 4 bits. O computador emprega circuitos comparadores (com palavras de 32 e 64 bits de tamanho, por exemplo) para decidir se duas palavras são iguais e, com isso, executar decisões de desvio (*if, else, switch*) corretamente para o programa de computador que está sendo executado.

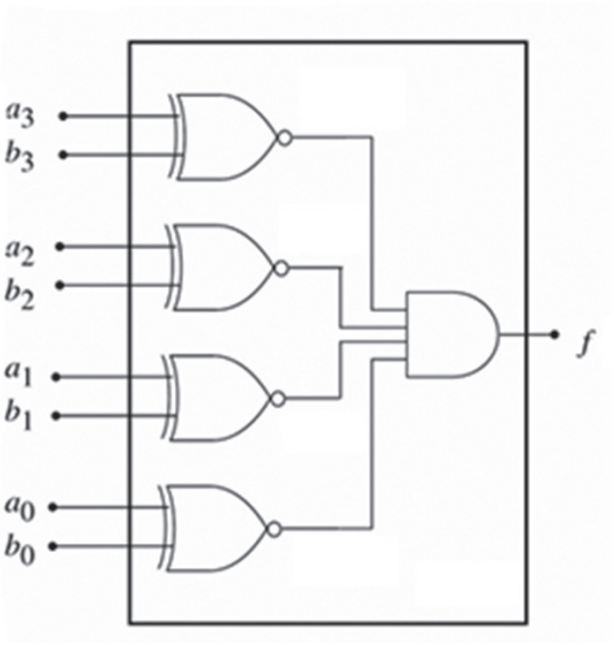


Figura 5.22. Circuito comparador implementado com portas lógicas (Fonte: o Autor).

Visite o site Play Hookey onde você poderá simular circuitos sequenciais: http://bit.do/iocuni6 .

Após ter estudado o conteúdo desta unidade, é hora de você realizar alguns exercícios para fixação.

1.Projeto um circuito combinacional que represente a função maioria de três entradas. O circuito deverá ter três entradas (**a**,**b** e **c**) e uma (**f**). A função maioria estabelece que se a maioria das entradas for igual a 1, a saída será igual a 1.

2. Projeto um circuito combinacional de três entradas (**a**,**b** e **c**) e de duas saídas (**f** e **g**). A saída **f** deverá ser igual a zero se o número binário, na entrada (composto por **a**,**b** e **c**), for maior que 4 e menor que 7 e a saída **g** será igual a 1 quando o número binário, na entrada, for maior ou igual a 5.

3. Projete um circuito combinacional que realize a divisão inteira (divisão inteira é aquela que não considera o resto) de duas palavras binárias A e B, sendo que cada palavra tem dois bits de tamanho. A saída do circuito é uma palavra binária de 2 bits. Projete uma saída adicional, chamada de *DZ*, que terá valor igual a 1 quando ocorrer divisão por zero. Por exemplo, se as entradas são A_1=1, A_0=0, B_1=1 e B_0=0, então as saídas serão: C_1=0, C_0=1 e *DZ*=0, pois não houve divisão por zero. O projeto do circuito deverá ter os seguintes elementos:

(a) Tabela verdade de 16 linhas com as quatro entradas (A_1, A_0, B_1 e B_0) e as cinco saídas relacionadas (C_1, C_0 e *DZ*);

(b) Simplificação das expressões de saída (C_1, C_0 e *DZ*) através da técnica de mapas de Karnaugh;

(c) Circuito com portas lógicas.

4.Mostre como a função **f(a,b) = a'+ b** pode ser implementada utilizando um multiplexador.

5.Mostre como construir um multiplexador 8×1 a partir de dois multiplexadores 4×1 e um multiplexador 2×1 (utilize apenas diagrama de blocos).


ANOTAÇÕES

UNIDADE VI

Circuitos lógicos sequenciais

Objetivos de aprendizagem

- Entender com é o projeto de circuitos lógicos sequenciais.
- Conhecer os circuitos lógicos *latch* e *flip-flop*.

Roteiro de estudos

- SEÇÃO 1 – Circuitos sequenciais
- SEÇÃO 2 – Registradores

UAB UNIVERSIDADE ABERTA DO BRASIL

PARA INÍCIO DE CONVERSA

Prezada(o) aluna(o)

Nesta unidade, você terá a oportunidade de conhecer elementos de memória que podem ser construídos com circuitos lógicos digitais. Você vai conhecer os elementos de memória *latch* e *flip-flop.*

Bom estudo!

SEÇÃO 1
CIRCUITOS SEQUENCIAIS

Na unidade anterior, vimos que circuitos combinacionais produzem uma saída por meio de uma função que depende apenas das entradas atuais. Por exemplo, o circuito comparador emprega portas XNOR e AND para produzir um resultado, considerando apenas os bits das duas palavras que estão sendo comparadas. Devido a esse fato, circuitos combinacionais podem representar todas as possibilidades de entradas e de saídas em tabelas verdade.

Nesta unidade, vamos estudar os **circuitos sequenciais**. Ao contrário de circuitos combinacionais, a saída de um circuito sequencial depende dos valores de entrada e também do estado interno (memória) do circuito. Notavelmente, tais circuitos são úteis para a construção de memórias no computador. De fato, as memórias cache dos processadores atuais são construídas dessa forma, por meio de circuitos sequenciais. Mais que isso, circuitos sequenciais permitem a construção de sistemas mais complexos, que detectam sequências válidas de bits ou sistemas que trabalham com

controle por meio de máquinas de estados (*finite state machines*). Antes de entrarmos no assunto principal desta Unidade, é importante conhecer um componente importante em sistemas digitais: o **clock**.

O **clock** é um sinal de controle em sistemas digitais que periodicamente faz uma transição de 0 para 1 e retornando novamente para o 0 (Figura 6.1). O *clock* repete esse comportamento em um período T, de forma que se pode calcular a sua frequência, pela equação **f = 1/T**. Sistemas digitais empregam o clock para sincronizar eventos. Sistemas computacionais operam com *clock* de frequência na ordem de GigaHertz, por exemplo 3 Ghz ou mais.

A utilização de um sinal de clock para controlar a operação de circuitos sequenciais permite controlar o momento no qual o estado interno (memória) do circuito é alterado. Com o uso do *clock,* sistemas digitais nos quais os dados necessitam ser movimentados de maneira síncrona podem ser construídos.

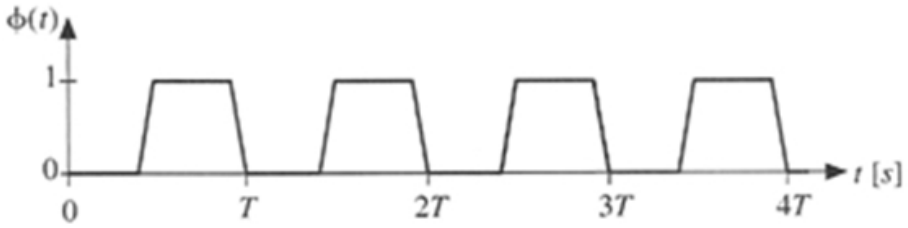


Figura 6.1. Sinal de *clock* (eixo y) em relação ao tempo (eixo x) (Fonte: Uyemura, 2002).

LATCHES E FLIP-FLOPS

Latches e flip-flops são elementos básicos para a construção de circuitos digitais sequenciais. Esses elementos são chamados de biestáveis, por permitir o armazenamento de um bit, na forma de 0 ou 1. Além de armazenar, os latche e flip-flops apresentam entradas pelas quais o circuito pode ser controlado para manter ou alterar o bit que está sendo armazenado internamente ao circuito.

Latch SR

O *latch* mais simples é o **Latch SR**, apresentado na Figura 6.2. Note a diferença principal desse circuito para com os circuitos combinacionais: existe a **realimentação** do circuito, de forma que: a) a saída da primeira porta é enviada como entrada para a segunda porta; b) a saída da segunda porta é enviada como entrada para a primeira porta. Essa diferença na forma como o circuito é construído é que permite a ocorrência do armazenamento de 1 bit.

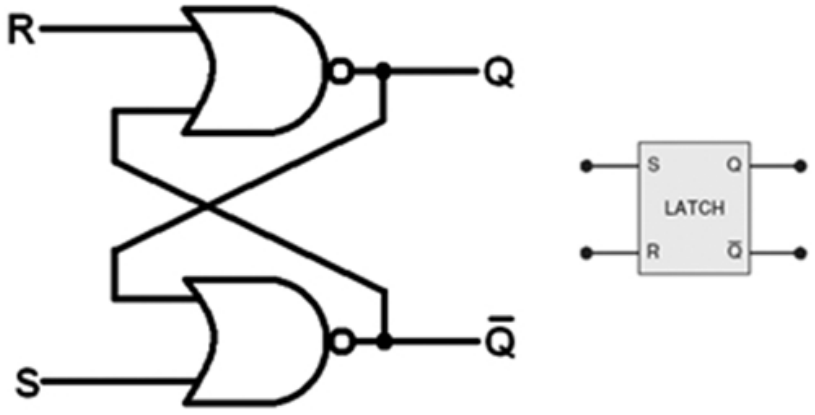


Figura 6.2. Circuito latch SR e representação por bloco (Fonte: o Autor).

O *latch* SR tem duas entradas, nomeadas de R e S, e duas saídas, ditas complementares, Q e Q′. O latch funciona da seguinte forma:

1. Com R=1 e S=0, o circuito realiza a operação de *reset*, de forma que o bit interno armazenado torna-se igual a 0;
2. Com S=1 e R=0, o circuito realiza a operação de *set*, de forma que o bit interno armazenado torna-se igual a 1;
3. Com R=1 e S=1, as duas saídas do circuito são inconsistentes (entradas não empregadas em projetos);
4. Com R=0 e S=0, a saída do circuito dependerá do estado anterior do circuito.

Por exemplo, se quisermos armazenar o bit 1 no circuito, devemos usar as entradas R=0 e S=1, e após as entradas R=0 e S=0, para realizar a operação de armazenamento, que não altera o valor, mantendo o bit armazenado pelo circuito. Esse latch é chamado de RS NOR, por

empregar portas NOR. Há uma versão do *latch* RS com portas NAND, que é chamado latch RS NAND. A Figura 6.3 mostra uma simulação de comportamento de um latch SR ao longo de um intervalo de tempo, ressaltando cinco momentos (T1 a T6). Note a operação de *set* sendo realizada nos instantes T1 e T6, a operação de *reset* nos instantes T3 e T5 e a operação de armazenamento nos instantes T2 e T4.

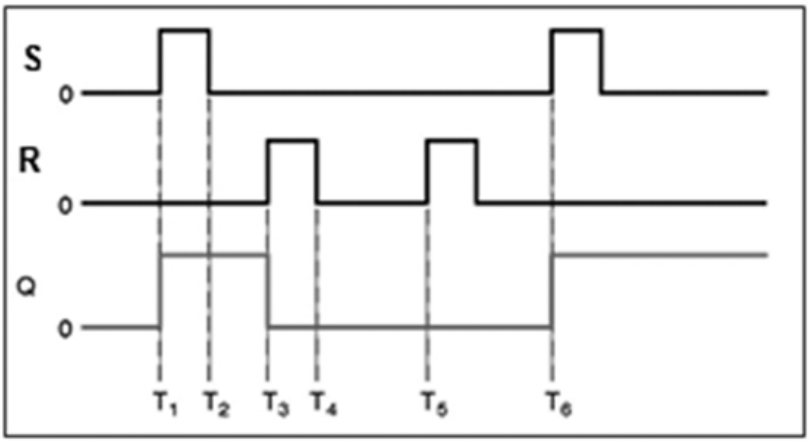


Figura 6.3. Simulação de um circuito latch SR.

Latch D

O **latch D** é uma melhoria do *latch* SR. O circuito do *latch* D, ilustrado na Figura 6.4, é similar ao latch SR, com a diferença que ao invés de duas entradas R e S, o *latch* D tem as entradas de Clock (CLK) e a entrada D.

O *latch* D transfere o valor da entrada de dados D para a saída Q quando a entrada CLK for igual a 1 e mantém o mesmo estado na saída (independente da entrada D) se a entrada CLK for igual 0. Note que o circuito usa o latch RS como componente lógico. A entrada D entra em sua forma normal na porta inferior e na forma complementar, de forma que nunca haverá o problema de entrada com valores inconsistentes no latch RS. A linha de clock CLK permite que o latch realize a operação quando o clock estiver em nível alto. Essa forma de construção de circuito sequencial é chamada de latch **sensível ao nível**. O *latch* D com *clock* é também chamado de **transparente**, pois as saídas refletem a operação determinada pelas entradas a qualquer momento em que o clock estiver em nível alto (igual a 1).

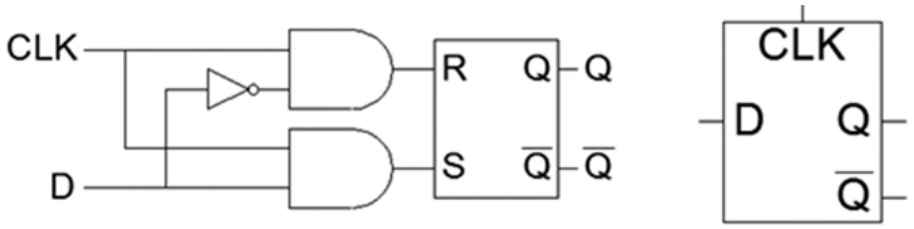


Figura 6.4. Circuito latch D (Fonte: o Autor).

Flip-flop

Embora o *latch* D com *clock* permita que haja a sincronização do controle da memória interna por meio do clock, seu comportamento em relação à sensibilidade do nível de clock pode causar erros no projeto, desde que o estado interno pode ser alterado em qualquer momento que o nível de clock for igual a 1. Os projetistas de computadores muitas vezes empregam *flip-flop* como circuito sequencial / elemento de memória devido a suas características mais adequadas em relação à sincronização dos eventos.

Flip-flops são circuitos sequenciais como latches, que servem para armazenar um bit. A diferença é que o *flip-flop* é dito **sensível à borda** do sinal de clock. Isso significa que alterações no estado interno do *flip-flop* só são possíveis no momento de transição do sinal de clock. Quando o *flip-flop* é sensível à transição de 0 para 1 do sinal de *clock*, diz-se que o flip-flop é **sensível à borda de subida** do sinal de clock. Quando o *flip-flop* é sensível à transição de 1 para 0 do sinal do *clock*, diz-se que o flip-flop é **sensível à borda de descida** do sinal de clock.

A escolha entre borda de subida ou descida depende de detalhes do projeto e da tecnologia adotada. Com essas características, o *flip-flop* é classificado como um tipo de **latch não transparente**, controlado pelas transições do sinal de clock. Isso significa que o valor atual tal saída não está relacionado com o valor atual da entrada.

Flip-flops são construídos empregando latches. Um *flip-flop* é formado por dois *latches*, o primeiro da sequência e mais próximo às entradas chamado de **mestre** e o segundo, mais próximo das saídas, chamado de escravo**.** Considerando um *flip-flop* sensível à borda de

subida, o latch mestre é responsável por *aceitar* a entrada apresentada ao circuito quando o clock está em nível baixo; o latch escravo é responsável por *armazenar* a entrada apresentada ao circuito quando o clock estiver em nível alto. Dessa forma, na transição do clock o valor é repassado do latch mestre para o escravo. A Figura 6.5 ilustra o circuito lógico de um flip-flop D. Note que os dois latches empregam o mesmo *clock* e que para o mesmo instante de tempo, os latches mestre e escravo recebem sinais complementares de *clock*.

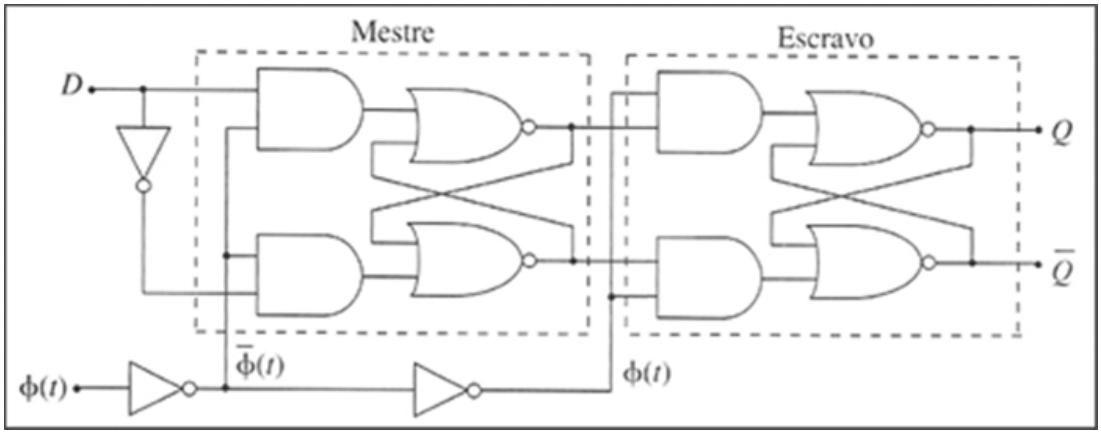


Figura 6.5. Circuito lógico do Flip-flop D (Fonte: Uyemura, 2002).

Flip-flop JK

Uma variante do *flip-flop* tradicional é o **flip-flop JK**. O *flip-flop* JK aprimora o comportamento de um flip-flop RS interpretando de forma diferente a entrada inconsistente S=R=1. O flip-flop aproveita essa combinação de entradas para realizar a operação de *inversão* do estado interno. Assim, além das operações de *set*, *reset* e armazenagem, o flip-flop JK permite que o projetista inverta o estado interno do elemento de memória.

A combinação J = 1, K = 0 corresponde à operação de *set*; a combinação J = 0, K = 1 corresponde a operação de *reset;* e a combinação J = K = 1 é um comando de inversão do estado interno. O *flip-flop* J-K recebeu este nome em homenagem a Jack Kilby, que inventou o circuito integrado, em 1958, invenção pela qual ele recebeu o prêmio Nobel em Física (2000). A Figura 6.6 ilustra o circuito lógico para o *flip-flop* JK. Nesse circuito, dois *latches* SR são empregados. Note que esse circuito é sensível à borda de descida.

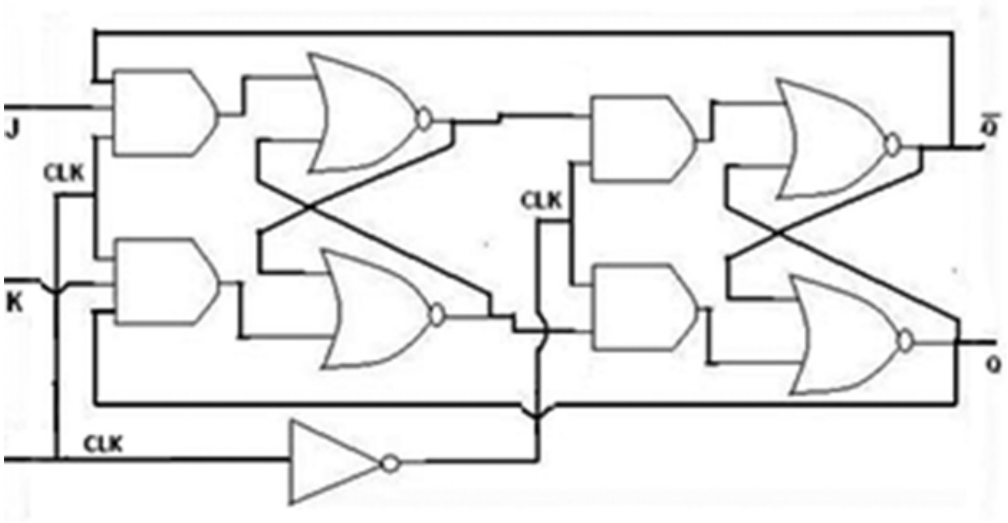
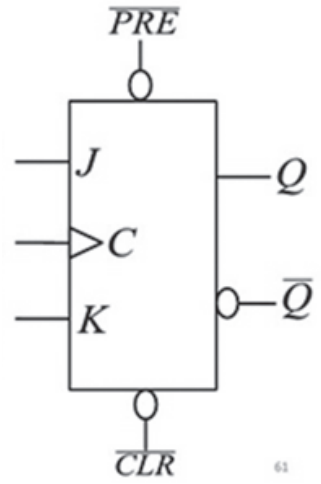


Figura 6.6. Circuito lógico do Flip-flop JK.

Por fim, é importante saber que na prática, os dispositivos eletrônicos que implementam *flip-flops* têm mais duas entradas, chamadas *preset* and *clear*. Essas entradas são chamadas de assíncronas, pois quando são usadas o sinal de clock (e o comportamento do flip-flop em relação ao sinal de clock) é ignorado. Essas entradas permitem que o flip-flop seja ligado com um estado inicial pré-definido. Se ambas as entradas *preset* e *clear* forem iguais a 1, o flip-flop conforme visto previamente. Com *preset*=0 e *clear*=1, o *flip-flop* tem sua saída setada, independente do Clock e das entradas do *flip-flop*. Com *preset*=1 e *clear*=0, o *flip-flop* tem sua saída igual a zero, também independente do sinal do clock e das entradas. A condição *preset=0* e *clear=0* não é empregada, por gera um estado inconsistente (Q=0 e Q′=0). Perceba que para realizar o *preset*, usa-se o valor 0. Dessa forma, diz que a lógica de ativação do *preset* é invertida. O mesmo ocorre para o *clear*, onde para realizar a tarefa, usa-se o bit 0. A Figura 6.7 ilustra a tabela de ativação e o diagrama de bloco de um *flip-flop* JK com as entradas *preset* and *clear.* Perceba o círculo nessas entradas, para indicar a lógica invertida.



CLR	PR	Q_f
0	0	Não permitido
0	1	0
1	0	1
1	1	Funcionamento normal

Figura 6.7. Circuito lógico do Flip-flop JK com entradas assíncronas *preset* e *clear*.

SEÇÃO 2
REGISTRADORES

Elementos de memória como *flip-flops* são úteis para o projeto de computadores para construir elementos de memória mais complexos, chamados de registradores. Registradores são componentes lógicos compostos de flip-flops que permitem armazenar palavras binárias. Computadores atuais implementam um conjunto de registradores de 32 ou 64 bits de tamanho, que servem como área de rascunho para as operações lógicas, aritméticas e de movimentação de dados realizadas pelo computador. Registradores residem internamente ao circuito integrado do processador. A Figura 6.8 ilustra um registrador de oito bits, criado com a utilização de oito flip-flops do tipo D.

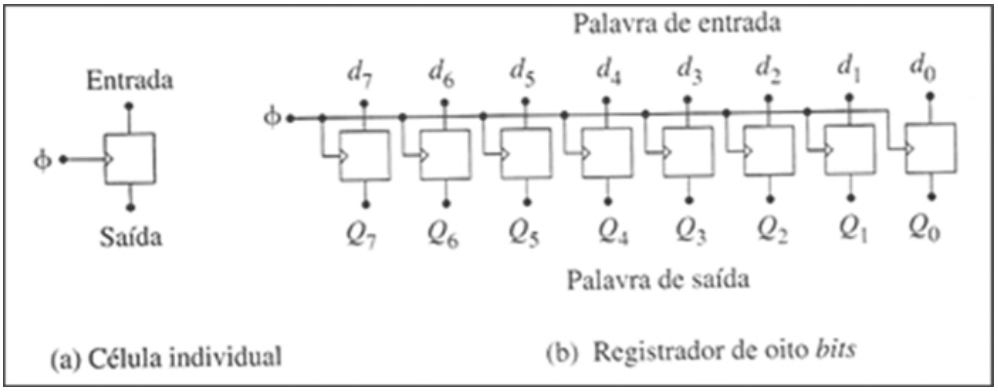


Figura 6.8. Registrador de oito bits (Fonte: Uyemura, 2002).

Registradores podem ser associados de forma a construir um **registrador de deslocamento**. Ao invés de uma entrada de uma palavra completa em um pulso de clock (por exemplo, na borda de subida), o registrador de deslocamento recebe 1 bit da palavra a cada pulso do clock. Os *flip-flops* associados em um registrador de deslocamento constituem um sistema síncrono, pois cada *flip-flop* é controlado pelo mesmo sinal de *clock*. A ligação de cada *flip-flop* com o próximo é feita de forma que a cada transição do *clock*, o conteúdo do *flip-flop* mais à esquerda é transferido para o *flip-flop* mais à direita.

A Figura 6.9 ilustra um registrador de deslocamento de 4 bits, onde são empregados 4 *flip-flops* do tipo JK. Perceba a ligação em série entre os *flip-flops* (no registrador da Figura 6.9 diz que o carregamento é paralelo, ou seja, a palavra é carregada em uma transição de clock apenas). O bit de entrada é aplicado na entrada **E** e a cada transição (borda de subida) do clock, o bit é transferido da esquerda para a direita entre os *flip-flops*.

Registradores de deslocamento são úteis para implementar conversores de dados serial/paralelo e para realizar operações aritméticas de multiplicação e divisão de números inteiros pelo computador. Perceba que se temos uma palavra de 4 bits e o valor armazenado igual a 0010, se deslocarmos tal palavra 1 bit a direita, temos o valor 0001 (o último bit mais a direita da palavra original é desprezado. Perceba que passamos, do valor 0010, ou 2 em decimal, para 0001, valor 1 em decimal.

A operação realizada pelo deslocamento à direita foi a divisão inteira por 2. De maneira geral, à medida que deslocamos uma palavra binária **n** bits à direita, estamos fazendo a operação de divisão da palavra

binária por **2^n.** A relação é parecida em relação à multiplicação. Para isso, considere o valor 0010, armazenado em binário. Se deslocarmos esse valor 2 posições à esquerda, temos 1000. Passamos do valor 0010, ou 2 em decimal, para o valor 1000, ou 8 em decimal. Isso é equivalente a $2x2^2 = 2x4=8$.

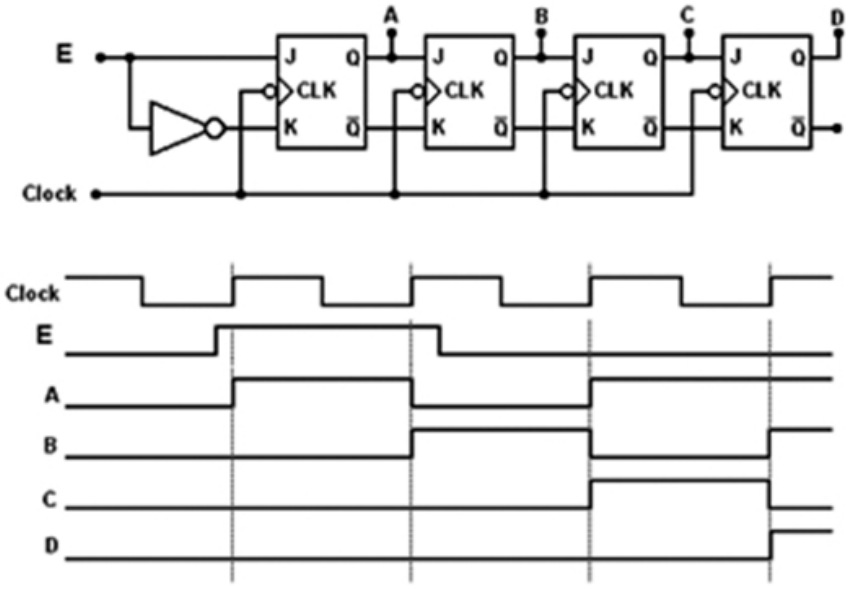


Figura 6.9. Registrador de deslocamento de 4 bits.

Registradores podem ser criados para executar as operações de deslocamento à esquerda e à direita, controlados por entradas (SHR e SHL, do inglês *shift right* e *shift left*). A Figura 6.10 o funcionamento de um registrador de deslocamento, que pode operar ora deslocamento à esquerda, ora deslocando à direita, conforme os valores de SHR e SHL. Além da divisão e multiplicação de valores em série, circuitos internos da unidade central de processamento realizam as operações de divisão e multiplicação de outros valores (além da potência 2) por meio da associação de circuitos somadores e de registradores de deslocamento.

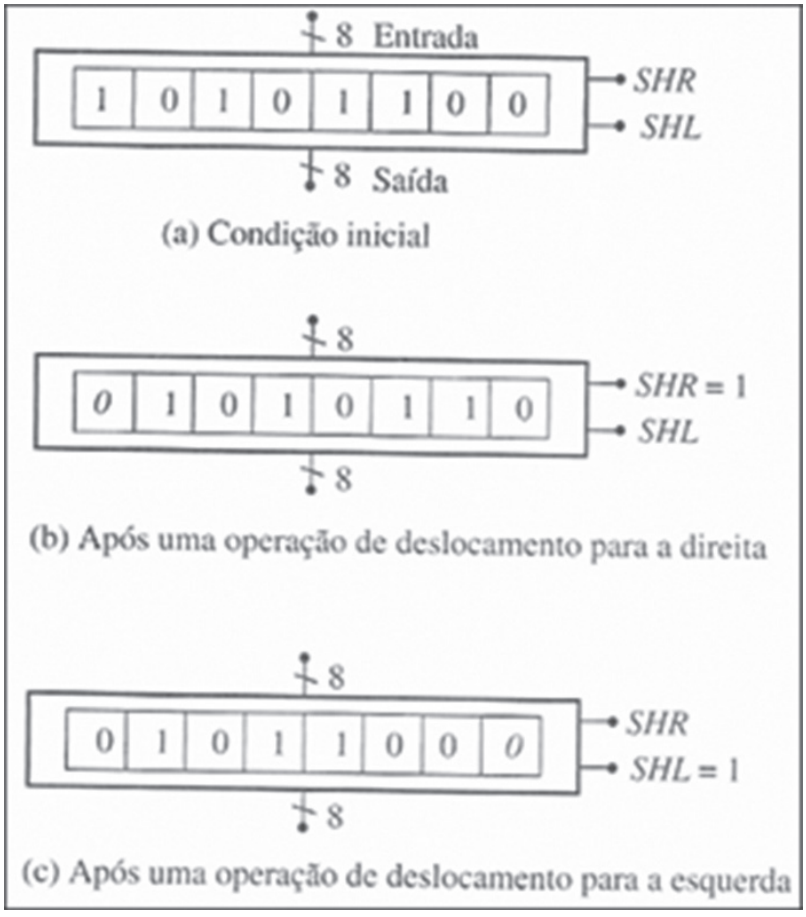


Figura 6.10. Registrador de deslocamento de 8 bits com entradas SHR e SHL.

Visite o site Play Hookey onde você pode simular circuitos sequenciais: http://bit.do/iocuni6.

ATIVIDADES

Após ter estudado o conteúdo desta unidade, é hora de você realizar alguns exercícios para fixação.

1.Considerando um latch NOR com o estado inicial igual a Q=0, verifique o valor final do circuito para os valores de entrada abaixo, aplicados em sequência:

- **S=0; R=0**
- **S=1; R=0**
- **S=0; R=0**
- **S=0; R=1**
- **S=0; R=0**

2. Desenhe as saídas considerando o diagrama abaixo (Figura 6.11), preencha as saídas do circuito, considerando as entradas CLK(Clock), J e K. Considere que o estado inicial do flip-flop é igual a 1.

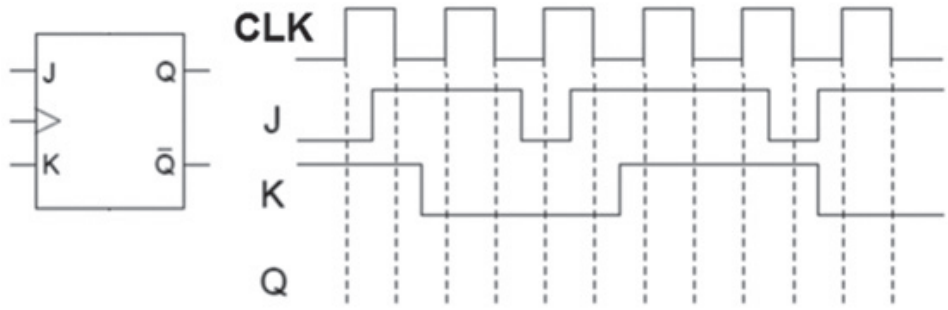


Figura 6.11. Exercício sobre flip-flop JK.


ANOTAÇÕES

UNIDADE VII

Tecnologias de memória

Objetivos de aprendizagem

- Conhecer como o computador organiza os elementos de memória por meio de uma hierarquia.
- Conhecer como são organizados os registradores e o banco de registradores da unidade central de processamento.
- Conhecer como memórias de acesso aleatório são empregadas internamente a unidade central de processamento.
- Conhecer a tecnologia adotada na memória principal do computador. Entender o conceito de volatilidade da memória.

Roteiro de estudos

- SEÇÃO 1 – Hierarquia de memória
- SEÇÃO 2 – Registradores e banco de registradores
- SEÇÃO 3 – Memórias de acesso aleatório

UAB UNIVERSIDADE ABERTA DO BRASIL

PARA INÍCIO DE CONVERSA

Prezada(o) aluna(o)

Nesta unidade, você terá a oportunidade de conhecer as principais tecnologias de memória utilizadas na organização do computador.

Bom estudo!

SEÇÃO 1
HIERARQUIA DE MEMÓRIA

Para realizar suas tarefas, o computador emprega uma quantidade diversa de tecnologias de memória. É comum representar os tipos de memória empregados pelo computador como uma pirâmide, ou hierarquia de memória. Em uma relação que expressa a quantidade de memória x velocidade, nessa pirâmide o eixo das abcissas representa a quantidade de dados que podem ser armazenados e o eixo das ordenadas o desempenho (quanto maior, melhor). Em direção ao topo, a memória torna-se mais rápida (tempo de acesso menor), tem custo mais elevado por Byte armazenado e menor capacidade de armazenamento. Em relação à base, a memória torna-se mais lenta (maior tempo de acesso), tem custo menor por Byte armazenado e maior capacidade de armazenamento.

Internamente a unidade central de processamento, o computador utiliza em seus **registradores** e na **memória cache**. Tanto os registradores quanto a memória cache armazenam dados e instruções de máquina que servem para o processamento dos dados pelo computador. A tecnologia de memória SRAM (*Static random access memory*) é empregada nos registradores e na memória cache.

Externamente a unidade central de processamento, na memória principal o computador emprega a tecnologia de memória DRAM (*Dynamic Random Access Memory*). Tais tecnologias diferem em relação ao desempenho (velocidade), custo e consumo de energia. As tecnologias adotadas nesses níveis são classificadas de **voláteis**, pois as informações (dados) só estão disponíveis enquanto há alimentação de energia, ou seja, enquanto o computador está ligado. Tanto SRAM como DRAM são memórias de acesso aleatório e serão discutidas na seção 2.

É importante salientar que a memória cache, interna a UCP, em computadores mais antigos era implementada em um circuito integrado externo. Com a integração em uma escala muito grande, tornou-se possível construir memória cache interna a UCP. Da mesma forma, a indústria incorporou na memória cache vários níveis, atualmente chamados de L1 a L3, cada nível com parâmetros diferentes de desempenho e de capacidade de armazenamento.

Também externamente a unidade central de processamento tem a memória secundária, composta de dispositivos de armazenamento magnéticos (discos magnéticos e rígidos, unidades de fita magnética), óticos (unidades de Blu-ray, DVD e CD) e eletrônicos (drives de armazenamento de estado sólido). As tecnologias adotadas na memória secundária são **não voláteis,** de forma que a informação permanece armazenada mesmo quando o computador é desligado.

A Tabela 7.1 apresenta os tempos de acesso e capacidade típicos de armazenamento nos diferentes níveis de memória, da hierarquia de um computador contemporâneo. As Tabelas 7.2 e 7.3 apresentam respectivamente as ordens de magnitude de tempo e unidades de armazenamento em sistemas de computação, como referência.

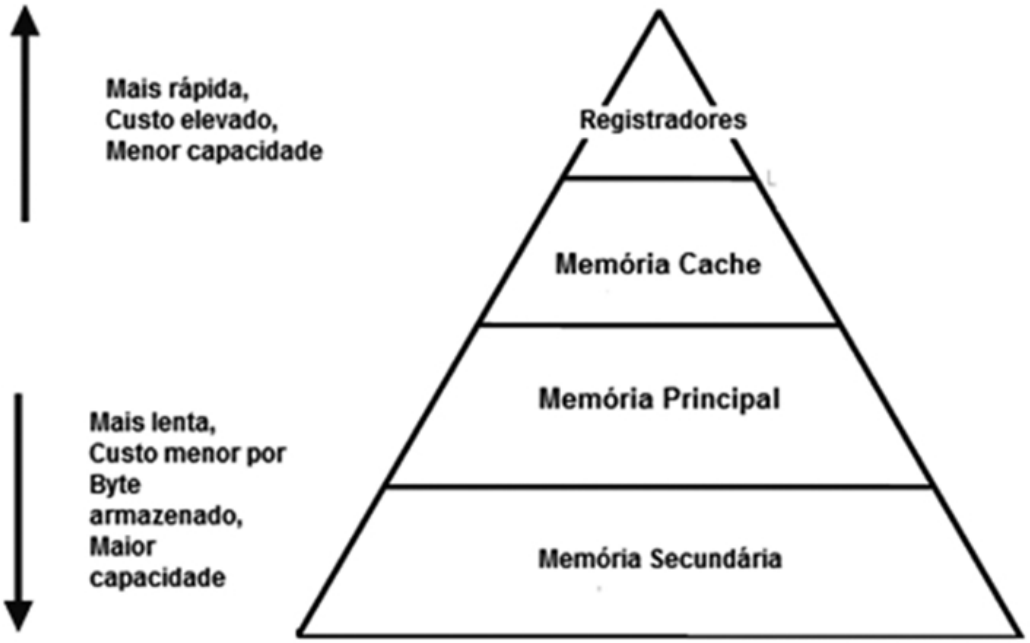


Figura 7.1. Hierarquia de memória simplificada.

Tabela 7.1. Tempos de acesso e capacidade de armazenamento, típicos dos computadores contemporâneos.

Nível da Hierarquia	Tempo de acesso	Capacidade de armazenamento
Registradores	300 ps	1KByte
Memória Cache	3 ns	4 MBytes
Memória Principal	10 ns	8GBytes
Memória Secundária	10 ms (disco magnético) / Memória flash (25 us)	16TBytes/ 240 GBytes

Tabela 7.2 Ordens de Magnitude do Tempo,

Fator (segundos)	Múltiplo	Símbolo
10^{-12}	Picossegundo	ps
10^{-9}	Nanossegundo	ns
10^{-6}	Microssegundo	μs
10^{-3}	Milissegundo	ms
10^{0}	Segundo	S

Tabela 7.3 Unidades de medida para capacidade de armazenamento,

Nome	Abreviação	Fator/Tamanho
bit	B	1
nibble	nibble	4 bits
Byte	B	8 bits
KByte (Kilo)	KB	2^{10} Bytes
MByte (Mega)	MB	2^{20} Bytes
GByte (Giga)	GB	2^{30} Bytes
TByte (Tera)	TB	2^{40} Bytes
PByte (Peta)	PB	2^{50} Bytes

SEÇÃO 2
REGISTRADORES E BANCO DE REGISTRADORES

Na unidade anterior (VI), vimos como um conjunto de flip-flops podem ser organizados para criar um registrador. Um registrador é uma memória de acesso rápido, localizada na pastilha do processador do computador (UCP), composto de **n** bits. Valores comuns de **n** são 16, 32 e 64 bits. Para que o computador possa executar as instruções de máquina, se faz necessário a existência de mais de um registrador.

Por exemplo, a arquitetura MIPS (Patterson & Henessy, 2012) emprega 32 registradores, de 32 bits de tamanho cada, no processamento das instruções. Tais registradores são organizados em uma estrutura chamada **banco de registradores**. Banco de registradores é então uma memória, composta de um conjunto reduzido (dezenas) de registradores que são acessados por linhas especiais de endereçamento. Através das linhas de endereçamento a unidade de controle do processador por especificar qual registrador será usado. Por exemplo, um computador como o MIPS quando recebe as seguintes instruções de máquina (especificadas em linguagem de montagem):

```
Lw $t0, a
Lw $t1, b
Add $t2, $t0, $t1
Sw $t2, c
```

Um programa como esse seria gerado pelo compilador para uma linha de programa (na linguagem C ou Java) como: **c = a +b;** que calcula a soma das variáveis **a** e **b** e armazena o resultado na variável **c**. Para isso, o programa utiliza os registradores $t0, $t1 e $t2, que são mnemônicos para endereços de registradores do banco de registradores MIPS, após trazer os dados das variáveis a e b da memória com as instruções **lw (***load word***)**. Logo, para cada instrução, sinais de endereçamento (bits)

são enviados para o banco de registradores, via o barramento interno do processador, pela unidade de controle. Esses sinais permitem que os valores oriundos das variáveis em memória a e b sejam armazenados no banco de registradores, nos registradores $t0 e $t1.

Um banco de registradores pode ser implementado através de um decodificador para cada porta de leitura e escrita e uma matriz *(array)* de registradores, construídos a partir de *flip-flops*, por exemplo, do tipo D. Como a leitura do registrador não altera o estado do elemento de memória, só precisamos fornecer o número do registrador como entrada e a saída será os dados contidos no registrador acessado. Usualmente, para escrever em um registrador, são necessárias três entradas: a) número (endereço) do registrador; b) dados a serem escritos no registrador; c) sinal de *clock* para sincronizar a escrita (Uyemura, 2002).

A Figura 7.2 apresenta diagrama de bloco de um banco de registradores. Nesse exemplo, o banco de registradores aceita a leitura de dois registradores ao mesmo tempo, que é útil para o processador computar uma instrução como add $t2, $t0, $t1, que tem dois operandos fonte.

Figura 7.2. Diagrama de bloco de um banco de registradores (Fonte: Patterson & Henessy, 2012).

As Figura 7.3 e 7.4 apresentam o detalhes da implementação de um banco de registradores. Note o uso de decodificadores e de multiplexadores para ter acesso ao registrador especificado pelas linhas de endereço.

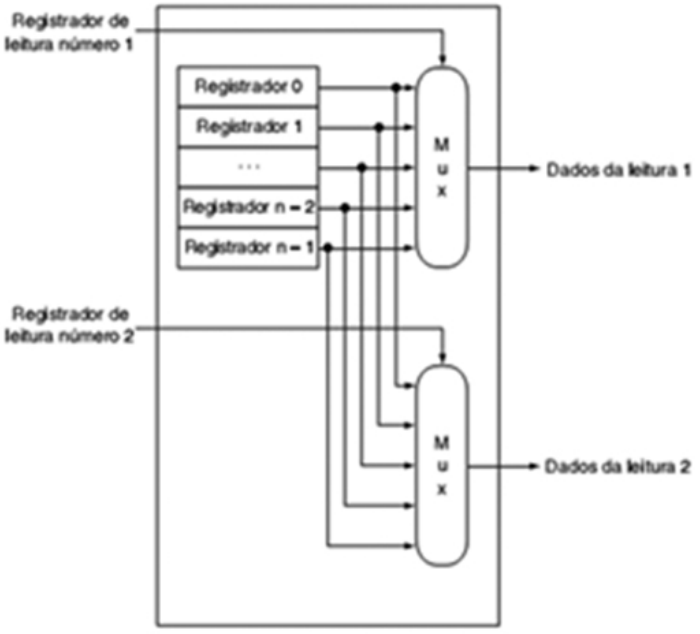


Figura 7.3. Implementação de um banco de registradores: portas de leitura (Adaptado de Patterson & Henessy (2012)).

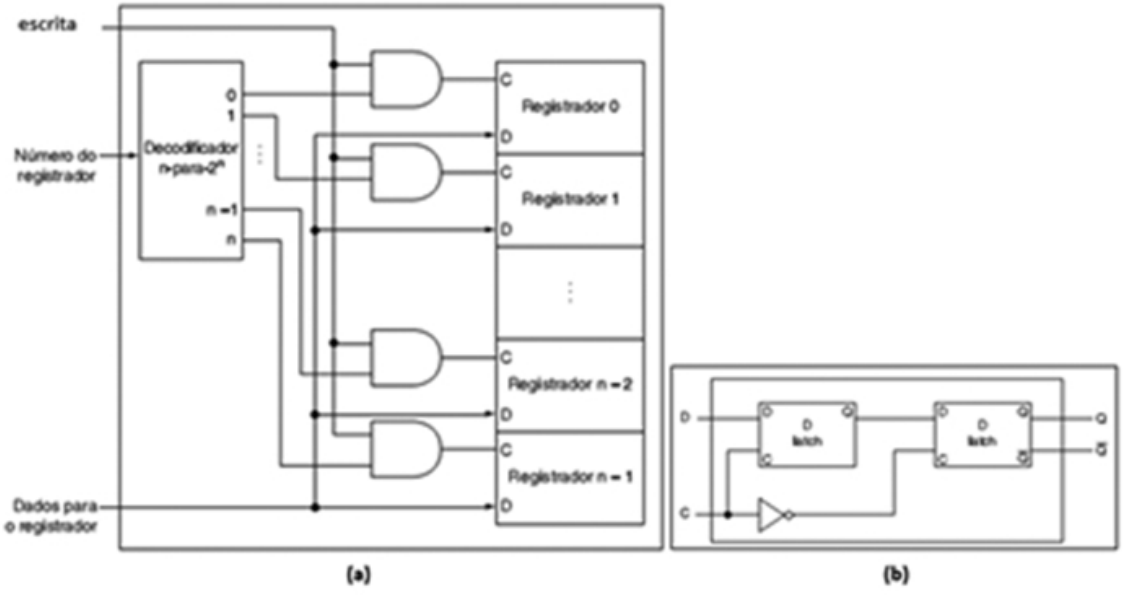


Figura 7.4. Implementação de um banco de registradores: portas de escrita. Em (a) o circuito decodificador é ilustrado. Em (b), a célula básica de cada bit de cada registrador (Adaptado de Patterson & Henessy, 2002).

SEÇÃO 3
MEMÓRIAS DE ACESSO ALEATÓRIO

Registradores e sua organização. por meio de banco de registradores. oferecem blocos de montagem básicos para pequenas memórias, principalmente que residem implementadas dentro do processador (UCP). Além dos registradores, o computador necessita de memória que possam armazenar mais palavras binárias, ou seja, maior capacidade de armazenamento. Tal necessidade levou a concepção e utilização de **memórias de acesso aleatório.**

O termo **memória de acesso aleatório** está relacionado com arranjos de células genéricas de memória que permitem a escrita e leitura, cujo acesso não necessita ser sequencial. Ou seja, por linha de endereço, pode-se acessar diretamente um item na memória (ao contrário de memórias de acesso sequencial, que para acessar o item de posição **n** se faz necessário acessar os itens de **0** até **n-1** obrigatoriamente, como uma fita magnética, por exemplo). Memórias de acesso aleatório são chamadas abreviadamente **RAM** (*Random Access Memory*). Em nosso exemplo, a instrução **lw $t0, a** realiza o acesso à memória principal do computador, organizada como uma memória, para ter acesso à variável **a** e processá-la internamente a UCP por meio do registrador **$t0**.

Assim, memórias do tipo RAM são frequentemente encontradas na memória principal de um computador pessoal e armazena dados necessários para o funcionamento do computador, incluindo os programas e o sistema operacional.

Memórias de acesso aleatório são classificadas em dois grupos principais: memórias estáticas (SRAM) e memórias dinâmicas. **Memórias estáticas** identificam uma classe de memória de acesso aleatório que mantém os dados armazenados desde que seja mantida sua alimentação. São formadas com circuitos digitais biestáveis, assim como os *latches*. Similares ao latch com porta NOR, memórias estáticas empregam estruturas semelhantes, com a diferença de utilizar portas NOT de uma única entrada lógica (Figura 7.5), com transistores atuando como chaves (interruptores) para acesso à célula.

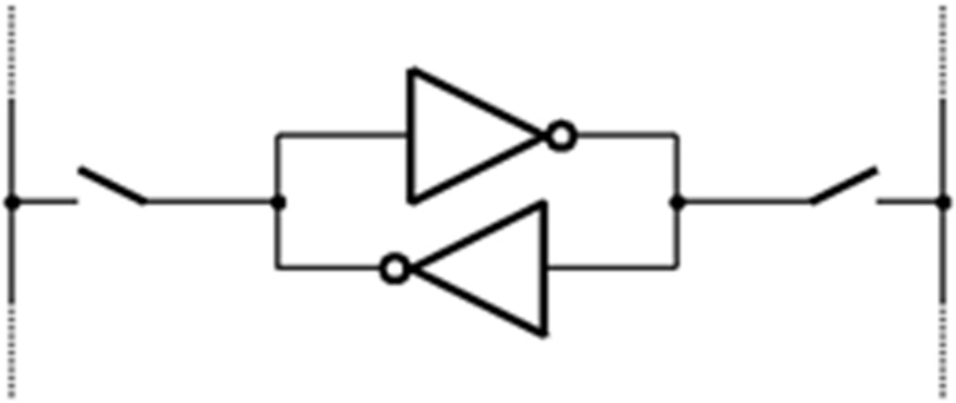


Figura 7.5. Célula básica de uma memória SRAM.

Assim, em uma memória estática o valor armazenado na célula é mantido em um par de portas inversoras (4 a 6 transistores) e, desde que haja energia elétrica sendo aplicada, o valor pode ser mantido (Uyemura, 2002). Nas **memórias de acesso aleatório dinâmicas (DRAM)**, ao invés da utilização de elementos biestáveis, o valor armazenado em uma célula é mantido como a carga de um capacitor. Nas memórias dinâmicas, um único transistor é utilizado para acessar a célula (leitura da carga armazenada). Como empregam apenas um transistor por célula, memórias DRAM são mais baratas e apresentam uma melhor densidade de informação por área no circuito integrado (Uyemura, 2002).

A carga elétrica armazenada no capacitor não pode ser mantida indefinidamente. Assim, torna-se necessário que o circuito execute um *refresh* periódico para garantir a integridade da informação armazenada. A necessidade do *refresh* é a característica principal desse tipo de memória. A carga pode ser mantida por vários milissegundos antes da necessidade de um *refresh.* Memórias dinâmicas são frequentemente encontradas nos computadores em módulos de memória, vários chips (usualmente 8) por módulo de memória. Cada chip contribui com 8 bits, para formar uma palavra de 64 bits por acesso (Figura 7.6).

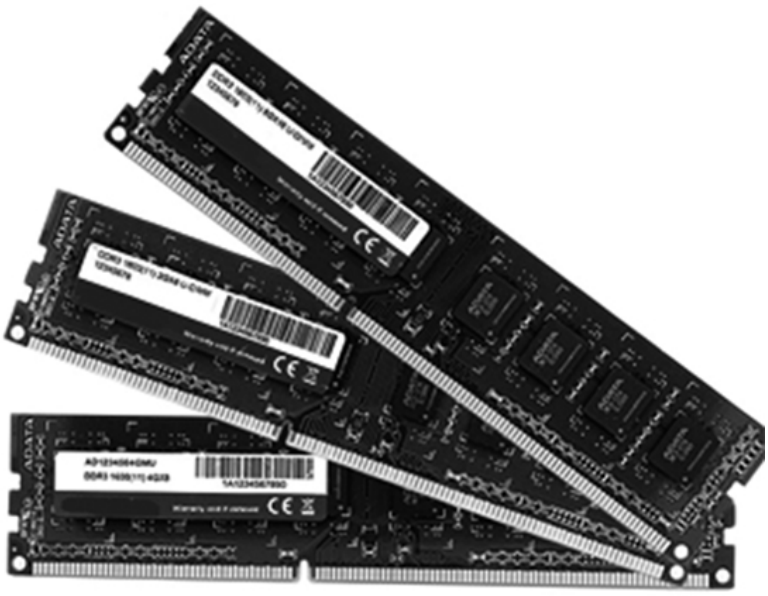


Figura 7.6. Módulos de memória RAM dinâmicas.

As tecnologias de memória RAM permitem a leitura e o armazenamento dos dados (escrita). Comumente, sistemas de computação empregam outro tipo de memória auxiliar, chamada de ROM (*Read Only Memory*). Por definição, esse tipo de memória não volátil permite o armazenamento da informação, também chamada de gravação, apenas uma vez, durante o processo de fabricação. Apesar disso, atualmente a sigla ROM tem sido adotada para nomear várias tecnologias, que permitem inclusive a regravação dos dados. Dessa forma, a interpretação atual da sigla ROM está mais relacionada com a **não volatilidade** da memória. Abaixo, segue uma listagem dos tipos mais comuns de ROM:

- PROMs (*Programmable Read-Only Memory*): memórias que podem ser escritas com dispositivos especiais, mas não podem mais ser apagadas ou modificadas.
- EPROMs (*Erasable Programmable Read-Only Memory*): tecnologia de memória cujo conteúdo pode ser apagado pela exposição à radiação ultravioleta e pode ser reutilizada.
- EEPROMs (*Electrically Erasable Programmable Read-Only Memory*) podem ter seu conteúdo modificado eletricamente, mesmo quando já estiver funcionando num circuito eletrônico.
- CD-ROM e DVD-ROM, que são discos de armazenamento óticos.

Outra tecnologia da família de memórias não voláteis que tem se destacado para armazenamento de dados é a **memória flash. A memória flash** é uma evolução das memórias do tipo EEPROM. A memória flash tem sido adotada em dispositivos de armazenamento de estado sólido, abreviados como SSD, que são substitutos para os discos rígidos e magnéticos de armazenamento. A Figura 7.7 ilustra dois dispositivos de armazenamento: um disco rígido e um drive SDD. A figura 7.8 ilustra um pendrive, dispositivo construído com memória do tipo *flash.*

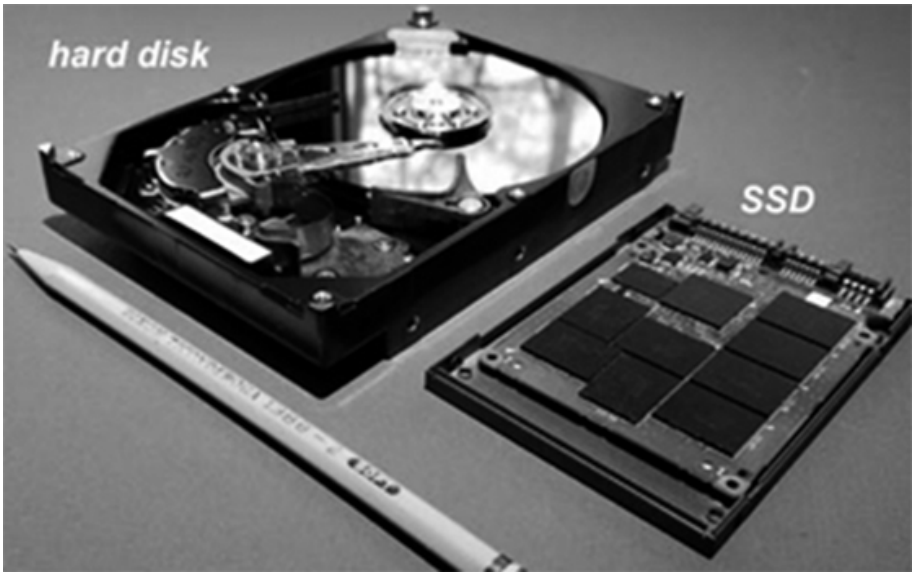


Figura 7.7. Disco magnético e rígido, ao lado de um drive de armazenamento SSD (Fonte: revista PCMAG).

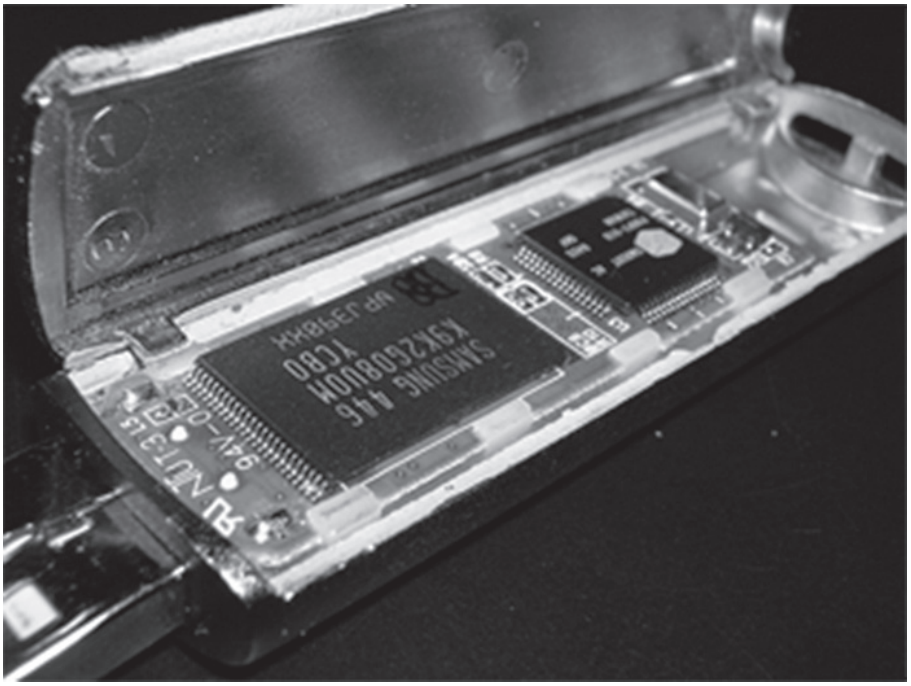


Figura 7.8. Pen drive aberto e circuito integrado de memória *flash* (Fonte: Wikipédia).

ARTICULANDO

Leve para sala de aula módulos de memória, discos magnéticos e drives para que o aluno perceba na prática como são fisicamente tais componentes. Faça uma atividade com o código do circuito integrado principal dos componentes na Internet para que o aluno saiba os parâmetros das tecnologias empregadas.

SAIBA MAIS

Visite o site Tutorial sobre DRAM e SRAM: http://bit.do/qrcode/iocuni7.


ANOTAÇÕES

PALAVRAS FINAIS

Parabéns!

Você acaba de completar a disciplina Introdução à Organização de Computadores. Com o conhecimento que você adquiriu, por meio desse estudo, você está apto a entender os demais conceitos de organização e da próxima disciplina da área de arquitetura de computadores.

Este livro é seu e agora que você já terminou a disciplina, guarde-o com cuidado. Consulte este livro quando aparecerem dúvidas nas próximas disciplinas, pois as respostas podem estar por aqui.

REFERÊNCIAS

STALLINGS, William. **Computer organization and architecture:** designing for performance. 4. ed. New Jersey: Prentice Hall, 1996.

MURDOCCA, M. **Introdução à arquitetura de computadores**. Elsevier, 2001.

TANENBAUM, Andrew S.; AUSTIN, T. **Organização estruturada de computadores.** 6. ed. São Paulo: Pearson Prentice Hall, 2013.

HENNESSY, J.L; PATTERSON, D. A. **Organização e projeto de computadores - A Interface Hardware Software**. 4. ed. Campus, 2014.

PEDRONI, A.V. **Eletrônica Digital Moderna e VHDL.** Elsevier, 2010.

HARRIS, D.; HARRIS, S. **Digital Design and Computer Architecture.** 2. ed. Morgan Kaufmann, 2012.

PATTERSON, D. A.; J. L. HENESSY. **Organização e projeto de computadores** - A Interface Hardware Software. 4. ed. Campus, 2012.

STALLINGS, W. **Arquitetura e organização de computadores.** 8. ed. Pearson, 2010.

NOTAS SOBRE O AUTOR

Luciano José Senger

Luciano José Senger é doutor em Ciências da Computação e Matemática Computacional pela Universidade de São Paulo (ICMC/USP). Professor associado na Universidade Estadual de Ponta Grossa (UEPG), onde atua desde 1998, tem trabalhado nas disciplinas nas áreas de arquitetura de computadores e computação de alto desempenho para o Curso de Engenharia de Computação. Professor permanente do Mestrado em Computação Aplicada da UEPG, desde 2010, atuando na área de pesquisa de computação de alto desempenho.

Ministério
da Educação

UEPG
UNIVERSIDADE ESTADUAL DE PONTA GROSSA

nute@d
Núcleo de Tecnologia e Educação Aberta e a Distância