

Organização de Computadores

- [Início](#)
- [Tags do conteúdo](#)
- [Referências](#)
- Imprimir

Organização de Computadores



Organização de Computadores

É bastante comum se fazer uma distinção entre arquitetura de computadores e Organização de Computadores (STALLINGS, 2017). A Arquitetura de Computador trata de aspectos e atributos que possuem um impacto direto sobre a execução lógica de um programa, ou seja, as partes de um Sistema (sua codificação) visíveis a um programador.

Um exemplo de uma questão de arquitetura é definir se um computador terá uma instrução de multiplicação. Por outro lado, do ponto de vista da Organização de Computadores define-se por exemplo se essa instrução será implementada por uma unidade de multiplicação especial ou por uma estrutura que faça uso repetido da unidade de adição do Sistema.

Segundo Stallings (2017) “Historicamente, e ainda hoje, a distinção entre Arquitetura e Organização tem sido importante. Muitos fabricantes de computador oferecem uma família de modelos de computador, todos com a mesma arquitetura, mas com diferenças na organização. Consequentemente, os diferentes modelos na família têm diferentes características de preço e desempenho. Além do mais, uma arquitetura em particular pode se espalhar por muitos anos e abranger diversos modelos diferentes de computador, com sua organização variando conforme a mudança da tecnologia”.

Este curso aborda a Organização e a Arquitetura do Computadores. Sendo que a ênfase é maior para a Organização. Entretanto, a organização de computador precisa ser projetada para implementar determinada especificação de arquitetura, portanto um tratamento completo da organização exige um exame detalhado também da arquitetura.

Pode-se perceber então que a Arquitetura de Computadores trata do comportamento funcional de um sistema computacional. Já a Organização de Computadores trata da estrutura interna que não é visível para o programador (ex. frequência do relógio “clock” ou tamanho da memória física).

A importância em se estudar a arquitetura e organização de computadores é enfatizada na célebre publicação **Computer Engineering 2004 Curriculum Guidelines** (apud STALLINGS, 2010), nela a arquitetura de computador é descrita como componente chave da engenharia da computação o que demonstra a relevância, e necessidade de o engenheiro da computação ter um conhecimento teórico e prático desse assunto. Visto que a Organização e Arquitetura de computadores oferecem tratativa a todos os aspectos de um projeto de organização e integração da CPU (*Computer Processing Unit*) ou UCP (Unidade Central de Processamento) no sistema de computação.

A arquitetura de um computador influencia diretamente na *software* do computador, uma vez que a arquitetura de um processador coopera de modo direto com o Sistema Operacional (SO) e o *software* do Sistema (programas de apoio como Editores de Texto, Navegadores, Planilhas, Bancos de Dados, etc.). Além de que o projetista do computador precisa ter conhecimento sobre o *software* a fim de implementar a arquitetura ideal. A arquitetura de computador dentre outros objetivos irá: abordar princípios básicos, oferecer uma visão geral da arquitetura e ensinar aos alunos a operação de um computador. Vai ainda permitir que estudantes da área entendam como diversos dispositivos periféricos interagem e são ligados a uma CPU (STALLINGS, 2010).

História da Computação Moderna

A figura 1 a seguir mostra de forma resumida a evolução histórica dos computadores, desde *Alan Turing* até a computação quântica, destacando os principais marcos e inovações da área.

Anos 30: *Alan Turing* cria a Máquina de Turing para “entender” computadores.

Anos 40: ENIAC inicia computadores eletrônicos. *Konrad Zuse* faz o Z3 programável.

Anos 50: Surgem [FORTRAN](#)¹ e UNIVAC I, computador vendido.

Anos 60: IBM cria o System/360. Surgem novas linguagens (mais alto nível e OO).

Anos 70: Altair 8800 - PC inicial. Xerox PARC cria a tela gráfica e o mouse.

Anos 80: Apple populariza telas gráficas. IBM define PCs.

Anos 90: Internet nasce com Web. PCs rápidos.

Anos 2000: Internet, redes sociais e IA crescem.

Anos 2010: IA e Big Data importantes.

Anos 2020: Computadores quânticos avançam, VR/AR e IoT mostram futuro.

História da Computação Moderna

1930

ALAN TURING E A MÁQUINA DE TURING

Alan Turing desenvolve a ideia teórica da "Máquina de Turing", um dispositivo lógico que se torna fundamental para o entendimento dos princípios da computação

**1940**

ENIAC E PRIMEIROS COMPUTADORES

- O ENIAC (Electronic Numerical Integrator and Computer) é construído nos EUA, marcando o início dos computadores eletrônicos.
- Konrad Zuse desenvolve o Z3, o primeiro computador programável totalmente automático.

**1950**

COMPUTADORES DE SEGUNDA GERAÇÃO

- Surge a ideia de programação de alto nível, resultando na criação da linguagem FORTRAN.
- UNIVAC I, o primeiro computador comercial, é vendido para empresas.

**1960**

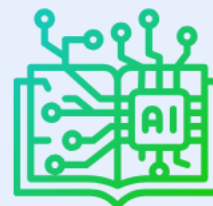
COMPUTADORES DE TERCEIRA GERAÇÃO

- IBM lança o IBM System/360, um computador que permite a compatibilidade entre diferentes modelos.
- Desenvolvimento das primeiras linguagens de programação Orientadas a Objetos (OO).

**1970**

COMPUTADORES PESSOAIS

- A revolução dos computadores pessoais começa com a introdução do Altair 8800.
- A Xerox PARC desenvolve a Interface Gráfica do Usuário (GUI) e o mouse.



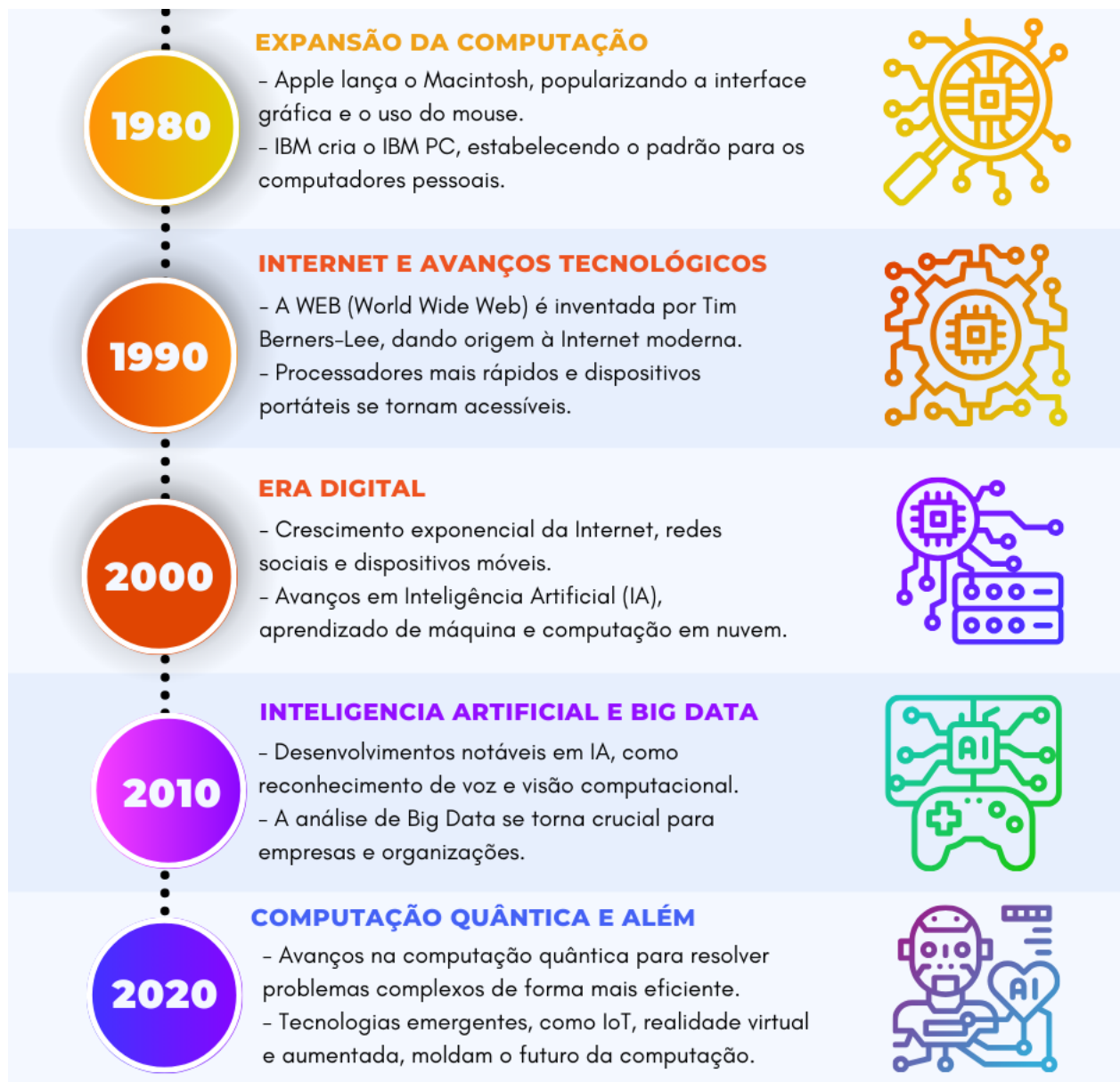


Figura 1: Infográfico - Evolução Histórica dos Computadores

Desde a antiguidade, a necessidade do homem de possuir meios para calcular e computar seus bens revela-se algo inerente a sua própria natureza. Sendo que, em suas fases iniciais o ser humano utilizava seus próprios dedos (Dez = Decimal) como um meio de efetuar contagens (MENNINGER, 2013). Consequentemente esta forma de contagem influenciou diretamente a formação e adoção do sistema de numeração decimal por grande parte das sociedades, em especial a sociedade ocidental (IFRAH, 2000).

Fato Curioso

Cálculo vem do latim “*Cálculus*” que era empregado para designar pequenas pedras, as quais eram utilizadas para a contagem. Esta prática rudimentar consistia em manipular tais pedras, representando quantidades específicas, a fim de realizar operações aritméticas básicas. Essa técnica, embora limitada pela sua simplicidade, representou um marco inicial na trajetória do desenvolvimento das ferramentas de cálculo. Uma excelente sugestão para entender a evolução dos números neste contexto é o vídeo “*Evolução dos Números / História Completa!*”, o qual apresenta uma narrativa elucidativa, acompanhada de ilustrações que contextualizam de maneira clara e acessível essa trajetória histórica.

O vídeo completo, está disponível no *YouTube* através do *link* a seguir:

<<https://www.youtube.com/watch?v=pa1sANJD1PE>>

Quando o ser humano deixou seu estilo de vida nômade e fixou habitação deixando de ser coletor e passando a se dedicar à agricultura e ao pastoreio, tornou-se necessário o desenvolvimento de concepções de quantificação e numeração (BRAUDEL, 1995). Por exemplo os pastores contavam suas ovelhas com pedras e se ao final do dia sobrassem pedras, ovelhas haviam sido perdidas.

A prática de associar pedras às ovelhas, que se refletia na contagem das unidades mediante a disposição das pedras, embora simples, estabeleceu os fundamentos para futuros desenvolvimentos numéricos e abriu caminho para a sofisticação dos sistemas de contagem e registro. Entretanto com o crescimento dos rebanhos as pedrinhas se tornam insuficientes. E agora o que fazer?

Com o crescimento dos números, ou seja, das quantidades o ser humano precisou se aprimorar e criar instrumentos de apoio à contagem. E assim surgiram ferramentas que são as ancestrais do computador, tais como: o Ábaco, a Régua de Cálculo e a Máquina de Cálculo – sendo os dois primeiros utilizados ainda hoje.

¹ O **Fortran** foi uma das primeiras linguagens de programação, tendo sido projetada para aplicações computacionalmente intensivas em ciência e engenharia. O seu primeiro compilador foi desenvolvido para o IBM 704 para o qual foi utilizada entre 1954-57.

História da Computação

Bem antes de começar a contar a história da computação, vamos definir os termos Computador e Computação. De modo bem simplificado se diz que Computador é “*aquele que faz cálculos*”, por sua vez o termo computar é um sinônimo de calcular.

Já a Computação é a ciência que estuda e sistematiza as ordens e atividades inseridas numa máquina, ao analisar os fatores que participam desse processo, entre os quais se encontram as linguagens de programação, que permitem criar uma lista de dados ordenada e perceptível para a máquina (EQUIPE EDITORIAL DE CONCEITO.DE, 2020).

Ribeiro *et al.* (2019), traz uma visão mais profunda sobre que é Computação, afirmando que a mesma se trata de uma ciência fundamentada em princípios que organizam de maneira sistemática, uma parcela do conhecimento humano. Estando presente em todos os âmbitos das ações cotidianas e de tudo o que fazemos. Os autores enfatizam adicionalmente, que a Computação se apresenta como uma ferramenta habilitadora para a análise profunda de problemas e a concepção de soluções. Além de oferecer explicações, ela facilita instrumentos capazes de catalisar transformações no mundo, ampliando, assim, seu impacto e alcance.

A história da computação, acelera, significativamente no século passado. Esse avanço adquiriu particular destaque partir da concretização de ideias abstratas, como a Máquina de Turing, criada pelo matemático e criptógrafo inglês **Alan Mathison Turing** na década de 1930, a qual consolidou as bases teóricas para o desenvolvimento dos computadores eletrônicos modernos. Estes conceitos começam a ser colocados em prática especialmente em decorrência e durante a Segunda Guerra Mundial, com o *Projeto Colossus* e o subsequente advento dos primeiros computadores eletrônicos programáveis (CAMPBELL-KELLY *et al.*, 2023).

Dica de vídeo

Para conhecer o *Projeto Colossus* e sua importância, assista ao vídeo “*Colossus: Creating a Giant*” ou “*Colossus: Criando um Gigante*”, o qual apesar de conciso é bastante esclarecedor e está disponível no *YouTube* através do link:

<<https://www.youtube.com/watch?v=knXWMjIA59c>>

Entretanto apenas nas décadas mais recentes, os sistemas computacionais começaram a se popularizar. Principalmente devido a avanços alcançados no *design* e fabricação de microprocessadores que permitiram a criação de dispositivos computacionais cada vez menores, mais eficientes e mais acessíveis. Abrindo assim caminho para a revolução da computação pessoal e a integração de computadores em praticamente todos os aspectos da sociedade moderna.

Sendo assim pode-se perceber que a evolução dos sistemas computacionais, apesar de se iniciar em um passado distante e ser praticamente uma necessidade inerente ao homem desde quando ele deixa de ser nômade-coletor e passa a fixar habitação em uma região, vai florescer plenamente graças à convergência de conhecimentos teóricos e à concretização tecnológica, culminando na atual era digital onde a computação é onipresente e essencial para inúmeras atividades cotidianas.

História da Computação desde seus primórdios em ordem cronológica

~**5500 a.C.** (aproximadamente), ocorreu a invenção do Ábaco, considerado um dos primeiros dispositivos desenvolvidos para auxiliar os seres humanos na realização de cálculos (IFRAH, 2000). O Ábaco, com sua estrutura de contas e movimentação de peças, ofereceu uma abordagem prática para efetuar operações aritméticas.

830 – *Muhammad ibn Mūsā al-Khwārizmī* criou os algoritmos (sequências finitas de passos para a resolução de problemas) empregando os algarismos hindus (RASHED, 1994). Essa contribuição forneceu a base para a manipulação sistemática de números e formulação de procedimentos lógicos.

1202 – *Fibonacci* traduz os livros de *al-Khwārizmī*, fazendo com que ele fosse conhecido no ocidente, ao traduzir seus livros para o latim, o que permitiu a propagação dos conhecimentos de algoritmos e álgebra.

1550-1617 – *John Napier* foi o precursor das réguas de cálculo. Essas réguas eram uma ferramenta engenhosa para simplificar cálculos complexos, transformando-os em operações mais acessíveis.

1623-1662 – *Blaise Pascal* cria a primeira máquina mecânica de calcular, capaz de efetuar adições e subtrações (ADAMSON, 1994; HADAMARD, 1954).

1646-1716 – *Gottfried Leibniz* cria uma máquina capaz de multiplicar e dividir (HADAMARD, 1954). Seu sonho era poder substituir a atividade intelectual humana por operações mecânicas, ou seja, poder substituir todo o raciocínio humano pelo puxar de uma simples alavanca.

1801 – A revolução industrial culminou com a transição do artesanato para o industrial, onde diversas tarefas humanas foram substituídas pelo trabalho das máquinas. Neste cenário *Joseph Jacquard* cria um tear mecânico controlado por cartões perfurados o que vem a representar um marco significativo na automação de tarefas (HYMAN, 1985).

1792-1871 – Os primeiros passos para uma nova era. Neste período, a taxa de erros humanos presentes em contas era muito grande. Isso estimulou a imaginação de *Charles Babbage* (1791-1871) um eminente matemático, engenheiro e inventor britânico cujas contribuições pioneiras tiveram um papel fundamental na história da computação conforme detalharemos a seguir. *Babbage* começou a pensar em alguma forma de mecanizar esse tipo de tarefa, eliminando as falhas e economizando o tempo das pessoas para realizarem tarefas (HYMAN, 1985).

Por volta de 1821, *Babbage* começou a tarefa de automatizar a produção de cálculos matemáticos, para tanto projetou o calculador analítico-mecânico. Inspirando-se na criação de *Jacquard* teve a ideia de utilizar cartões perfurados para armazenar instruções, dados e até desvios condicionais, uma inovação que influenciou a arquitetura moderna dos computadores (DUNCAN, 2009).

Por volta de 1834, o calculador analítico-mecânico foi desenvolvido, representando a primeira máquina programável com capacidade para executar diversos tipos de comandos (Swade, 2005). Sendo a primeira máquina que poderia ser programada para executar vários comandos de qualquer tipo. O mais interessante é que o desenho e a estrutura básica da invenção de *Babbage* fazem parte dos computadores que usamos ainda hoje.

A Máquina Analítica, movida a vapor, funcionava com base nos princípios dos cartões perfurados e apresentava uma distinção entre Unidade Central de Processamento (UCP) e memória expansível, ou seja, o projeto possuía uma UCP e memória expansível separados um do outro, o que é mais uma característica dos computadores modernos (WILLIAMS, 1985).

Fato Curioso

Babbage concebeu ainda um segundo modelo de Máquina Analítica, no entanto, a implementação da mesma não foi possível devido as barreiras tecnológicas da época, que impediram a realização prática do primeiro projeto de computador de uso geral.

Visto que *Babbage* desenvolveu tal projeto em um período de transição, entre técnicas artesanais tradicionais e processos de produção em massa, o que impossibilitava a fabricação de peças idênticas. Este projeto é considerado a concepção de um primeiro computador de uso geral (HYMAN, 1985), há que se considerar ainda, que o não desenvolvimento da máquina analítica de *Babbage* não deve ser interpretado como uma lacuna em sua genialidade, mas sim como um testemunho das complexas interações entre visão inovadora e limitações tecnológicas intrínsecas a um período específico da história da ciência e tecnologia (SWADE, 2005).

Dica de Vídeo

Para saber em detalhes o funcionamento da máquina de *Babbge*, assista ao vídeo intitulado “A Máquina de Charles Babbage”, o qual está disponível no *YouTube* através do [link](#):

<<https://www.youtube.com/watch?v=JtYAbLwnKfA>>

1815-1852 – *Augusta Ada Lovelace* se destaca ao registrar suas contribuições em anotações anexadas à tradução de um artigo de *Charles Babbage*. Em suas anotações ela formulou programas destinados à máquina projetada por *Babbage*, nos quais ela indicava como seria a configuração da entrada de dados, assim como o fluxo de operações a serem conduzidas pela UCP.

Fatos Curiosos

Ada Lovelace é considerada a primeira programadora da história. É extremamente importante destacar-se a relevância da presença feminina na história da computação em especial na história da programação. Em que se destaca como seu primeiro e mais emblemático exemplo a própria *Ada Lovelace*, não sendo a única (Essinger, 2014).

A Condessa de *Lovelace* foi também homenageada, ao ter seu nome dado a uma linguagem de programação. A linguagem ADA até hoje é utilizada pode ser adquirida no site oficial de sua comunidade em <<https://www.adacore.com/about-ada>> ou no site do *SourceForge*¹ em <<https://sourceforge.net/projects/gnuada/>>

Dentre várias outras personalidades femininas da área da Computação, destacam-se por exemplo também: *Grace Murray Hopper* programadora responsável pelo desenvolvimento da linguagem de programação COBOL (WALLMARK, 2020), assim como *Margaret Hamilton* a programadora cujas habilidades foram cruciais para o sucesso da missão de pouso do homem na Lua (INCERTI; CASAGRANDE, 2018). A história da programação se enriquece quando se lança luz sobre essas contribuições muitas vezes subestimadas, evidenciando a diversidade de talento e perspectivas no âmbito da tecnologia.

Para saber mais

Para conhecer mais sobre a trajetória da primeira programadora de computadores, assista aos vídeos sobre quem foi *Ada Lovelace*, os quais estão disponíveis no *YouTube* através dos [links](#) a seguir:

<<https://www.youtube.com/watch?v=kveunrBU5UM>>

<<https://www.youtube.com/watch?v=GZO7cGo93B8>>

1854 – *George Boole* publica os princípios da lógica booleana, através da qual descreve a base conceitual para a representação e manipulação de informações por meio dos símbolos ‘0’ e ‘1’, os quais se tornam uma significativa contribuição ao campo da lógica e estabelecem os alicerces para as modernas tecnologias computacionais e sistemas digitais (WILLIAMS, 1985).

1890 – *Herman Hollerith* funda uma companhia que viria a se tornar a IBM - International Business Machines Corporation, anteriormente conhecida como Tabulating Machine Company (TBM). *Hollerith* introduz um método pioneiro para a manipulação eficiente de dados ao utilizar cartões perfurados para processar o censo americano de 1890. E alcançando com isso uma redução de tempo de processamento em cerca de um terço em comparação aos métodos convencionais da época (WILLIAMS, 1985) os quais até então demoravam oito anos para serem realizados

Desafio

Você conhece mais alguma personagem feminina importante para a área da computação? Pesquise na Internet e/ou em livros e surpreenda-se com os resultados.

¹ É um repositório de páginas dedicado para que programadores e equipes desenvolvam, colaborem e distribuam *software open-source* <<https://sourceforge.net/>>.

A Era da Eletrônica (1930-1940): Invenção do Tubo de [Vácuo](#)¹ e criação do ENIAC.

1936 – *Konrad Zuse* inventor, engenheiro e empresário alemão, construiu e tentou vender o *Z1* junto às forças militares alemãs para a realização de cálculos balísticos. O *Z1* era o projeto de computador-eletromecânico que marcou um passo significativo em direção à construção de dispositivos de processamento de dados, todavia o mesmo não é reconhecido como o primeiro computador eletrônico por sua dependência de [relés](#)² (ROJAS; HASHAGEN, 2002).

Os sucessores *Z2* e *Z3* derivaram das concepções e princípios originais do *Z1*. O *Z2*, foi desenvolvido em 1939, foi um computador mecânico que manteve a estrutura da memória mecânica do *Z1*, porém substituiu os componentes de aritmética e lógica de controle por circuitos de relé elétrico. O *Z3*, por sua vez foi criado em 1941, marcando a história ao se tornar a primeira máquina programável do mundo operada por instruções. Totalmente automatizado e equipado com recursos de ramificação condicional, o *Z3* empregava cerca de 2.000 relés. Durante a Segunda Guerra Mundial, o *Z3* desempenhou um papel crucial na codificação de mensagens, sendo operado por uma equipe de 15 pessoas em uma instalação anexa à fábrica da empresa *Henschel & Sohn*. Infelizmente, o *Z3* original foi destruído em 1943 durante um ataque aéreo aliado em Berlim. (ROJAS, 1998).

1944 – Um esforço colaborativo entre a Universidade de Harvard e a Marinha americana foi iniciado com o objetivo de construir o *Mark 1* (também conhecido como *Colossus*) conforme o *Projeto Colossus* citado anteriormente que veio a ser mais um marco no desenvolvimento das capacidades de processamento, embora não seja plenamente considerado o primeiro computador pois também utilizava relés (COPELAND, 2004).

Para saber mais

Para conhecer alguns detalhes do Projeto “*Mark I*” – primeiro computador programável, o qual atualmente fica em exposição em um dos halls da Universidade de Harvard em Boston - USA, acesse ao vídeo disponível no *YouTube* através do [link](#):

<https://www.youtube.com/watch?v=GuOlkl_uEA>

1912-1954 – Destaque especial a *Alan Turing* considerado o pai da Ciência da Computação, responsável pelo *Projeto Enigma* na 2ª Guerra Mundial com o objetivo de quebrar códigos de segurança utilizados pela Alemanha nazista durante aquele período. O projeto logrou êxito e uma excelente sugestão para entender um pouquinho mais sobre o assunto, e todo o contexto a sua volta, é assistir ao filme “*O Jogo da Imitação*” de 2014.

Destaque ainda para a criação do conceito de uma *Máquina Teórica* conhecida como *Máquina de Turing* que serviu como base para toda a computação moderna (STRATHERN, 2000) (MCCARTNEY, 1999).

Para saber mais

Para conhecer em detalhes como era o funcionamento da máquina de encriptação de mensagens alemã *Enigma*, assista ao excelente vídeo desenvolvido pelo Museu da Universidade Federal do Rio Grande do Sul, intitulado “*Demonstração Máquina Enigma - Museu da UFRGS*”, disponível no canal oficial do Museu no *YouTube*, através do [link](#):

<<https://youtu.be/VMJeDLv2suw?t=87>>

¹ Um tubo de **vácuo** é um dispositivo que age como uma "chave" para a eletricidade, controlando o fluxo de energia entre partes diferentes dentro de um espaço vazio. Esses tubos eram muito importantes nos primeiros aparelhos eletrônicos, como rádios, TVs e computadores antigos.

² **Relés** operam como um interruptor elétrico que pode ligar ou desligar um caminho para a eletricidade fluir. É como um "guarda" que decide se o tráfego de eletricidade pode passar ou não.

Gerações de Computadores

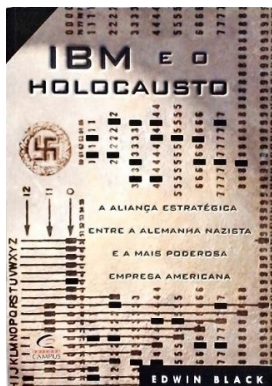
A Primeira Geração: **Válvulas!** (1940-1950)

Os primeiros computadores não possuíam Sistemas Operacionais (SO), o que implicava em um acesso direto do usuário ao *hardware* da máquina. A programação, era executada diretamente na linguagem de máquina, sendo necessário a inserção manual de todas as instruções através de dispositivos de chaveamento de circuitos. Neste período primordial da computação imperava a máxima de que "*Computadores são caros! Pessoas são baratas!*", evidenciando uma abordagem em que o esforço humano era otimizado para compensar a complexidade e as limitações dos dispositivos. A primeira geração de computadores foi caracterizada pelo uso de válvulas para implementar elementos lógicos digitais e funções de memória (STALLINGS, 2017).

Dica

Você aluno(a) que chegou até aqui, certamente já percebeu há tempos que para se aprofundar em alguns temas, não pode se valer apenas da Internet, com suas redes sociais e da mídia em geral.

Neste sentido e de modo a incentivá-lo(a) a refletir sobre a 1ª geração de computadores frente ao período em que ocorreu, indicamos a leitura do livro "**IBM e o Holocausto: A aliança estratégica entre a Alemanha e a mais poderosa empresa Americana**", onde o autor **Edwin Black** desvendou um dos grandes mistérios do Holocausto, qual seja, como **Hitler** conseguiu os nomes? – Quando o autor afirma que os números assombrosos assassinatos atingidos pelo nazismo, só foram possíveis em parte, pela assistência tecnológica que a IBM forneceu ao 3º Reich.



O livro físico está disponível entre outros, no seguinte link:

<<https://www.amazon.com.br/IBM-Holocausto-Edwin-Black/dp/8535207597>>

1946-ENIAC – Os cientistas norte-americanos **John Presper Eckert** e **John Mauchly** concebem e constroem o primeiro computador digital eletrônico de grande escala o ENIAC (*Electronic Numeric Integrator And Calculator*). O ENIAC foi desenvolvido para suprir as demandas do *Ballistic Research Laboratory* de realizar cálculos e edições de tabelas balísticas de forma ágil e precisa (ALMEIDA, 2000). O ENIAC entrou oficialmente em operação em julho de 1946, sendo a primeira grande estrela desta primeira geração com suas cerca de 19 mil válvulas, dois metros de altura, 30 toneladas e ocupando cerca de 180 metros quadrados (MCCARTNEY, 1999).

Você sabia

O termo *Bug* utilizado comumente quando algo sai errado na área da computação, surgiu nesta época devido aos insetos que atraídos pelo calor e luminosidade das válvulas do ENIAC acabavam por ocasionar danos e consequente mau funcionamento do computador por meio de curtos-circuitos e outros problemas eletrônicos. A figura 4, a seguir apresenta um dos primeiros relatórios sobre este tipo de incidente.

Figura 4: Origem do termo Bug

1903-1957 – **John von Neumann** formalizou a arquitetura básica de um computador, a qual é utilizada até hoje. o nome de von Neumann é intrinsecamente associado à "**Arquitetura de von Neumann**", ou seja, à estrutura clássica, de computadores digitais com programa armazenado na própria memória (KOWALTOWSKI, 1996).

Você sabia

A grande maioria dos computadores que existem atualmente segue um modelo proposto por volta de 1940 pelo matemático húngaro de origem judaica, naturalizado estadunidense, **John von Neumann**. Nesse modelo, um elemento processador segue as instruções armazenadas em uma memória de programas, para ler canais de entrada, enviar comandos sobre canais de saída e alterar as informações contidas em uma memória de dados.

¹Uma válvula se parece com uma lâmpada, mas é mais complexa. E isso não é coincidência, tanto a válvula quanto a lâmpada têm um vidro vazio com um filamento que conduz eletricidade. Nas lâmpadas, o filamento aquece e emite luz, iluminando. Nas válvulas, o filamento aquece uma placa chamada cátodo "eletrodo negativo do tubo".

A Segunda Geração: Transistores e Computadores de Grande Escala (1950-1960)

1947 – A universidade de Stanford inventa o transistor sendo que esta invenção marca um ponto de virada na história da computação. O transistor representou um marco significativo na evolução dos dispositivos eletrônicos, visto que, substituíram com grande vantagem as válvulas anteriormente utilizadas. Além de apresentarem vantagens econômicas, os transistores se destacavam por sua maior velocidade de operação e longevidade (STALLINGS, 2017).

Fato curioso

O transistor, que é menor, mais barato e gera menos calor do que a válvula, pode ser usado da mesma maneira que uma válvula para construir computadores. Ao contrário da válvula, que requer fios, placas de metal e cápsula de vidro, além de vácuo, é um dispositivo de estado sólido, feito de silício. O transistor foi inventado na *Bell Labs* em 1947, e na década de 1950 proporcionando uma revolução eletrônica. Foi só no final dessa década, no entanto, que computadores totalmente transistorizados estiveram comercialmente disponíveis (STALLINGS, 2017).

O uso de transistores define a segunda geração de computadores impulsionando uma categorização convencional baseada nas tecnologias subjacentes empregadas em seus componentes de *hardware*. Tornou-se amplamente aceito classificar os computadores em gerações com base na tecnologia nos fundamentos de *hardware* empregados. Este critério classificatório, amplamente aceito, estabelece distintas gerações de computadores que sucedem uma à outra.

Cada nova geração é caracterizada por maiores desempenhos de processamento, maior capacidade de memória e menor tamanho do que o anterior (STALLINGS, 2017). É importante entender ainda que a classificação por gerações, a partir das inovações tecnológicas, constitui um paradigma essencial para compreender a evolução da computação.

Você sabia

A segunda geração viu uma introdução de Unidades Lógicas e Aritméticas (ULA) / ALU (*Arithmetic Logic Unit*) e Unidades de Controle (UC) / CU (*Control Unit*), o uso de linguagem de programação de alto nível e a disponibilização dos *softwares* de sistema com o computador. Em termos gerais, o *software* de sistema proporcionou a capacidade de carregar programas, mover dados a periféricos e bibliotecas para executar computações comuns, similar ao que fazem os sistemas operacionais modernos, como *MS-Windows*, *macOS* e *Linux* (STALLINGS, 2017).

A 2ª geração a qual compreendeu o período de 1950 à 1960, é caracterizada pela transição significativa do uso de válvulas para a adoção dos transistores, que trouxeram melhorias notáveis em termos de eficiência, tamanho e confiabilidade dos sistemas computacionais. Durante essa fase, surgiram diversos computadores de grande escala que marcaram avanços substanciais na capacidade de processamento e armazenamento de dados.

A seguir estão listados alguns computadores notáveis que se destacaram na segunda geração (CAMPBELL-KELLY *et al.*, 2023).

IBM 709 (1957): Parte da série de computadores IBM 700/7000, foi um dos primeiros sistemas a utilizar transistores em vez de válvulas. Era capaz de processar informações com maior velocidade e precisão em comparação com suas contrapartes baseadas em válvulas. Foi utilizado em aplicações científicas e empresariais, desempenhando um papel crucial em várias áreas, incluindo pesquisa, simulações e processamento de dados (PUGH *et al.*, 1991).

UNIVAC (1957): Desenvolvido pelo LARC (*Livermore Automatic Research Computer*) foi um dos primeiros computadores de grande escala a utilizar transistores. Foi projetado para aplicações científicas e de pesquisa, sua utilização inovadora de transistores contribuiu para um aumento significativo no desempenho e eficiência do processamento de dados (WILLIAMS, 1985).

UNIVAC II (1958): Foi um dos primeiros computadores comerciais a utilizar transistores, tendo sido desenvolvido para substituir os sistemas baseados em válvulas, consequentemente trouxe melhorias em termos de eficiência, tamanho e confiabilidade. O UNIVAC II foi amplamente utilizado em empresas e organizações para processamento de dados e tarefas comerciais (WILLIAMS, 1985).

Terceira Geração: Circuitos Integrados e Microprocessadores - Era dos Computadores Pessoais (1960-1970)

Nos anos 1950 e início dos anos 1960, a paisagem da eletrônica era caracterizada por uma abordagem conhecida como "componentes discretos". Nesse contexto, um transistor individual, isolado e autocontido, era denominado "componente discreto". Os componentes discretos eram fabricados separadamente, empacotados em seus próprios invólucros e soldados ou ligados em placas de circuito, que eram então instaladas nos computadores, osciloscópios e outros equipamentos eletrônicos.

Imagine, por exemplo, que um equipamento eletrônico necessitava de um transistor. Naquele tempo, era necessário soldar manualmente um pequeno invólucro metálico contendo uma peça minúscula de silício do tamanho de uma cabeça de alfinete em uma placa de circuito. Todo esse processo de fabricação, desde a produção do transistor até a montagem na placa de circuito, envolvia uma série de etapas complexas e dispendiosas (SWAINE; FREIBERGER, 2014).

O cenário supracitado trouxe uma série de desafios significativos na indústria dos computadores. Quando olhamos para os primeiros computadores da segunda geração, percebemos que eles incorporavam em torno de 10.000 transistores. No entanto, esse número começou a crescer rapidamente, atingindo a marca de centenas de milhares de transistores em máquinas subsequentes. Isso, por sua vez, estava introduzindo complexidades consideráveis no processo de fabricação, uma vez que a criação de novos e mais poderosos computadores estava se tornando cada vez mais intrincada (CREASE; MANN, 1996).

Em 1958, chegou à realização que revolucionou a eletrônica e iniciou a era da microeletrônica: a invenção do Circuito Integrado (CI), que define a terceira geração de computadores. Agora, com a chegada do circuito integrado, esses elementos poderiam ser incorporados em um único *chip* de silício. Isso permitiu que mais componentes fossem colocados em um espaço muito menor e a um custo relativamente mais baixo. Essa inovação teve um impacto profundo na indústria da computação, possibilitando a criação de computadores mais poderosos, compactos e eficientes. A terceira geração de computadores estava oficialmente então em andamento (STALLINGS, 2017).

Na 3ª geração vários computadores notáveis surgiram, sendo que alguns deles merecem destaque devido aos avanços significativos na capacidade de processamento e no uso de tecnologias mais avançadas.

Um destes destaques foi o **IBM System/360**, lançado em 1964. Famoso pela introdução da ideia de compatibilidade entre diferentes modelos, permitindo assim que as organizações atualizassem seus sistemas sem a necessidade de reescrever *software*. Outro exemplo digno de nota foi o **CDC 6600** lançado em também 1964, ele foi pioneiro em termos de desempenho, sendo considerado o computador mais rápido da época. Vale ainda uma breve nota sobre o **DEC PDP-8**, lançado em 1965, o qual desempenhou papel fundamental na popularização dos minicomputadores. Com um *design* compacto e mais acessível em termos de custo, o **PDP-8** foi amplamente adotado em universidades e empresas (CAMPBELL-KELLY *et al.*, 2023).

Durante a terceira geração de computadores, diversos avanços tecnológicos notáveis emergiram, contribuindo para transformações significativas no cenário da computação. Algumas destas inovações incluem:

O surgimento dos Sistemas de Tempo Compartilhado (*Time-Sharing*), que permitiram a interação de vários usuários em um mesmo sistema simultaneamente. O surgimento dos Sistemas em Tempo Real (*Real-time Systems*), que permitiram que se fornecessem respostas dentro de limites de tempo definidos. Esses sistemas encontraram aplicações críticas em campos como aviação, automação industrial e controle de processos, onde a tomada de decisões precisa ocorrer em frações de segundos (CAMPBELL-KELLY *et al.*, 2023).

E finalmente já no finalzinho da terceira geração surgem os Computadores Pessoais (PCs - *Personal Computers*) tendo como seu principal representante o microcomputador **Altair**. Sendo que os primeiros passos rumo à criação de computadores pessoais foram dados durante essa época, sinalizando uma tendência significativa para as gerações subsequentes (WILLIAMS, 1985). Esses marcos tecnológicos da terceira geração ilustram as mudanças profundas que moldaram a evolução da computação, refletindo um cenário diversificado de avanços que impactaram tanto o âmbito comercial quanto as aplicações científicas e sociais.

O **Altair 8800** foi um importante computador lançado em 1975, o mesmo ajudou a criar a ideia de computadores pessoais. O equipamento possuía um *chip* Intel 8080, e permitiu uma maneira nova de as pessoas interagirem com um microcomputador, sendo um dos primeiros computadores que as pessoas podiam construir por conta própria.

Foi ainda importante na formação da *Microsoft Corporation*, pois para o mesmo os futuros fundadores da empresa, *Bill Gates* e *Paul Allen*, desenvolveram uma versão do interpretador BASIC. Esse marco definiu o curso da *Microsoft* como uma das empresas mais influentes na história da tecnologia e seu legado pode ser observado na proliferação de microcomputadores e na rápida evolução da indústria de tecnologia que se seguiu (RHEINGOLD, 2000).

Gerações posteriores

Além da terceira geração, o consenso em relação à definição das diferentes gerações de computadores é relativamente limitado. A tabela 1: “Geração de Computadores” a seguir, sugere a existência de várias gerações subsequentes, delineadas pelo progresso na tecnologia de circuitos integrados.

Tabela 1: Gerações de Computadores

Fonte: (STALLINGS, 2017) pg. 15

Na figura 5 “Crescimento no número de transistores nos circuitos integrados” a seguir, é possível observar que com a introdução da Integração em Grande Escala (LSI - Large Scale Integration), tornou-se possível incorporar mais de 1.000 componentes em um único chip de circuito integrado. Esse avanço foi posteriormente superado pela Integração em Escala Muito Grande (VLSI - Very Large Scale Integration), alcançando a marca de mais de 10.000 componentes por chip, enquanto os chips com Integração em Escala Ultragrande (ULSI - Ultra-Large Scale Integration) podem comportar mais de um bilhão de componentes (STALLINGS, 2017).

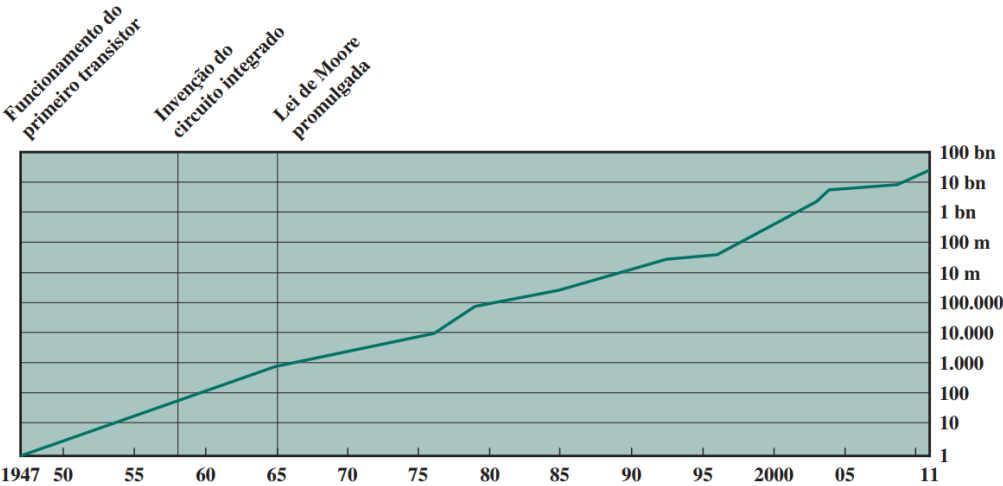


Figura 5: Crescimento no número de transistores nos circuitos integrados

Fonte: (STALLINGS, 2017) pg. 18

Fato curioso

A famosa **Lei de Moore** foi proposta em 1965 pelo cofundador da empresa Intel Corporation *Gordon Moore*. *Moore* observou que o número de transistores que poderiam ser colocados em um único chip dobrava a cada ano e previu corretamente que esse ritmo continuaria no futuro próximo. Para a surpresa de muitos, incluindo o próprio *Moore*, o ritmo continuou ano após ano e década após década. O ritmo diminuiu para dobrar a cada 18 meses na década de 1970, mas sustentou essa taxa desde então (STALLINGS, 2017).

Quarta Geração de Computadores: Surgimento dos Microprocessadores (1977-1991)

A transição para a 4ª geração foi impulsionada pela miniaturização dos componentes eletrônicos, que culminou com a tecnologia de microprocessadores permitindo assim que a lógica computacional fosse integrada em um único chip semicondutor, resultando em computadores mais poderosos, compactos e acessíveis. A 4ª geração de computadores causou impactos significativos na computação, visto que expandiu o seu alcance para uma gama mais ampla de aplicações. Contribuindo ainda para o surgimento e a popularização do computador pessoal, o que permitiu que as pessoas tivessem acesso a computadores poderosos em suas casas (CAMPBELL-KELLY et al., 2023).

É importante frisar que em consequência destas mudanças e evoluções aconteceu uma verdadeira revolução na disseminação da informação que acabou por levar o ser humano a uma nova era denominada como a “**Era da Informação**” (DRUCKER, 2000). Todo o legado da 4ª geração de computadores perdura ainda nos dias atuais, podendo ser encontrada nos dispositivos eletrônicos modernos de *smartphones* a sistemas embarcados. Sendo assim é facilmente percebível que a transição para a quarta geração estabeleceu todo um padrão de inovação contínua e constante na computação impulsionando avanços significativos na evolução tecnológica. A quarta geração de computadores trouxe diversos avanços que moldaram a computação atual (GUGIK, 2009). Dentre os quais destacam-se os seguintes:

A performance do *hardware* cresce exponencialmente, resultando em sistemas mais eficientes e rápidos. Simultaneamente, acontece uma queda significativa nos custos de processamento e armazenamento, o que vem a tornar a tecnologia computacional mais acessível;

Surgem sistemas operacionais como o *UNIX*, o *MS-DOS*, o *MS-Windows* e ainda o *Macintosh* da Apple Inc., que pavimentaram o caminho para a interface gráfica com o usuário. Sendo que as Interfaces Gráficas com o Usuário (GUI - *Graphical User Interface*) se tornaram uma característica marcante nesta geração;

Os computadores pessoais se tornaram populares sendo os anos 80 e 90 considerados as décadas dos PCs e das *workstations*. A Microsoft Corporation emerge como uma força dominante com o sistema operacional *Windows*, incorporando conceitos dos primeiros sistemas operacionais *Macintosh*. Essa integração permitiu que os usuários navegassem facilmente por múltiplas aplicações concorrentes; e

A padronização, aceleração e popularização da computação distribuída, com sistemas operacionais que passaram a suportar tarefas em rede ganhou destaque, com a transferência de informações via rede se tornando prática e economicamente viável, o que impulsionou a disseminação do modelo *client/server* ou cliente/servidor (CERUZZI, 2003).

Para Saber Mais

As gerações de computadores são marcadas por revoluções tecnológicas e sempre estiveram ligadas a diferentes fatores da sociedade, para saber mais assista ao vídeo “HISTÓRIA: A EVOLUÇÃO DOS COMPUTADORES” disponível no YouTube através do link <<https://www.youtube.com/watch?v=mFdUqgwzbVs>> e conheça em detalhes como as mesmas ocorreram.

Sistemas Numéricos e Aritmética Binária

Numeração nas bases 10, 2, 8 e 16 + Aritmética Binária

Os sistemas de numeração foram criados pelo homem com o objetivo de quantificar e representar grandezas decorrentes de observações. Esses sistemas, foram concebidos a partir da utilização de símbolos e caracteres, e do estabelecimento de regras para a sua representação gráfica. A base ou raiz do sistema, denotada como "r", consiste no conjunto de símbolos ou caracteres que são utilizados para compor o sistema. A base de um sistema de numeração é um valor que é empregado para expressar uma quantidade, sendo frequentemente igual ao número de caracteres distintos utilizados na construção do sistema. Dentre os sistemas numéricos mais utilizados estão o **Decimal**, o **Binário**, o **Octal** e o **Hexadecimal** (STALLINGS, 2017). Para fins de referência, neste contexto, iremos utilizar o sistema decimal como ponto de partida.

O Sistema Decimal

No dia a dia, usamos um sistema baseado em dígitos decimais para a representação de números, e nos referimos a esse sistema como **sistema decimal**. O sistema decimal é conhecido como **uma base 10**, visto que emprega uma seleção de **dez** caracteres distintos para a representação dos números, correspondendo aos dígitos de **0 a 9**. Uma característica distintiva do sistema decimal reside na sua base numérica de **10**, o que implica que cada posição numérica à esquerda de um número possui um peso dez vezes superior em relação à posição numérica imediatamente à sua direita, sendo que o sistema se vale de um conjunto de **10** algarismos (ou símbolos) para expressar eficazmente qualquer quantidade. Esses algarismos, a saber, **0, 1, 2, 3, 4, 5, 6, 7, 8 e 9**, formam a base estrutural do sistema decimal (STALLINGS, 2017).

Exemplo: $(5248)_{10} = 5 \times 10^3 + 2 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$

Em qualquer número, o dígito mais à esquerda é conhecido como dígito mais significativo, pois ele contém o valor mais alto. O dígito mais à direita é chamado de dígito menos significativo. Ou seja, o sistema decimal, então, é um caso especial de um sistema numérico posicional com raiz **10** e com dígitos no intervalo **0 a 9** (STALLINGS, 2017).

Por outro lado, é importante destacar que existe diversos sistemas de numeração, visto que há diversas abordagens para a representação de valores numéricos. A adoção de um sistema de numeração alternativo não apenas conduz a uma alteração na maneira pela qual um número é representado, mas também ocasiona modificações nas operações aritméticas que podem ser aplicadas dentro desses novos contextos. A introdução da notação posicional desempenha um papel crucial ao possibilitar o cálculo da magnitude representada por um número específico em um dado sistema numérico.

A representação de um número em um sistema de numeração diferente muda para um mesmo valor, assim como as operações com números nesses novos sistemas podem ser readequadas. A notação posicional permite calcular a quantidade que um número representa (STALLINGS, 2017).

Por exemplo: Que quantidade representa o símbolo 1? Se você respondeu: “Um, oras!” ... **errou feio!** A resposta correta é “depende!” – Mas depende de quê? Da posição em que ele aparece no número completo!

Ao escrever um número estamos representando uma quantidade com base em símbolos que equivalem à uma quantidade. Pense no financeiro: a nota de **\$100** só representa que você possui algo que vale **\$100**, mas a nota não tem esse valor. Observe o número **1537...** O que ele significa, em termos de contagem?

$1 \times 1000 + 5 \times 100 + 3 \times 10 + 7 \times 1$

Observe que o valor de contagem de cada símbolo (algarismo) depende da sua posição.


Os números decimais são os mais utilizados atualmente de nosso conhecimento. Uma representação posicional no sistema decimal pode ser desenvolvida numa forma polinomial que envolve um somatório de potências de **10** (STALLINGS, 2017).

A tabela 2 intitulada “Interpretação Posicional de Números Decimais” apresenta uma ilustração da relação entre cada posição ocupada por um dígito em um número decimal e o valor associado a essa respectiva posição. Nesse contexto, cada posição é ponderada por um fator multiplicativo de **10** vezes o valor da posição situada à sua direita, bem como por um décimo do valor da posição à esquerda. A adoção dessa ponderação resulta em uma representação que reflete potências sucessivas da base **10**, contribuindo para a compreensão da estrutura numérica subjacente. A numeração sequencial das posições, conforme delineado na tabela, conduz a uma valiosa conclusão: a posição de índice “i” é ponderada por um valor equivalente a **10** elevado à potência “i” (STALLINGS, 2017).

Tabela 2: Interpretação posicional de um número decimal

4	7	2	2	5	6
100s	10s	1s	décimo	centésimos	milésimos
10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}
posição 2	posição 1	posição 0	posição -1	posição -2	posição -3

Fonte: (STALLINGS, 2017) pg. 272

Em geral, para a representação decimal de , o valor de x é $x = \sum_i d_i x 10^i$

Em um sistema numérico posicional, cada número é representado por uma cadeia de dígitos em que cada posição “i” do dígito tem um peso associado “ri”, em que “r” é a raiz, ou base, do sistema numérico. A forma geral de um número em tal sistema com raiz “r” é:



Representação = $a_3 a_2 a_1 a_0$

Quantidade = $a_0 \times b^0 + a_1 \times b^1 + a_2 \times b^2 + a_3 \times b^3 + \dots$

2734

$$\begin{aligned}
 a_0 &= 4 \rightarrow 4 \times 10^0 = 4 \times 1 = 4 \\
 a_1 &= 3 \rightarrow 3 \times 10^1 = 3 \times 10 = 30 \\
 a_2 &= 7 \rightarrow 7 \times 10^2 = 7 \times 100 = 700 \\
 a_3 &= 2 \rightarrow 2 \times 10^3 = 2 \times 1000 = 2000
 \end{aligned}$$

Exemplo: **Sistema Decimal**

Representação valor decimal

Casa	Milhar	Centena	Dezena	Unidade
	3	2	1	0
Dígito	4	5	3	2
Quantidade	4.000	500	30	2

Considere o número a seguir: $4532 = 4000 + 500 + 30 + 2$

Casa	Milhar	Centena	Dezena	Unidade
	3	2	1	0
Dígito	4	5	3	2
Quantidade	4.000	500	30	2

Casa	Milhar	Centena	Dezena	Unidade
	3	2	1	0
Dígito	4	5	3	2
Quantidade	4x 1000	5x 100	3x 10	2x 1

Casa	Milhar	Centena	Dezena	Unidade
	3	2	1	0
Dígito	4	5	3	2
Quantidade	4×10^3	5×10^2	3×10^1	2×10^0

Observe: Na casa 3, há 3 zeros; na casa 2, há 2 zeros... E assim por diante! Isso não ocorre por acaso!

Como exemplo, o número: **6007 (Seis Mil e Sete)**

- $6007 = 6 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$
- $6007 = 6 \times 1000 + 0 \times 100 + 0 \times 10 + 7 \times 1$
- $6007 = 6000 + 0 + 0 + 7$
- $6007 = 6007$

Por que “10”? Porque a base é decimal e temos **10** símbolos para representar cada dígito.

Você sabia

O mesmo artifício descrito para o sistema posicional é utilizado em outros sistemas de numeração, ou seja, cada caractere que compõe um número possui um “**peso**” de potências do valor da base que variam de acordo com a posição ocupada pelo caractere no número (HARRIS; HARRIS, 2015).

Nota: O *Microsoft Windows* possui em sua calculadora uma opção de modo programador para conversão entre pesos.

Os sistemas de numeração foram criados pelo homem com o objetivo de quantificar as grandezas relacionadas às suas observações. Tais sistemas foram desenvolvidos por meio de símbolos, caracteres e do estabelecimento de regras para a sua representação gráfica.

Ao conjunto desses símbolos ou caracteres chamamos de **base** ou **raiz** - “r” ou “b” do sistema, sendo que a raiz indica quantos símbolos há por dígito.

A base **Binária** usa **dois** símbolos para cada dígito:

0, 1

A base **Octal** usa **oito** símbolos para cada dígito:

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

A base **Decimal** usa **dez** símbolos para cada dígito:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

A base **Hexadecimal** usa **dezesesseis** símbolos para cada dígito:

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

Sistema Binário de Numeração em Bases 10, 2, 8 e 16 + Aritmética Binária

Qual sistema numérico é utilizado nos computadores? É o **sistema binário**, o qual utiliza somente dois algarismos **0** e **1** que são os dígitos binários (**bits**) para representar qualquer quantidade. O termo *bit* vem das palavras (*binary digit*).

Mas por que utilizar o sistema binário e não o decimal, que já estamos tão bem acostumados no dia a dia? Porque com o sistema decimal seria muito difícil de implementar com circuitos digitais!

No sistema decimal, **10** dígitos diferentes são usados para representar números com uma base de **10**. Já no sistema binário, temos somente dois dígitos, **1** e **0**. Os quais representam muito bem os dois estados possíveis de um circuito (**ligado/desligado – on/off**), dessa maneira, os números no sistema binário são representados na **base 2**.

O sistema binário pode ser utilizado para representar dois estados de um elemento por exemplo uma lâmpada (**acesa** ou **apagada**), uma chave (**aberta** ou **fechada**), uma fita magnética (variação ou não na magnetização / **magnetizada** ou **desmagnetizada**).

A contagem segue o mesmo raciocínio utilizado no sistema decimal: após o último dígito, incrementa-se uma posição à esquerda, e a posição à direita é zerada, repetindo-se toda a sequência de números anterior.

Para evitar confusão, vamos algumas vezes colocar um subscrito em um número a fim de indicar sua base.

Por exemplo: **8240₁₀** e **4.7340₁₀** são números representados na notação decimal ou, resumidamente, números decimais. Os dígitos 1 e 0 na notação binária têm o mesmo significado na notação decimal.

Para evitar confusão com o sistema de numeração decimal, lemos dígito por dígito no sistema binário.

Exemplo: **10₂** ler como: “**um, zero**”, mais um exemplo **1101₂** ler como “**um, um, zero, um**”.

Atenção

Quando utilizamos sistemas de numeração diferentes, procura-se adotar uma convenção para a identificação de números com bases de numeração diferentes.

Exemplo: 

O número **11100** no sistema de **base 2** é igual ao número **28** no sistema decimal

No sistema binário cada posição da representação armazena até 2 valores, ou seja, Base: **2**, Símbolos: **0** e **1**. O sistema binário é o sistema numérico usado para representar as operações do computador que são executadas com o uso da lógica booleana que descreveremos mais à frente.

Para representar números maiores, como com a notação decimal, cada dígito em número binário tem um valor que depende de sua posição:

$$11_2 = (1 \times 2^1) + (1 \times 2^0) = 3_{10}$$

$$100_2 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4_{10}$$

$$101_2 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 5_{10}$$


Atenção

Valores fracionários são representados com as potências negativas da raiz:

$$1001,101_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) = 9,625_{10}$$

$$1001,101_2 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9,625_{10}$$

a representação binária do valor de $Y = \{ \dots \dots b_2, b_1, b_0, b_{-1}, b_{-2}, \dots \}$

onde o valor de y é: 

Você sabia

Determinados grupos de bits recebem nomes especiais:

Exemplo:

Bit: Menor unidade de informação

4 bits: *Nibble*

8 bits: *Byte* (*Byte* → unidade de transferência de informação = 8 *bits*)

16 bits: *Word*¹ (Palavra → múltipla do *byte* 32, 64, 128, etc.)

¹ Unidade básica de informação da computação, similar a palavra humana.

Sistema Octal

O sistema numérico de **base 8**, conhecido como **sistema octal**, caracteriza-se pela utilização sequencial dos dígitos de **0** a **7**, correspondentes à ordem dos dígitos no sistema de numeração decimal. Este sistema era muito utilizado antigamente, pois é uma simplificação do sistema binário. Essa simplificação é evidenciada pela

substituição de grupos de 3 dígitos binários por um único dígito no sistema octal, uma vez que o valor máximo representado por 3 dígitos binários, **111**, equivale a 7, que é o número máximo de dígitos distintos admitidos pelo sistema octal, operante em **base 8** (STALLINGS, 2010).

Atualmente, o sistema octal entrou em desuso pela utilização cada vez maior da informática e de circuitos eletrônicos digitais, que empregam somente números binários. Em substituição a esse sistema, surgiu o sistema hexadecimal, oferecendo uma abordagem alternativa para a representação eficiente de valores, especialmente no contexto computacional (STALLINGS, 2010).

Sistema Hexadecimal

É o sistema numérico usado para simplificar as representações usadas na computação, por exemplo para representar endereços de rede. Basicamente o **sistema hexadecimal** é uma versão compactada e simplificada do sistema binário onde os dígitos binários são agrupados em conjuntos de quatro *bits*, chamados de *nibble* (STALLINGS, 2010). No qual cada combinação possível de dígitos binários é dada por um símbolo conforme pode se observar na tabela 3 a seguir.

Tabela 3: Sistema Hexadecimal

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

Atenção: No sistema hexadecimal cada posição da representação armazena até **16** valores **Base 16** e Símbolos: **0** a **9**, **A** a **F**

Exemplos de algumas conversões entre essas bases

De qualquer base para decimal:

Basta resolver a expressão de representação. O sistema decimal, por ser o sistema natural, será usado como referência para todos os sistemas numéricos

$$\text{Representação} = a_3 a_2 a_1 a_0$$

$$\text{Quantidade}_{10} = a_0 \times b^0 + a_1 \times b^1 + a_2 \times b^2 + a_3 \times b^3$$

Exemplo:



Conversão entre bases a partir de um valor decimal

De decimal para qualquer base: divisão no número pela base onde a representação são os restos da divisão:

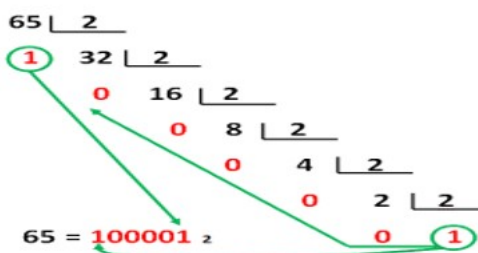
$$\text{Representação} = a_3 a_2 a_1 a_0$$

Exemplo numérico

$$a_3 a_2 a_1 a_0 / b$$

$$\begin{array}{r} r_0 \quad d_2 d_1 d_0 / b \\ r_1 \quad d_5 d_4 d_3 / b \\ r_2 \quad d_8 d_7 d_6 / b \\ r_3 \quad d_9 \end{array}$$

$$\text{Representação} = d_9 r_3 r_2 r_1 r_0$$



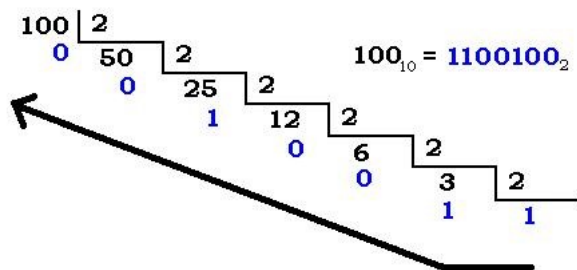
Conversão Sistema Decimal em Sistema Binário

Divide-se sucessivamente o número decimal por dois até resultar em um número menor que dois, e os restos dessas divisões com o último resultado formarão o número binário (HARRIS; HARRIS, 2015).

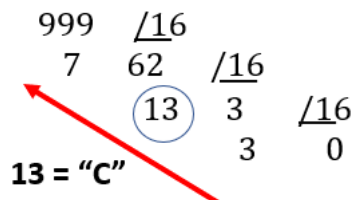
Esse mesmo método pode ser usado para outros sistemas de numeração de base diferente de 2, como o sistema octal que é base 8 ou o hexadecimal cuja a base é 16 (HENNESSY; PATTERSON, 2012).

Exemplos:

Base 2



Base 16

Representação = 999_{10} 

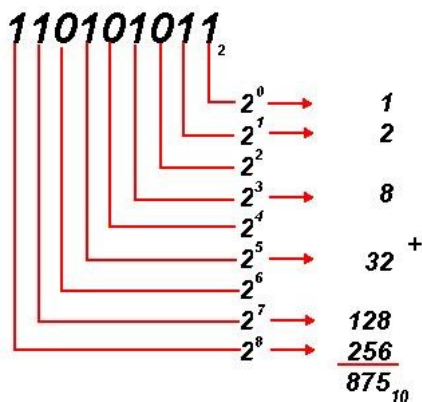
$$999_{10} = 03C7_2$$

Outro modo de conversão de Binário para Decimal e vice-versa, é criação de uma tabela com a base exponencial.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	1	0	0	1	1

$1 + 2 + 16 + 32 = 51$ Valor Decimal

Você pode fazer desta forma:



Ou ainda dessa:

$$\begin{aligned}
 &110101_2 \\
 &\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 &1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
 &\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 &32 + 16 + 0 + 4 + 0 + 1 = 53 \\
 &110101_2 = 53_{10}
 \end{aligned}$$

Aritmética Binária: Uso da Soma, Multiplicação e Potência da base

Operação de Adição em Números Binários

A adição binária é um processo fundamental no âmbito dos números binários, um sistema de numeração caracterizado pela sua base 2, onde somente dois algarismos estão em uso: **0 (zero)** e **1 (um)**. Durante essa operação, certos padrões resultam em representações específicas que merecem destaque.

Quando somamos o algarismo **0** a **1**, o resultado direto é **1**, refletindo a regra básica da adição. Porém, ao considerar a adição de **1** a outro **1**, um fenômeno (intrigante) ocorre. Nesse caso, o somatório resultaria em **2** no contexto decimal. No entanto, no sistema binário, o número **2** é representado como **10**. Assim, para manter a coerência dentro da base binária, o resultado é registrado como **0** (zero), e uma unidade é "transportada" para a posição subsequente.

Este processo de "carregar" um valor para o dígito à esquerda após a soma de **1** a **1** é um componente chave na operação de adição em números binários e desempenha um papel crucial na manutenção da integridade das representações numéricas (HENNESSY; PATTERSON, 2012). A seguir na tabela 4 é possível observar as regras para a soma binária.

Tabela 4: Soma binária

Regra	Cálculo	*Observação
Regra 1	0 + 0 = 0	-----
Regra 2	0 + 1 = 1	-----
Regra 3	1 + 0 = 1	-----
Regra 4	1 + 1 = 0	Com transporte de 1
Regra 5	1 + 1 + 1 = 1	Com transporte de 1

1 1 0 0;
+ 1 1 1;

= 1 0 0 1 1;

Exemplo de soma com aplicação da regra (Tabela 4)

Subtração Binária: Procedimento de Subtração em Números Binários

Quando nos deparamos com a situação de subtrair **1** de **0** no sistema binário, é necessário recorrer a um processo de "empréstimo" proveniente do dígito vizinho. No contexto binário, esse "empréstimo" assume um valor de **2**, uma vez que o sistema se baseia no binômio **0** e **1**.

Para entender melhor imagine do seguinte modo, ao realizar a subtração **0 - 1**, o resultado aparente é **1**; no entanto, a operação (em decimal) é, na verdade, **2 - 1**, resultando em **1**. Esse padrão de "empréstimo" é recorrente e gera uma alteração no valor do dígito que concedeu o "empréstimo", reduzindo-o de **1** para **0**. Os dígitos **1**, por sua vez, são marcados como aqueles que efetuaram esse "empréstimo" para seus dígitos vizinhos.

É importante salientar que, logicamente, quando um dígito possui valor **zero**, ele não tem a capacidade de "emprestar" para outros dígitos, o que faz com que a demanda de "empréstimo" seja transmitida ao próximo dígito subsequente, elevando o dígito inicial de **zero** a **um**. Sendo que por meio desses procedimentos que a subtração é executada no sistema binário, permitindo a manipulação precisa das representações numéricas dentro dessa base (HENNESSY; PATTERSON, 2012). A seguir na tabela 5 é possível observar as regras para a subtração binária

Tabela 5: Subtração binária

Regra	Cálculo	*Observação
Regra 1	0 - 0 = 0	-----
Regra 2	0 - 1 = 1	Com empréstimo de 1 (2) do seguinte
Regra 3	1 - 0 = 0	-----
Regra 4	1 - 1 = 0	-----
Regra 5	1 - 1 - 1 = 1	Com empréstimo de 1 (2) do seguinte

0 2 0 0 2
1 1 0 1 1 0
- 1 0 1 1 1

= 1 0 1 0 1 1

Exemplo de subtração com aplicação da regra (Tabela 5)

Complemento de 2

Na memória do computador podemos armazenar informações em formato binário. Estes sinais podem ser positivos ou negativos. Entretanto como podemos (definir) gravar o sinal de positivo e negativo? Para valores binários negativos precisamos representá-los por meio do complemento de 2. Por isso em alguns casos precisamos falar se uma variável numérica deve ou não ter sinal. Mas, como fazer?

Primeiro, converta o número positivo para binário. Não se esqueça que no computador temos um número de *bits* definido.

Se não completar todos os *bits* do computador, zero neles! Complete os mais significativos com zero até dar o número de *bits* do *word* (palavra).

Segundo, inverta todos os *bits*. O que é “0” passa a ser “1”, e o que “1” passa a ser “0”. Some “1” no *bit* menos significativo e pronto (HARRIS; HARRIS, 2015).

Pense, Calcule e Reflita sobre os resultados mostrados e comentados nas 3 questões a seguir.

1. Efetue as seguintes operações:

$$i) 1001_2 + 110_2 = 1111$$

$$iii) 11111_2 + 1_2 = 11110$$

$$ii) 101011_2 - 10101_2 = 10110$$

$$iv) 1111_2 - 110_2 = 1001$$

2. Como seria feita a conversão entre um número na **base octal** para uma **base 3**?

Fazendo sucessivas divisões pelo número 3, ou seja, da mesma forma que uma conversão binária, mas com 3 ao invés de 2.

3. Por que não vemos computadores com **128 bits** de dados?

Principalmente por que aumentar o tamanho dos bits implica em lidar com números maiores, o que pode resultar em maior consumo de recursos computacionais. Sem contar que aumentar para 128 bits pode não trazer ganhos significativos em desempenho para a maioria das tarefas, além de envolver questões de complexidade, compatibilidade e custo-benefício.

“E então, você conseguiu chegar aos mesmos resultados dos cálculos binários? Concorde com as repostas das questões 2 e 3?” - Pesquise e veja como se saiu!

Lógica Booleana e Circuitos Digitais

Operadores Lógicos, Álgebra Booleana e Tabela Verdade

Os computadores atuais manipulam os dados de forma binária, ou seja, toda a informação é traduzida (representada) em **zeros** e **uns**. O que é bastante lógico visto que cada transistor possui dois estados de funcionamento: que são: *Conduz* ou *Bloqueia* corrente ou *Possui* ou *Não Possui* tensão elétrica. Dentro deste contexto torna-se necessário que toda operação seja feita de acordo com a lógica booleana (STALLINGS, 2017).

A lógica booleana foi inicialmente desenvolvida para permitir automação de painéis de telefonia operados por relés eletromagnéticos. A álgebra booleana é baseada em variáveis e operações, sendo que estas variáveis e operações são lógicas. De tal forma na álgebra booleana uma variável pode ter o valor **1** (**VERDADEIRO**) ou **0** (**FALSO**).

As operações de lógica básica são:

AND (representadas pelo PONTO) → *Representação: $A \text{ AND } B = A . B$*

OR (representadas pela ADIÇÃO) → *Representação: $A \text{ OR } B = A + B$*

e

NOT (representada pela BARRA SOBRESCRITA) → *Representação: $\text{NOT } A = \sim A$*

Cada operação lógica possui uma tabela verdade. Sendo que uma tabela verdade mostra como as entradas e saídas se relacionam.

Você sabia

O nome “lógica booleana” é uma homenagem ao matemático inglês *George Boole*, que propôs os princípios básicos da álgebra em 1854 em seu tratado “*An Investigation of the Laws of Thought on Which to Found the Mathematical Theories of Logic and Probabilities*”. Sendo que em 1938, *Claude Shannon*, um pesquisador do MIT (*Massachusetts Institute of Technology*), sugeriu que a Álgebra Booleana poderia ser usada para resolver problemas em projetos de circuitos com [comutadores](#)¹ (STALLINGS, 2017).

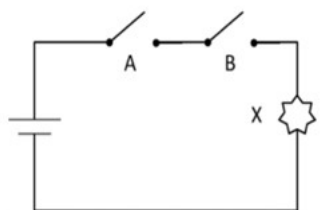
¹ Segundo Stallings (2017), um **comutador** trata-se de um dispositivo que oferece conexões físicas ponto a ponto para outros dispositivos direcionando e redirecionando o tráfego de dados.

Portas Lógicas e Circuitos Digitais, Microprocessadores e Microcontroladores

Sistemas digitais são formados por circuitos lógicos componentes eletrônicos chamados portas lógicas, que, quando empregadas estrategicamente, têm a capacidade de concretizar todas as expressões provenientes da Álgebra de Boole. Os sistemas digitais, possuem três portas lógicas fundamentais (**AND**, **OR** e **NOT**), cuja interconexão pode gerar uma variedade de configurações, abrangendo desde relógios digitais simples até computadores de grande porte.

A **operação lógica AND (E)** necessita da presença de dois argumentos para produzir um resultado. Sendo que, o resultado é **1** sempre que ambas as entradas assumem o valor 1, enquanto qualquer outra combinação sempre vai resultar em **0**. A função lógica **AND (E)** equivale à multiplicação de variáveis booleanas, sendo a sua representação algébrica para o caso de duas variáveis expressa como $S = A . B$ em que se lê $S = A \text{ e } B$ (STALLINGS, 2017):

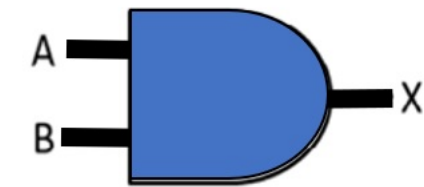
Para compreender a função **E** da álgebra booleana, deve-se analisar o circuito da Figura seguir para o qual se adota as seguintes convenções: **chave aberta = 0**, **chave fechada = 1**, **lâmpada apagada = 0** e **lâmpada acesa = 1**.



A lâmpada **X** acenderá (**1**) quando as chaves **A** e **B** forem fechadas (**1**)

A representação gráfica da operação **AND (E)** é comumente conhecida como **porta lógica E**, que é uma das peças básicas na construção de circuitos lógicos e sistemas digitais. A tabela verdade da operação **AND**, que lista todas as combinações possíveis de entrada e o resultado correspondente, é fundamental para compreender seu funcionamento e aplicação prática. A operação (função) **AND (E)** para duas variáveis de entrada **A** e **B** possui o seguinte símbolo (porta lógica) e a tabela verdade, como pode ser visto na tabela 6 “*Tabela Verdade da Operação OR*” a seguir.

Tabela 6: Tabela Verdade da Operação **AND**



$X = A \cdot B$
 $X = A \wedge B$
 $X = A \& B$
 $X = A \text{ AND } B$

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

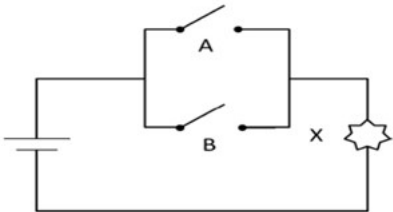
Expressão

Atenção: **AND (E)** retorna **1** somente se todas as entradas forem **1** e **0** para outras combinações.

Em resumo, a operação **AND** desempenha um papel crucial na lógica booleana e na concepção de circuitos digitais. Sua representação algébrica, símbolo lógico e tabela verdade são elementos fundamentais para a compreensão de sua aplicação em sistemas digitais complexos.

A operação lógica **OR (OU)** é um dos elementos fundamentais da álgebra booleana e tem aplicação direta em circuitos digitais e sistemas de processamento de informação. Esta operação requer dois argumentos, normalmente referidos como **A** e **B**, e produz um resultado que denotaremos como **S**. Em termos simples, o resultado **S** será igual a **1** quando pelo menos um dos argumentos (**A** ou **B**) for igual a **1**; caso contrário, **S** será igual a **0**. A operação **OR** é representada através do símbolo lógico "+", e sua representação algébrica para dois argumentos (variáveis) **A** e **B** pode ser expressa como $S = A + B$ que se lê **S = A ou B** (STALLINGS, 2017):

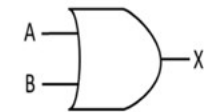
Para compreender a função **OR** da álgebra booleana, deve-se analisar o circuito da Figura seguir para o qual se adota as seguintes convenções: **chave aberta = 0**, **chave fechada = 1**, **lâmpada apagada = 0** e **lâmpada acesa = 1**.



A lâmpada **X** apagará (**0**) quando as chaves **A** e **B** forem abertas (**0**)

A representação gráfica da operação **OR** é comumente conhecida como porta lógica **OU**, também é uma das peças básicas na construção de circuitos lógicos e sistemas digitais. A tabela verdade da operação **OR**, que lista todas as combinações possíveis de entrada e o resultado correspondente, é fundamental para compreender seu funcionamento e aplicação prática. A operação (função) **OU** para duas variáveis de entrada **A** e **B** possui o seguinte símbolo (porta lógica) e a tabela verdade a seguir:

Tabela 7: Tabela Verdade da Operação **OR**



$X = A \vee B$
 $X = A + B$
 $X = A | B$
 $X = A \text{ OR } B$

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

Expressão: $X = A + B$

Atenção: **OR (O)**: retorna **1** se qualquer uma das entradas forem **1**.

Em resumo, a operação **OR** assim como a operação **AND** desempenha um papel crucial na lógica booleana e na concepção de circuitos digitais. Sua representação algébrica, símbolo lógico e tabela verdade são elementos fundamentais para a compreensão de sua aplicação em sistemas digitais complexos.

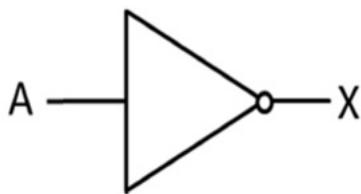
A operação lógica **NOT (não, em português)** diferentemente de outras operações, o **NOT** requer apenas um argumento, denotado como **A**, e produz um resultado que representa o complemento do bit de entrada. Em outras palavras, a operação **NOT** inverte o estado da variável de entrada, transformando **0** em **1** e **1** em **0**. A representação algébrica da operação **NOT** é expressa da seguinte forma: $\sim A$ em que se lê: **A barra** ou **NÃO A**.

Essa notação representa o resultado da operação **NOT** aplicada ao argumento **A**. O símbolo de barra sobre o **A** indica a negação ou inversão do valor de **A**. Por exemplo, se **A = 0**, então $\sim A$ será igual a **1**; e se **A = 1**, então $\sim A$ será igual a **0**.

A representação gráfica da operação **NOT (NÃO)** é comumente conhecida como porta lógica **NÃO**, também é uma das peças básicas na construção de circuitos lógicos e sistemas digitais. A tabela verdade da operação **NOT**, que lista todas as combinações possíveis de entrada e o resultado correspondente, é fundamental para compreender seu funcionamento e aplicação prática. A operação (função) **NOT (NÃO)** necessita de um argumento e um inversor como variável de entrada, a porta lógica **NOT** possui o seguinte símbolo (porta lógica) e a tabela verdade a seguir:

A operação (função) **NOT** inverte a entrada:

Tabela 8: Tabela Verdade da Operação **NOT**



A	X
1	1
0	1

$$X = \sim A$$

$$X = \bar{A}$$

$$X = NOT A$$

Expressão: $x = \sim A$

A operação **NOT** é fundamental não apenas por sua funcionalidade, mas também por seu papel na construção de outras portas lógicas. A combinação das portas lógicas básicas, incluindo a porta **NOT**, forma a base para a criação de portas lógicas mais complexas, como a **XOR (ou-exclusivo)** e **NAND (não-e)**. Essa composição hierárquica de portas lógicas é essencial para a implementação de funções lógicas e circuitos digitais mais elaborados.

Os circuitos lógicos, construídos a partir das portas lógicas, desempenham um papel vital em várias áreas da tecnologia moderna. Desde microprocessadores até sistemas de comunicação e automação, circuitos lógicos estão presentes em uma variedade de dispositivos eletrônicos. Por exemplo, microprocessadores são compostos por bilhões de transistores interconectados que implementam funções lógicas complexas por meio da combinação de portas lógicas simples (MARCOVITZ, 2008).

Arquitetura Básica de Computadores (Processadores, Memória, Barramentos, Dispositivos de Entrada/Saída e Armazenamento Secundário)

Arquitetura de von Neumann, ULA, Memória e I/O

A grande maioria dos computadores existentes atualmente segue um modelo proposto pelo matemático naturalizado americano *von Neumann*, por volta de 1940. O modelo proposto pelo matemático americano *von Neumann*, exerceu um impacto profundo no design dos computadores modernos. Este modelo, que continua sendo a base para a maioria dos computadores em uso atualmente, delineia uma abordagem integral para o processamento de informações (KOWALTOWSKI, 1996).

Neste modelo (*von Neumann*), conforme pode ser observado na figura 6 é proposto um processador que executa instruções que estão armazenadas em uma memória de programas. Esse processador é capaz de ler dados provenientes de canais de entrada, transmitir comandos através de canais de saída e, além disso, modificar o conteúdo de uma memória de dados (STALLINGS, 2017).

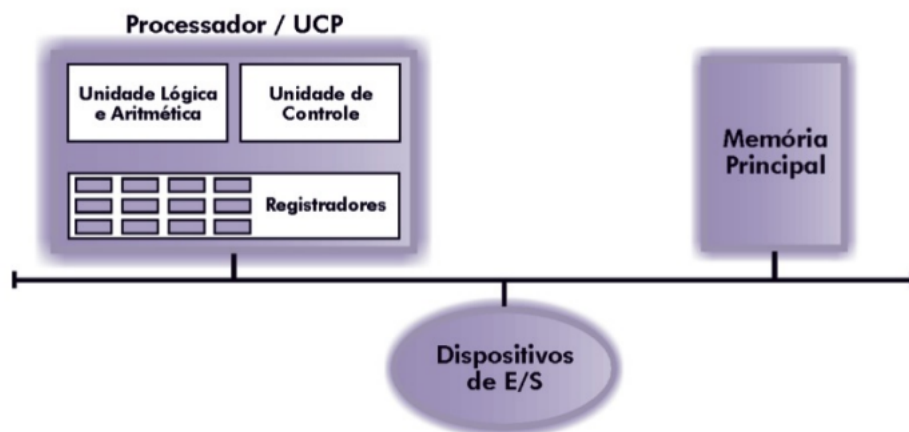


Figura 6: Arquitetura de von Neumann

Fonte: (MACHADO; MAIA, 2004) pg. 23

Com o passar do tempo o modelo proposto por *von Neumann* passou por uma transformação significativa, culminando na arquitetura moderna que conhecemos hoje, mas mantendo sua estrutura essencial. Nesse estágio evolutivo, as antigas memórias de dados e de programa foram integradas em uma única entidade de memória. Esse desenvolvimento é a base para o que chamamos de "estrutura em barramento". Essa estrutura em barramento representa uma via de comunicação compartilhada e de alta velocidade que interconecta diferentes elementos do computador. Essa via comum permite uma troca eficiente e rápida de informações entre os componentes do sistema, aprimorando a eficácia e a velocidade das operações (STALLINGS, 2017).

A estrutura moderna de um computador é composta basicamente dos seguintes componentes que detalharemos a seguir: processadores, memória, barramentos, dispositivos de entrada/saída e armazenamento secundário (HENNESSY; PATTERSON, 2012).

O processador é como o cérebro de um computador. Ele é responsável por mexer e organizar as informações. Imagine que seja a parte mais importante e complexa. O processador é um tipo de *chip*, ou seja, CI que realiza as funções de cálculo e tomadas de decisão do computador.

A memória é como um armário onde o computador guarda coisas. Essas coisas podem ser diferentes: listas de tarefas, números, resultados de contas que está fazendo ou até mesmo informações finais (BRYANT *et al.*, 2003).

A memória constitui um componente físico capaz de armazenar algum tipo de conteúdo para uso no computador. Esse conteúdo pode ser de natureza variada incluindo instruções, dados brutos, bem como resultados originados de fases intermediárias e definitivas do processo de processamento, englobando, desse modo, informações resultantes do tratamento computacional (PATTERSON; HENNESSY, 2013).

Os dispositivos de entrada/saída (*Input/Output*) são responsáveis pela comunicação de entre o computador e o mundo exterior, permitindo que as informações entrem (entrada) ou saiam (saída) do computador. É como uma ponte que conecta o mundo real às operações que o computador pode executar. ****Entrada (*Input*):**** É o processo de enviar informações para o computador. Isso pode ser feito por meio de dispositivos como teclado, mouse, microfone, scanner, etc. Esses dispositivos enviam dados para o computador, que podem ser processados ou armazenados (TANENBAUM, 2016).

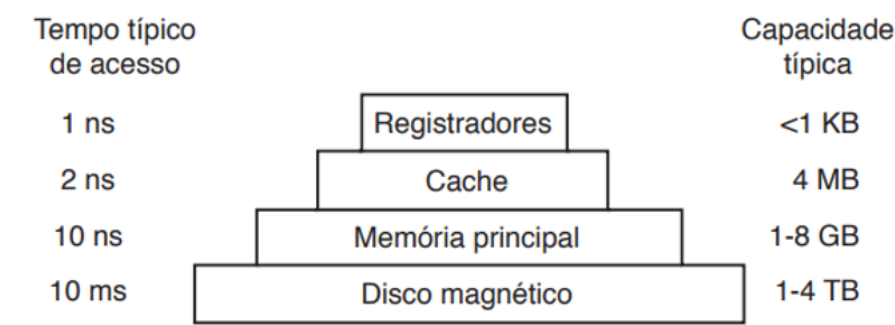


Figura 7: Hierarquia de Memória

Fonte: (TANENBAUM, 2016) pg. 17

Quando um programa é executado, torna-se necessário que ele seja trazido para a memória. É neste momento que programas se transformam em processos em execução. Como sabemos, a hierarquia de memória utilizada em computadores compreende diferentes tipos de memórias, cada uma com características específicas, combinando memórias voláteis e não-voláteis.

Esses tipos incluem memória cache (próxima ao processador), e tem como objetivo acelerar o acesso a dados frequentemente utilizados (ponto de atenção não confundir com registradores que fazem parte da estrutura da CPU), memória principal (RAM), que fornece um espaço maior para armazenamento temporário; e memória secundária (HD - *Hard Disk*) “dispositivos de massa”, e outros dispositivos do tipo I/O (*Input/Output* – Entrada/Saída) que possui capacidade significativamente maior, mas com velocidade de acesso mais lenta (TANENBAUM, 2018) .

Ao sistema operacional é destinada a coordenação e o gerenciamento eficiente desses diversos níveis de memória de forma eficiente. Este serviço é implementado pelo sistema operacional através de um componente específico chamado gerenciador de memória.

Memória Interna e Externa

A memória interna, também conhecida como memória primária, desempenha um papel crucial no contexto do armazenamento transitório de informações que são empregadas durante as operações de processamento. Sua função primaria é possibilitar a execução eficiente dos processos computacionais, ao servir como repositório imediato e temporário para os dados manipulados pelo sistema. Encontra-se dentro desta categoria os registradores, a memória cache e memória principal. Sabe-se que a memória interna desempenha um papel fundamental no funcionamento otimizado dos sistemas de computação.

Ainda dentro deste contexto sabe-se que os registradores são espaços de armazenamento de alta velocidade dentro do próprio processador, permitindo assim o acesso instantâneo a dados frequentemente utilizados (HENNESSY; PATTERSON, 2012).

Já a memória *cache*, funciona como uma área de armazenamento intermediária entre os registradores e a memória principal, visando minimizar os tempos de acesso e maximizar o desempenho do sistema (TANENBAUM, 2016).

Por sua vez a memória principal, também conhecida como RAM (*Random Access Memory*), possui capacidade de armazenar dados e instruções em uma escala maior, facilitando a realização de operações de leitura e escrita pelo processador (HENNESSY; PATTERSON, 2012)

Quando se fala a respeito da memória externa, ou memória secundária, sabe-se que a mesma tem por função a conservação de informações não voláteis em dispositivos periféricos que são acessíveis por meio de controladores de entrada e saída (E/S). Essa categoria de memória é vital para a persistência de dados e informações, além das operações correntes do sistema “leitura e gravação”, uma vez que seu conteúdo é retido mesmo em situações de desligamento (STALLINGS, 2017). São exemplos típicos de meios de armazenamento de memória externa (secundária) os, dispositivos como discos rígidos e CDs-ROM (SILBERSCHATZ *et al.*, 2013).

Em resumo, a compreensão das distinções e funções das memórias interna e externa é essencial para um conhecimento abrangente do funcionamento dos sistemas computacionais. A memória interna, composta por elementos como registradores, cache e memória principal, assegura a eficiência das operações de processamento. Já a memória externa, manifestada em dispositivos periféricos como discos rígidos e CDs-ROM, tem a responsabilidade de manter a persistência dos dados ao longo do tempo (TANENBAUM, 2016).

A memória física, ou seja, a memória RAM é a memória “real” do sistema implementada com CIs. Sendo que a memória física possui áreas reservadas (ex. vetor de interrupções) e endereço físico que acessa posições da memória física. A memória física (RAM) é gerenciada pela Unidade de gerenciamento de memória MMU (*Memory Management Unit*). A MMU mostrada na figura 8 vai prover mecanismos básicos para gerência de memória, proteção de acesso, mapeamento de endereços lógicos em endereços físicos. A MMU costuma estar localizada junto ao processador.

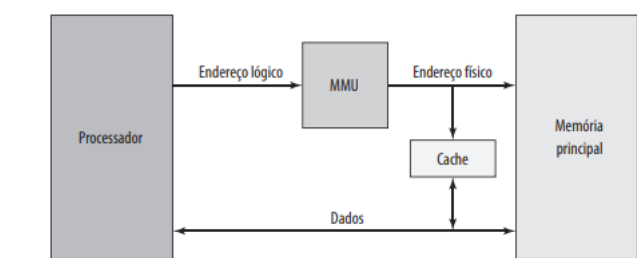


Figura 8: Unidade Gerenciamento de Memória (MMU)

Fonte: (STALLINGS, 2010) pg. 99

O Gerenciador de Memória desempenha funções fundamentais no contexto de sistemas computacionais, visando otimizar a utilização dos recursos de memória disponíveis. Algumas das principais atribuições desse componente incluem (STALLINGS, 2017):

Transferência de Programas entre Memórias sendo uma das tarefas centrais é a movimentação de programas da memória secundária para a memória principal, onde ocorre a execução dos processos.

Minimização das Operações de E/S visto que existe uma diferença de velocidade (latência) entre a memória principal e a memória secundária em relação à memória principal.

Maximização da Utilização de Recursos que busca otimizar a utilização dos recursos de memória, o gerenciador trabalha para manter o maior número possível de processos residentes na memória. Dessa forma, recursos valiosos não ficam ociosos, aumentando a eficiência do sistema.

A aceitação de Novos Programas sem Restrições mesmo quando não há espaço livre na memória, o Gerenciador de Memória permite a aceitação de novos programas. Isso é alcançado através de técnicas como a substituição de processos em memória, assegurando que novas tarefas possam ser executadas sem bloqueios desnecessários.

A execução de Programas Maiores que a Memória Física ao possibilitar a execução de programas que excedem a capacidade da memória física. Para tanto faz-se o uso de técnicas como a paginação ou segmentação, que permitem que partes do programa sejam carregadas na memória conforme necessário.

A proteção de Áreas de Memória por Processo visando evitar interferências indesejadas, o gerenciador implementa mecanismos de proteção que garantem que cada processo tenha acesso apenas às áreas de memória designadas para ele.

E finalizando o compartilhamento de Dados e Informações entre diferentes processos, promovendo a cooperação e a comunicação eficiente entre as partes do sistema. Essas funções do Gerenciador de Memória contribuem para a eficácia e eficiência do funcionamento de sistemas computacionais, permitindo a alocação, o acesso e a proteção de recursos de memória de maneira organizada e coordenada (SILBERSCHATZ *et al.*, 2013).

Conclusões e considerações sobre memória e sua hierarquia

Em se tratando da hierarquia de memórias, existem algumas considerações inerentes e muito importantes quanto a projeto de memória de um sistema computacional que podem ser abstraídas em três interrogações fundamentais (SILBERSCHATZ *et al.*, 2013): qual é a capacidade de armazenamento da memória, qual é a velocidade de acesso oferecida e qual é o custo associado por bit armazenado?

A implementação de sistemas de memória envolve uma ampla gama de tecnologias. Dentro desse espectro de opções tecnológicas, emergem relações cruciais a serem consideradas (TANENBAUM, 2016):

- Um tempo de acesso mais curto tende a aumentar o custo por *bit* armazenado.
- À medida que a capacidade de armazenamento aumenta, a tendência é que o custo por *bit* armazenado diminua.
- A ampliação da capacidade de armazenamento geralmente resulta em um aumento no tempo de acesso.

Essas considerações estruturam a tomada de decisões no projeto de sistemas de memória, uma vez que os projetistas enfrentam compromissos entre velocidade, capacidade e custo. A seleção apropriada de tecnologias de memória e suas características associadas é um fator determinante para alcançar um equilíbrio eficiente entre esses três aspectos críticos.

Interconexão e Comunicação entre Componentes de Computador

Barramentos de Comunicação em Computadores

É necessário que os componentes físicos de um computador (*hardware*) efetuem a troca de informações de modo eficiente para que exista um funcionamento coerente e coordenado das operações. Devido ao isolamento físico desses componentes, torna-se necessário estabelecer um meio de comunicação que possibilite a interconexão entre eles. Tal papel é desempenhado pelos barramentos. Mas o que exatamente é um barramento? Um barramento trata-se de um sistema composto por um conjunto de fios ou condutores operando em paralelo, destinado a viabilizar a transferência de dados e sinais entre dispositivos (TANENBAUM, 2016).

Um barramento também chamado de "bus", trata-se de um trajeto compartilhado através do qual os dados fluem dentro de um computador. Este trajeto atua como um canal de comunicação que facilita as interações entre os diversos elementos presentes no sistema. A dinâmica de um barramento envolve a possibilidade de criar conexões entre dois ou mais dispositivos, promovendo a troca de informações de forma eficaz. Existem diversos tipos de barramentos com diversas funções em um computador, dentre estas três funções distintas emergem como essenciais:

Barramento de Dados: É responsável pela troca bidirecional de informações entre os componentes do sistema. Ele desempenha um papel crucial no envio e recebimento de dados, permitindo a fluência de informações dentro do ambiente computacional.

Barramento de Endereços: É responsável pela indicação da localização na memória onde processos ou dados devem ser acessados ou armazenados. Ao estabelecer uma rota de endereçamento, viabiliza-se a organização ordenada dos dados no sistema.

Barramento de Controle: Desempenha um papel de controle das demais funções dos barramentos. O barramento de controle possui a capacidade de governar e coordenar a operação dos outros barramentos, possibilitando a limitação ou expansão das operações conforme as demandas do sistema (SILBERSCHATZ *et al.*, 2013).

Compreender estas três funções é vital no entendimento da arquitetura computacional, uma vez que os barramentos visualizados na figura 9, constituem o arcabouço sobre o qual as operações e interações dos componentes são construídas. Ao alinhar a troca de dados, a indicação de endereços e a regulação das operações, os barramentos constituem a espinha dorsal que viabiliza o funcionamento coeso dos sistemas computacionais (PATTERSON; HENNESSY, 2013).

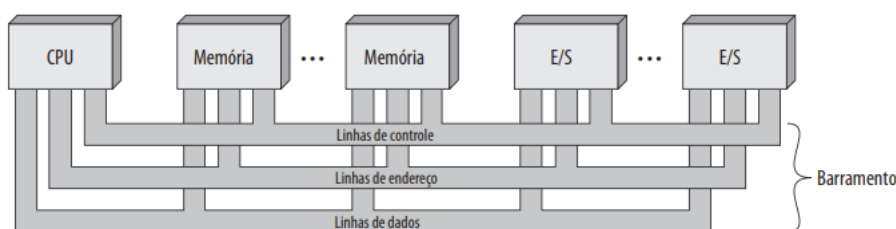


Figura 9: Barramentos

Fonte: (STALLINGS, 2010) pg. 69

Os barramentos apresentam traços distintivos que moldam suas operações e eficiência, dentre estes traços destaca-se. Os tipos de transferência dos barramentos, a seleção de métodos de arbitragem, a determinação da sincronia e a definição da largura são fatores essenciais que influenciam a dinâmica da comunicação interna (PATTERSON; HENNESSY, 2013).

Tipo de Transferência: Os barramentos podem ser dedicados, com uma função específica, ou multiplexados, nos quais uma mesma via é utilizada para transmitir tanto endereços quanto dados, resultando em custo menor, mas com um tempo de transferência maior.

Métodos de Arbitragem: A arbitragem pode ser centralizada, envolvendo um árbitro que controla a disputa pelo barramento, ou distribuída, onde cada dispositivo tem sua própria linha de requisição com um nível de prioridade.

Sincronia: Os barramentos podem ser síncronos, com operações coordenadas por um relógio mestre, ou assíncronos, permitindo operações de duração variável, sem a restrição de ciclos de relógio.

Largura: Os barramentos podem ser em bloco, mais eficientes para transferir um conjunto de dados, ou usar o método de leitura-modificar-escrever, necessário em sistemas multiprocessadores para assegurar o acesso exclusivo a dados compartilhados.

Além dos traços supracitados (tipo de transferência, métodos de arbitragem, sincronia e largura), é importante entender que a arquitetura de barramentos se desdobra em várias categorias, cada uma com uma função específica no sistema. Estas categorias são: O barramento do processador, o barramento de cache, o barramento de Memória, e o barramento de Entrada e Saída (E/S).

Barramento do Processador: é o barramento responsável por conectar o processador ao chipset adjacente, permitindo a troca de informações com outros sistemas.

Barramento de Cache: é o barramento dedicado ao acesso à memória cache, sendo fundamental para processadores que possuem cache externa à CPU.

Barramento de Memória: É o barramento que conecta a memória ao chipset e ao processador, gerenciado por um controlador específico ou compartilhado com o barramento de processador.

Barramento de Entrada e Saída (E/S): É o barramento que liga periféricos de alto desempenho à memória, chipset e processador, incluindo expansões internas e externas.

Concluindo os barramentos operam por meio de instruções e sinais específicos que regulam as operações e a comunicação entre dispositivos. Instruções como escrita e leitura de memória, escrita e leitura de E/S, bem como sinais de transferência aceita (*ACK* “comunica o recebimento de instrução”) e outros, constituem a base para o funcionamento coordenado e eficiente dos barramentos. A compreensão das características e funções dos barramentos é essencial para a concepção, o *design* e a operação eficiente dos sistemas computacionais, bem como para uma apreciação aprofundada da complexa interconexão dos componentes envolvidos (TANENBAUM, 2018).

Linguagem de Montagem e Programação de Baixo Nível

A linguagem de montagem é uma forma de programação de baixo nível que permite aos programadores interagirem diretamente com a arquitetura do computador. A linguagem de montagem é composta por elementos básicos que permitem a escrita de programas para a arquitetura do computador. As instruções na linguagem de montagem são traduzidas diretamente em código de máquina.

A linguagem de montagem e a programação de baixo nível representam um nível de abstração fundamental na interação entre o *software* e o *hardware* de um computador. A estrutura da linguagem de montagem e os princípios subjacentes à programação de baixo nível são de importância crítica para os desenvolvedores de *software* que desejam maximizar o desempenho e a eficiência de seus programas (PETERSON, 2014) .

A linguagem de montagem é uma representação simbólica das instruções de máquina executadas pelo processador. Ela possui uma estrutura que permite aos programadores interagirem diretamente com a arquitetura do computador.

A estrutura da linguagem de montagem é construída em torno de componentes fundamentais, como mnemônicos, operandos e comentários. Os mnemônicos representam as operações a serem executadas, enquanto os operandos especificam os dados envolvidos na operação. Comentários fornecem informações auxiliares para o programador e são ignorados pelo montador. As instruções de montagem são representações simbólicas das operações que o processador deve executar. Cada instrução tem um formato específico, geralmente composto por um campo de código de operação e campos de operandos. O código de operação indica a operação a ser realizada, e os operandos especificam os dados ou endereços envolvidos na operação (BRYANT *et al.*, 2003).

Já no que se refere a programação de baixo nível envolve a escrita de código que se alinha mais diretamente com a arquitetura física do computador. Isso inclui aspectos como acesso direto à memória e manipulação eficiente de dados. Alguns aspectos cruciais da programação de baixo nível são:

O acesso à memória sendo que dentro deste contexto a manipulação de memória desempenha um papel fundamental na programação de baixo nível. Os programadores podem ler e escrever diretamente na memória, o que requer um entendimento sólido dos endereços de memória e da hierarquia de cache. O acesso eficiente à memória é essencial para otimizar o desempenho do *software*.

Os modos de endereçamento que determinam como os operandos são especificados em uma instrução. Isso inclui endereçamento direto, indireto, imediato e relativo, entre outros. A escolha do modo de endereçamento afeta a flexibilidade e a eficiência do código de montagem.

E para finalizar a manipulação de dados visto que a programação de baixo nível envolve a manipulação direta de dados em memória. Isso inclui operações como carregar e armazenar dados, conversão de tipos e operações aritméticas. O programador deve estar ciente das representações de dados na memória e das operações que podem ser executadas de forma eficiente.

Concluindo a compreensão profunda da estrutura da linguagem de montagem e da programação de baixo nível é essencial para os programadores que buscam otimizar seus programas e compreender a interação entre o *software* e o *hardware*. Ao explorar os elementos fundamentais da linguagem de montagem e da programação de baixo nível, os desenvolvedores podem criar *software* eficiente e maximizar o desempenho em uma variedade de ambientes de computação (IRVINE, 2011).

Arquiteturas de Processadores

Existem diversas abordagens arquiteturais sobre os processadores. Compreender estas arquiteturas é crucial para compreender as estruturas subjacentes que conduzem o funcionamento dos sistemas computacionais. As arquiteturas de processadores, são bastante variadas entretanto, elas compartilham entre si o propósito essencial de harmonizar e administrar as operações e a execução de tarefas nos níveis mais fundamentais da computação. Sendo assim, o aprofundamento no estudo das arquiteturas de processadores é vital para a compreensão de como os sistemas computacionais realizam suas funções, bem como para avaliar as implicações de design e desempenho das diferentes abordagens arquiteturais (HARRIS; HARRIS, 2015).

Em se tratando das arquiteturas de processadores, duas abordagens distintas – RISC (*Reduced Instruction Set Computing* - conjunto reduzido de instruções) e CISC (*Complex Instruction Set Computing* - conjunto complexo de instruções) se destacam como alternativas para projetar a funcionalidade e eficiência dos processadores. Sendo assim analisar e comparar essas arquiteturas vai oferecer *insights* valiosos e importantes sobre suas características, vantagens e desvantagens.

Comparação entre Arquiteturas de Processadores: RISC e CISC

A arquitetura CISC, presente nos processadores *Intel Corporation* e *AMD Inc.*, é caracterizada por um conjunto complexo de instruções. Apesar de oferecer suporte a muitas instruções, a execução dessas instruções costuma ser mais lenta devida a seu tamanho e complexidade. Visto que esses processadores incorporam uma microprogramação, que consiste em um conjunto de códigos de instruções gravados no próprio processador. Essa abordagem permite que o processador receba e execute as instruções dos programas, fazendo uso das instruções presentes na sua microprogramação. Consequentemente isso resulta em uma redução no tamanho do código executável, uma vez que o código comum a vários programas é condensado em uma única instrução.

Na arquitetura CISC, operações como " $a = a + b$ " são descritas de maneira resumida, por exemplo, como "**add a, b**". Esses processadores podem operar com dois operandos em uma única instrução, sendo um deles fonte e destino (acumulador "registros temporários"). Eles também permitem um ou mais operandos em memória para a realização das instruções (PATTERSON; HENNESSY, 2013).

Por sua vez a arquitetura RISC caracteriza-se pela simplicidade e velocidade uma vez que a abordagem da arquitetura RISC, segue uma filosofia oposta a arquitetura CISC. Sendo assim o foco principal da arquitetura RISC é a simplicidade do conjunto de instruções, o que resulta em uma execução mais rápida das instruções combinadas. Esta arquitetura se destacou muito no início da década de 80, enquanto a tendência era a construção de chips com conjuntos de instruções cada vez mais complexos e alguns fabricantes começaram a optar por adotar a abordagem RISC. Processadores RISC suportam menos instruções, o que resulta em uma execução mais veloz das instruções combinadas. Esses *chips* são mais simples e, conseqüentemente, mais acessíveis economicamente, uma vez que possuem menos circuitos internos. Além disso, têm a capacidade de operar em frequências mais altas.

Na arquitetura RISC é possível realizar operações como por exemplo, " $a = b + c$ " como "**add a, b, c**", especificando três operandos para uma única instrução, mas apenas se esses operandos forem "registros", devido à natureza reduzida das instruções RISC (STALLINGS, 2017) .

A tabela 6 a seguir apresenta de forma resumida a principal diferença entre as arquiteturas RISC e CISC.

Tabela 9: Diferenças entre as arquiteturas RISC e CISC

RISC	CISC
Conjunto de instruções de origem simples e de tamanho fixo reduzido	Conjunto de instruções de origem complexas e de tamanho variável extenso
Decodificação simplificada (por tabela – que será consultada na memória)	Decodificação complexa (microcódigo – está contido no próprio <i>chip</i>)
Execução regular, interpretada pelo próprio programa	Cada instrução executa à sua maneira, pelo <i>hardware</i>
Instruções requerem o mesmo número de ciclos de <i>clock</i> para executar	Grande variação no número de ciclos de <i>clock</i> por instrução
Possibilita o uso de <i>pipeline</i> (intenso)	Extremamente difícil/impossível o uso de <i>pipeline</i> (ameno)
Apenas algumas operações (<i>load/store</i>) em memória	Qualquer instrução pode referenciar a memória
Poucas instruções e modos de endereçamento	Várias instruções e modos de endereçamento
Múltiplos conjuntos de registradores	Conjunto de registradores únicos (operadores aritméticos, ponteiros/apontadores para a memória, memória segmentada)
Não possuem instruções para multiplicação ou divisão	Possuem instruções para multiplicação ou divisão

Fonte: Adaptado de (STALLINGS, 2017)

Escolher entre as arquiteturas RISC e CISC implica uma análise cuidadosa das necessidades do sistema e das prioridades em relação à eficiência de execução e complexidade. As características de cada uma dessas abordagens impactam diretamente na funcionalidade e no desempenho dos processadores, o que torna essa escolha um aspecto fundamental no design de sistemas computacionais, entretanto, hoje em dia a maioria dos processadores possuem uma arquitetura híbrida.

Os processadores que utilizam esta abordagem híbrida são essencialmente processadores CISC, mas que incorporam muitos recursos encontrados nos processadores RISC (ou vice-versa). Apesar de por questões de Marketing, muitos fabricantes ainda venderem seus *chips*, como sendo "Processadores RISC", não existe praticamente nenhum processador atualmente que siga estritamente uma das duas filosofias (STALLINGS, 2017).

Microcontroladores e Microcomputadores para projetos diversos

Noções introdutórias sobre *Raspberry Pi* e *Arduino*

Nos últimos anos, a crescente disponibilidade de plataformas de *hardware* acessíveis e versáteis tem proporcionado uma oportunidade emocionante para entusiastas e estudantes explorarem os campos da eletrônica e da programação de maneira prática. Duas das plataformas mais notáveis nesse contexto são o *Raspberry Pi* e o *Arduino*, os quais ambos tem sido largamente utilizados em atividades ligadas a *Cultura Maker*. A seguir conheceremos um pouco sobre estas duas plataformas

Primeiramente falaremos sobre o *Raspberry Pi* que é basicamente um microcomputador de baixo custo, desenvolvido com a finalidade de promover a educação em ciência da computação e a acessibilidade tecnológica. O *Raspberry Pi* possui uma arquitetura semelhante à de um computador pessoal oferecendo uma série de recursos tais como: Navegação na Internet, reprodução de mídia, programação e criação de projetos eletrônicos personalizados. Sua versatilidade o torna um instrumento valioso para educadores e estudantes que desejam desenvolver habilidades em programação, eletrônica e automação (BERGSTRÖM, 2023).

Imagine um computador que cabe na palma da mão. Isso é o *Raspberry Pi*! É possível por exemplo conectá-lo a uma TV ou monitor e usá-lo para navegar na internet, assistir a vídeos e até mesmo programar jogos. É possível ainda utilizar o *Raspberry Pi* para criar projetos eletrônicos como por exemplo sistemas de

automação residencial, monitoramento do clima, dentre outros. O *Raspberry Pi* é um dispositivo extremamente versátil e consequentemente empolgante para aprender e explorar a eletrônica e a programação.

Já o *Arduino* pode ser chamado de o “Microcontrolador para Criação Interativa” visto que é uma plataforma baseada em microcontroladores que se destaca por sua facilidade de uso e flexibilidade. Projetado originalmente para prototipagem rápida e experimentação, o *Arduino* permite que os usuários criem dispositivos interativos por meio da programação de componentes eletrônicos, como sensores, LEDs e motores. Possui uma comunidade bastante ativa e os inúmeros recursos *online* oferecem uma vasta gama de projetos, exemplos de código e tutoriais, tornando-o uma escolha popular para iniciantes e entusiastas da eletrônica (BARRETT, 2020).

Com o *Arduino* é possível criar e controlar dispositivos eletrônicos interativos. É possível por exemplo acender LEDs, mover motores, medir temperaturas e até mesmo criar um teclado musical dentre outros. O *Arduino* é perfeito para projetos criativos, experimentação e aprendizado de eletrônica. Além disso o possui uma linguagem de programação simples e muitos recursos *online* para ajudar no aprendizado e no processo criativo.

Ambas as ferramentas a saber, o *Raspberry Pi* e o *Arduino* oferecem abordagens distintas, entretanto complementares, para a exploração do mundo da eletrônica e da programação. Visto que enquanto o *Raspberry Pi* visa proporcionar uma experiência mais próxima à computação tradicional, o *Arduino* por sua vez vai focar em uma experiência de interatividade e experimentação com componentes eletrônicos. Sendo assim ambas as plataformas apresentam enorme potencial para educadores, pesquisadores e aprendizes, fornecendo oportunidades para aquisição de habilidades técnicas e criativas.

Dica

Para conhecer em detalhes o *Raspberry Pi* e o *Arduino*, acesse o [links](https://www.arduino.cc/) dos sites oficiais dos mesmos.

Arduino: <<https://www.arduino.cc/>> - *Raspberry Pi*: <<https://www.raspberrypi.com/>>

Sistemas de Entrada/Saída e Interrupções

Sistema de Entrada/Saída (I/O)

A administração eficiente e eficaz de dispositivos de Entrada/Saída (I/O) é uma das tarefas essenciais e mais complexas executadas pelos Sistemas Operacionais (SOs). Os módulos de entrada e saída desempenham um papel primordial nos sistemas computacionais estabelecendo a interligação entre os barramentos internos do sistema, viabilizando assim a comunicação com o mundo exterior. A função central dos módulos de I/O é facilitar a comunicação entre os periféricos controlados e os barramentos do sistema, proporcionando uma interface essencial para a interação externa (TANENBAUM, 2018).

Como pode ser visualizado na figura 10, diversos elementos impulsionam a necessidade de intercalar módulos de I/O, entre os periféricos e o barramento do sistema. Sabe-se que existe uma imensa variedade de periféricos, e que cada um deles possui a sua própria lógica operacional. O que torna consequentemente inviável que um único componente, como o processador, possa controlar todos esses dispositivos (SILBERSCHATZ *et al.*, 2013).

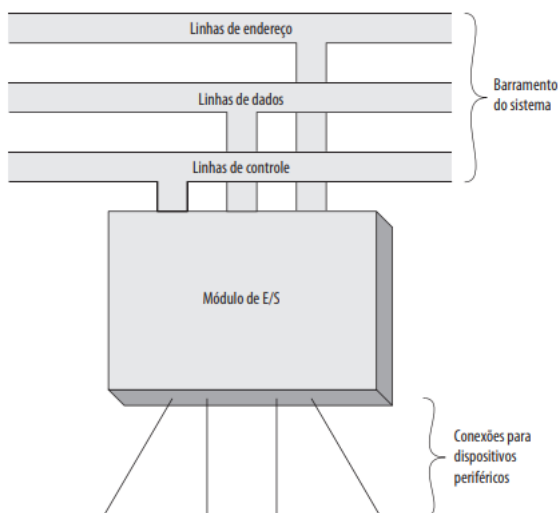


Figura 10: Módulo de Input/Output

Fonte: (STALLINGS, 2010) pg.177

A disparidade nas taxas de transferência de dados entre dispositivos periféricos e a Unidade Central de Processamento (CPU) também vai motivar a necessidade de uma camada intermediária. A presença de diferentes formatos de dados e tamanhos de palavras utilizados pelos periféricos, formatos estes que são diferentes dos padrões adotados pelo computador, destaca a importância dos módulos de I/O para facilitar e estabelecer uma conexão adequada com o processador e a memória (SILBERSCHATZ *et al.*, 2013).

Quanto aos dispositivos periféricos, externos ao sistema, a conexão vai ser intermediada através de módulos de entrada e saída, que estabelecem um canal para a troca de dados além de transmitirem informações de controle e *status* dos periféricos ou das operações realizadas. Os sinais de controle têm a tarefa de indicar o tipo de operação (*read/write* – leitura/escrita) ou ações de controle específicas. Por sua vez os sinais de estado refletem a condição (*status*) do dispositivo, como "*pronto*" ou "*não pronto*" (TANENBAUM, 2018).

Os dados por sua vez são constituídos por *bits* que são transmitidos ou recebidos pelos módulos de entrada e saída (I/O). Dentro desta interação o *buffer* por sua vez, assume um papel essencial, visto que o *buffer* se trata de uma área de armazenamento temporário que facilita e simplifica o gerenciamento de fluxos de dados variáveis. Além disso, os transdutores como vistos na figura 11, desempenham a tarefa de converter dados codificados em sinais elétricos em outras formas de energia (STALLINGS, 2010).

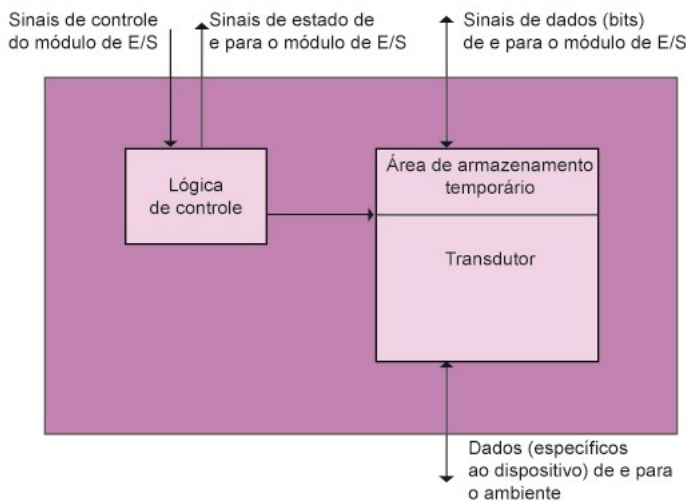


Figura 11: Módulo de *Input/Output*

Fonte: (STALLINGS, 2010) pg.178

Por sua vez os dispositivos externos podem ser organizados em diferentes categorias conforme a natureza de sua interação: com o usuário (teclado, vídeo, impressora), com a máquina (discos magnéticos, sensores) ou com dispositivos remotos (modems, placas de rede). Os módulos de entrada/saída (I/O) executam múltiplas funções essenciais, incluindo o controle, a sincronização e a comunicação com o processador e dispositivos, armazenamento temporário de dados e detecção de falhas (SILBERSCHATZ *et al.*, 2013).

A integração bem-sucedida das funções de controle e temporização é de vital importância, uma vez que recursos compartilhados como por exemplo o barramento e a memória são utilizados para diversas atividades, incluindo as operações de entrada e saída (I/O). O controle e a temporização desempenham um papel essencial na regulação do fluxo de dados entre o sistema computacional e o ambiente externo.

O processo de comunicação entre dispositivos e processador envolve fases como o questionamento do estado do módulo de entrada/saída (I/O), sendo que o resultado deste questionamento vai ser devolvido ao processador, através da solicitação de transferência de dados pelo processador, tendo finalmente a sua efetiva transferência de dados ocorrendo entre o dispositivo e o processador (TANENBAUM, 2018).

Quando se inicia a execução de instruções de entrada/saída (I/O), o processador gera um endereço que vai apontar o módulo de entrada/saída (I/O). Esse endereço é transmitido através do barramento, acompanhada da operação requisitada. Para que isto aconteça diferentes tipos de comandos de entrada/saída (I/O) são utilizados, abrangendo comandos de controle (para ativar um periférico e executar uma ação), comandos de teste (para verificar as condições do estado do dispositivo), comandos de leitura (solicitação de leitura de dados do periférico) e comandos de gravação (para armazenar dados no periférico) (STALLINGS, 2017).

Os tipos de comandos de E/S são:

Controle: Ativa um periférico e indica uma ação, por exemplo: *eject/dev/cdrom* (Linux).

Teste: Verifica as condições de estado do dispositivo associado ao módulo de E/S, por exemplo: *ready* ou *not-ready*.

Leitura: Solicita a leitura de itens de dados do periférico. O módulo de E/S os armazena no registrador de dados interno até que o processador solicite a transferência deles para o barramento de dados.

Gravação: A CPU ordena ao módulo de E/S que pegue o dado do barramento de dados e o armazene no periférico.

A gestão de dispositivos de entrada/saída (I/O) é organizada em camadas, onde as estruturas de níveis inferiores ocultam as particularidades dos dispositivos físicos das camadas superiores, isolando, assim, as aplicações dos usuários. Esse isolamento permite que as aplicações dos usuários operem sem a necessidade de conhecimento detalhado sobre a arquitetura de *hardware* utilizada nas operações de entrada/saída (TANENBAUM, 2016). Nesse contexto, três abordagens fundamentais surgem para a implementação dessas operações a saber:

1. Por programa onde o processador estabelece uma comunicação sincronizada com o periférico, iniciando a transferência de dados. Contudo, o processador permanece constantemente ocupado, monitorando o estado do periférico até o desfecho da operação de E/S.
2. Por interrupção onde uma vez iniciada a transferência de dados, o processador é liberado para executar outras tarefas. O sistema operacional, em intervalos específicos, verifica o estado dos periféricos para confirmar a finalização da operação de E/S. Esse método introduz um grau de paralelismo, permitindo a execução de programas enquanto as operações de leitura/gravação de periféricos transcorrem, viabilizando os pioneiros sistemas multiprogramáveis.
3. Por Acesso Direto à Memória DMA (*Direct Memory Access*), o DMA vai possibilitar a transferência de um bloco de dados entre a memória e os dispositivos de E/S sem a intervenção constante do processador, limitando sua intervenção somente no início (solicitação) e no término da transferência (notificação de conclusão – *ACK "Acknowledgement Signal"*).

A tecnologia DMA utiliza uma área de memória denominada "*buffer* de E/S". O DMA é uma técnica que representa uma alternativa mais eficiente para operações de E/S, onde a transferência de dados entre o módulo de E/S e a memória principal é realizada diretamente, sem a necessidade de envolvimento constante do processador. O controlador DMA possui a capacidade de emular as funções do processador, simplificando a transferência de dados entre a memória e os dispositivos por meio do barramento do sistema. Essa competição pelo uso do barramento com a CPU é evidente (SILBERSCHATZ *et al.*, 2013).

A técnica de DMA acontece da seguinte forma: Quando a CPU deseja ler ou gravar (*load/write*) um conjunto de dados em um periférico, ela segue um protocolo específico. Por meio do barramento de controle, a CPU indica ao DMA sobre o tipo de operação por meio do barramento de controle, fornecendo o endereço do módulo de E/S correspondente, o endereço de memória para a transferência de dados e a quantidade de palavras a serem lidas ou escritas, através de linhas de dados.

O DMA assume a gestão da operação de I/O. Por exemplo, em uma operação de leitura de um arquivo em disco, o DMA supervisiona a operação, armazenando os dados lidos diretamente na memória principal, conforme indicado pela CPU. Ao término da transferência, o DMA gera um sinal de interrupção para a CPU, indicando o término da operação. A CPU acessa os dados diretamente da memória, minimizando o tempo de acesso à controladora do disco e, consequentemente otimizando a eficiência global do sistema computacional. Dessa forma, o processador se envolve apenas no início e no final do processo (TANENBAUM, 2016).

Mecanismo de Interrupção

As interrupções são mecanismos fundamentais para lidar com eventos assíncronos que demandam a execução de instruções específicas e dedicadas para o tratamento de eventos específicos. Visto que durante a execução de um programa, situações imprevistas podem acontecer resultando em uma mudança abrupta no

fluxo normal da operação. Esses incidentes são chamados de interrupções ou exceções. Uma interrupção pode ser desencadeada por notificações provenientes de dispositivos de *hardware* externos ao ambiente de memória/processador.

As interrupções desempenham um papel essencial na implementação da concorrência em sistemas computacionais, sendo que o princípio de concorrência é a base de sistemas multiprogramáveis/multitarefa. É importante observar que uma interrupção sempre é originada por um evento externo ao programa em execução, independente da instrução atual (MACHADO; MAIA, 2004).

Ao término de cada instrução um procedimento específico é desencadeado no tratamento de interrupções conforme pode ser observado na figura 12. Neste procedimento a Unidade de Controle monitora a ocorrência de qualquer tipo de interrupção. Nesse momento, o programa em execução é interrompido e direcionado para uma rotina de tratamento no sistema, responsável por gerenciar o evento ocorrido (MACHADO; MAIA, 2004).



Figura 12: Rotina de Tratamento de Interrupção.

Fonte: (MACHADO; MAIA, 2004) pg. 42

O tratamento de uma interrupção consiste em uma ação assíncrona que é executada em paralelo ao programa principal. Sendo que cada tipo de interrupção aciona uma rotina de tratamento específica, desviando o fluxo do programa para lidar com a situação. Intervenções podem ser originadas tanto de ações do usuário, como a entrada de dados pelo console ou teclado, necessitando de rotinas de tratamento personalizadas, quanto pelo *hardware*, como operações de E/S, que já possuem rotinas de interrupção predefinidas. Além disso, o sistema operacional também pode gerar interrupções, por exemplo, ao final de um determinado período de execução do programa, com rotinas de tratamento específicas associadas (MACHADO; MAIA, 2004).

É importante salientar que interrupções são sempre tratadas pelo SO. É importante notar ainda que o tratamento de interrupção não é necessariamente parte do programa principal. Essas intervenções podem ocorrer tanto via *software* quanto via *hardware*. No caso de interrupções por *software*, o programa deve constantemente verificar a ocorrência de uma condição para que a demande a execução de uma tarefa assíncrona. Essa verificação ocorre dentro do programa principal, e é de responsabilidade da verificação do programador. Já no caso de interrupções por *hardware*, a CPU é notificada por um controlador de interrupção assim que a condição predefinida é satisfeita. Aqui, o programador deve se concentrar na programação do tratamento da interrupção, sem a necessidade de se preocupar com o momento em que ela ocorrerá (SILBERSCHATZ *et al.*, 2013).

Dica

Como você verá ou até mesmo já viu em outros cursos do Programa de Residência em TIC BRISA-Softex. Com a linguagem *Python* é possível realizar tarefas das mais diversas e complexas, o mesmo ocorre com os assuntos relacionados a este curso de Organização de Computadores.

Vamos ver então conhecer algumas bibliotecas em *Python* que interagem com dispositivos dos computadores e Sistemas Operacionais:

Vejamos alguns exemplos de interação com o SO e/ou o *hardware*. Para começar, você deverá fazer o *download* da biblioteca que deseja/necessita, então instalá-la e importá-la para poder utilizá-la na linguagem.

Supondo que o arquivo de instalação da biblioteca “*os*” já foi baixado, você deverá então instalá-lo. Para tal, abra um terminal e utilize o gerenciador de pacotes “*pip*” executando o seguinte comando:

```
1. pip install os // instala a biblioteca "os"
```

Agora para poder utilizar as funções da biblioteca “*os*” e executá-las, você deverá importá-la:

```
2. import os // importa a biblioteca "os"
```

Feito isso você já poderá executar alguns comandos, como demonstrado a seguir:

```
3. os.listdir() // cria uma pasta/diretório de nome "Curso de OC"
```

```
4. os.mkdir("Curso de OC") // exibe o conteúdo da pasta/diretório padrão
```

```
5. os.path.abspath('.') // retorna o caminho absoluto para a pasta/diretório
```

```
6. os.path.abspath('Curso de OC') // retorna o caminho absoluto para a pasta/diretório path
// da pasta anteriormente criada "Curso de OC"
```

Algumas outras funções e suas bibliotecas

Função	Biblioteca
SO em uso, processador	<i>platform</i>
Quantidade de memória, tempo de CPU	<i>psutil</i>
Data da BIOS, execução de processos	<i>pywin32</i> para Windows e <i>dmidecode</i> para Linux
Hora e Data: Comp./Server	<i>datetime</i>
Velocidade de conexão	<i>speedtest-cli</i>

Para conhecer muitas outras bibliotecas e poder explorar em profundidade suas funções, não deixe de acessar as páginas de documentação oficial da linguagem *Python*, através do *link*:

<<https://docs.python.org/pt-br/3/library/>>

Você pode também pesquisar a página PyPi, a qual é um excelente repositório de *software* para linguagem, e está disponível através do *link*:

<<https://pypi.org/>>

Até breve . . .

À medida que chegamos ao final deste curso de arquitetura e Organização de Computadores, é importante que você reflita sobre a jornada que percorremos juntos através dos intrincados circuitos e sistemas que formam o coração da tecnologia moderna. Desde a fascinante história da computação até os complexos conceitos de linguagem de montagem e arquiteturas de processadores, exploramos um mundo de conhecimento que sustenta a sociedade digital em que vivemos hoje.

No início, mergulhamos na história da computação moderna, compreendendo como as ideias inovadoras de pioneiros como *Alan Turing* e *John von Neumann* moldaram o curso da tecnologia. Avançando, desvendamos os sistemas numéricos e a aritmética binária que subjazem a linguagem dos computadores, revelando como esses conceitos fundamentais continuam a ser a base de toda a computação.

Aprofundando-nos na lógica booleana e nos circuitos digitais, conhecemos segredos dos operadores lógicos e portas que formam os alicerces da lógica computacional. Exploramos a arquitetura básica dos computadores, compreendendo os processadores, memória e dispositivos de entrada/saída que compõem o funcionamento interno dessas máquinas surpreendentes.

Nossa jornada nos levou a mergulhar na organização e hierarquia da memória, onde descobrimos como as diferentes camadas de armazenamento trabalham em harmonia para oferecer eficiência e velocidade. Discutimos as nuances da interconexão e comunicação entre os componentes do computador, destacando a importância dos barramentos na transmissão de dados cruciais.

Mas esta não é uma despedida, é sim um convite para que você continue explorando e buscando mais conhecimento. Ao compreender as arquiteturas de processadores, as diferenças entre RISC e CISC e as possibilidades emocionantes do *Raspberry Pi* e *Arduino*, você está se posicionando para abraçar um mundo de criação e inovação tecnológica.

Portanto, à medida que fechamos este curso, lembre-se de que a arquitetura e organização de computadores é muito mais do que uma disciplina acadêmica; é o alicerce sobre o qual se constrói a revolução digital. Que esta jornada o(a) inspire a continuar buscando novos horizontes, a mergulhar mais fundo na essência da tecnologia e a se tornar um(a) verdadeiro(a) arquiteto(a) do futuro digital. O conhecimento adquirido aqui é apenas o começo de uma emocionante jornada de descoberta e realização.

Siga sempre em busca de novos conhecimentos!

Tags do conteúdo

História da Computação Moderna Gerações de Computadores Sistemas Numéricos, Notação Posicional, Sistema Decimal, Sistema Binario, Sistema Octal, Sistema Hexadecimal Lógica Booleana, Circuitos Digitais Aritmética Binária Hierarquia de Memória Arquitetura Básica de Computadores, Processadores, Memória, Barramentos, Dispositivos de Entrada/Saída, Armazenamento Secundário Arquitetura de von Newmann Sistemas de Entrada/Saída, I/O, Interrupções, DMA, ULA, UC, UCP, RAM PC, RISC, CISC

Referências

ADAMSON, D. **Blaise Pascal: mathematician, physicist and thinker about God**. [S. l.]: Springer, 1994.

ALMEIDA, J. M. F. **A gênese da máquina da Sociedade da Informação: Baby Machine, o protótipo dos actuais computadores**. [S. l.], 2000. 1º Congresso Luso-Brasileiro de Ciência e da Técnica.

BARRETT, S. F. **Arduino I: getting started**. [S. l.]: Morgan & Claypool Publishers, 2020.

BERGSTRÖM, P. **A Guide to Raspberry Pi and Microsoft Azure (IoT)**. [S. l.], 2023.

BRAUDEL, F. **The Mediterranean and the Mediterranean World in the Age of Philip II: Volume II**. [S. l.]: University of California Press, v. 2. - 1995.

BRYANT, R. E.; DAVID RICHARD, O.; DAVID RICHARD, O. **Computer systems: a programmer's perspective**. [S. l.]: Prentice Hall Upper Saddle River, v. 2 - 2003.

CAMPBELL-KELLY, M. *et al.* **Computer: A history of the information machine**. [S. l.]: Taylor & Francis, 2023.

CERUZZI, Paul E. **A history of modern computing**. MIT press, 2003.

COPELAND, B. J. **Colossus: its origins and originators**. IEEE Annals of the History of Computing, [S. l.], v. 26, n. 4, p. 38-45, 2004.

CREASE, R. P.; MANN, C. C. **The Second Creation: makers of the revolution in twentieth-century physics**. [S. l.]: Rutgers University Press, 1996.

DUNCAN, P. **The Big Switch: Rewiring the World, from Edison to Google**. WW Norton and Co., New York, 2009.

DRUCKER, Peter. **O futuro já chegou**. Revista Exame, v. 22, n. 03, 2000.

EQUIPE EDITORIAL DE CONCEITO.DE. **Computação - O que é, conceito e definição**. CONCEITO.DE, 09 de 2020. Disponível em: <<https://conceito.de/computacao>>

ESSINGER, J. **Ada's algorithm: How Lord Byron's daughter Ada Lovelace launched the digital age**. [S. l.]: Melville House, 2014.

GUGIK, Gabriel. **A história dos computadores e da computação**. TecMundo, Curitiba, 2009.

HADAMARD, J. **An essay on the psychology of invention in the mathematical field**. [S. l.]: Courier Corporation, 1954.

HARRIS, S.; HARRIS, D. **Digital design and computer architecture**. [S. l.]: Morgan Kaufmann, 2015.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Organization and Design: The Hardware/Software Interface, Waltham**. [S. l.]: USA: Morgan Kaufmann, 2012.

HYMAN, A. **Charles Babbage: Pioneer of the computer**. [S. l.]: Princeton University Press, 1985.

IFRAH, G. **The universal history of numbers**. [S. l.]: Harvill London, 2000.

INCERTI, T. G. V.; CASAGRANDE, L. S. **Elas fizeram parte da história da ciência e da tecnologia e são inventoras sim!** Cadernos de Gênero e Tecnologia, [S. l.], v. 11, n. 37, p. 5-26, 2018.

IRVINE, K. R. **Assembly language for x86 processors**. [S. l.]: Pearson Education, Inc, 2011.

KOWALTOWSKI, T. **von Neumann: suas contribuições à Computação**. Estudos Avançados, [S. l.], v. 10, p. 237-260, 1996.

MACHADO, F. B.; MAIA, L. P. **Arquitetura de sistemas operacionais**. [S. l.]: LTC, v. 4 - 2004.

MARCOVITZ, A. B. **Introduction to logic and computer design**. [S. l.]: McGraw-Hill, 2008.

MCCARTNEY, S. **ENIAC: The triumphs and tragedies of the world's first computer**. [S. l.]: Walker & Company, 1999.

MENNINGER, K. **Number words and number symbols: A cultural history of numbers**. [S. l.]: Courier Corporation, 2013.

PATTERSON, D. A.; HENNESSY, J. L. **Computer organization and design: the hardware/software interface (the morgan kaufmann series in computer architecture and design)**. Paperback, Morgan Kaufmann Publishers, [S. l.], 2013.

PETERSON, J. L. **Computer organization and assembly language programming**. [S. l.]: Academic Press, 2014.

PUGH, E. W.; JOHNSON, L. R.; PALMER, J. H. **IBM's 360 and Early 370 Systems**. [S. l.]: MIT press, 1991.

RASHED, R. **The development of Arabic mathematics: between arithmetic and algebra**. [S. l.]: Springer Science & Business Media, v. 156 - 1994.

RHEINGOLD, Howard. **Tools for thought: The history and future of mind-expanding technology**. MIT press, 2000.

RIBEIRO, Leila *et al.* **Diretrizes da sociedade brasileira de computação para o ensino de computação na educação básica**. Sociedade Brasileira de Computação, 2019.

ROJAS, R.; HASHAGEN, U. **The first computers: History and architectures**. [S. l.]: MIT press, 2002.

ROJAS, R. **How to make Zuse's Z3 a universal computer**. IEEE Annals of the History of Computing, v. 20, n. 3, p. 51-54, 1998.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating system concepts essentials**. [S. l.]: Wiley Publishing, 2013.

STALLINGS, W. **Arquitetura e organização de computadores**. 8. ed. São Paulo (SP): Pearson, 2010.

STALLINGS, W. **Arquitetura e organização de computadores / Computer organization and architecture : designing for performance**. 10. ed. São Paulo (SP): Pearson, 2017.

STRATHERN, P. **Turing e o computador em 90 minutos**. [S. l.]: Editora Schwarcz-Companhia das Letras, 2000. 2000.

SWADE, D. D. **The construction of Charles Babbage's difference engine no. 2**. IEEE Annals of the History of Computing, [S. l.], v. 27, n. 3, p. 70-88, 2005.

SWAINE, M.; FREIBERGER, P. **Fire in the valley: The birth and death of the personal computer**. [S. l.]: Pragmatic Bookshelf, 2014.

TANENBAUM, A. S. **Modern operating systems**. [S. l.]: Pearson, 2018.

TANENBAUM, A. S. **Structured computer organization**. [S. l.]: Pearson Education India, 2016.

WALLMARK, L. **Grace Hopper: Queen of computer code**. [S. l.]: Union Square & Co., 2020.

WILLIAMS, M. R. **A history of computing technology**. [S. l.]: Prentice-Hall, Inc., 1985.

FIGURA 2. **Origem do termo bug.** [S. G. Ganesh](#), March 20, 2012. Disponível em: <<https://www.opensourceforu.com/2012/03/joy-of-programming-why-software-glitch-called-bug/>>

Design by [Fábrica de Conteúdos](#) Educação ©