

INTRODUÇÃO A COMPUTAÇÃO – VOL. 1

PROFESSOR: ANTÔNIO EUGÊNIO SILVA – ANO:2024

Sumário

1. HISTÓRIA DA COMPUTAÇÃO.....	3
1.1 Introdução	3
1.2 Época dos Dispositivos Mecânicos (500 a.C. – 1880).....	5
1.3 Época dos Dispositivos Eletromecânicos (1880 – 1930)	8
1.4 Época dos Componentes Eletrônicos	10
1.4.1 Primeiras Invenções (1930 – 1945)	10
1.4.2 A Evolução dos Computadores Eletrônicos - (1945 – ?).....	11
1.4.3 Evolução dos Computadores de Grande Porte (<i>Main Frames</i>) ..	17
1.4.4 Computadores Pessoais: Microcomputadores	17
1.5 Softwares	19
1.5.1 A História dos Sistemas Operacionais	19
1.6 Biografia de Cientistas da Área.....	24
2. SISTEMAS DE NUMERAÇÃO E PADRÕES DE CARACTERES	27
2.1 Notação posicional – base decimal	27
2.2 Outras Bases de Numeração	28
2.3 Conversão de bases	32
2.3.1 Conversão entre bases Potência de 2	32
2.3.2 Conversão de números de uma base B para a base 10.....	33
2.3.3 Conversão de Números Decimais para uma Base B	33
2.4 Aritmética Binária e Hexadecimal.....	38
2.4.1 Soma Binária.....	38
2.4.2 Subtração Binária	39
2.4.3 Aritmética Octal (em Base 8)	39
2.4.4 Aritmética Hexadecimal (em Base 16)	39
2.4.5 Conversão com Números Fracionários.....	39
2.5 Tipo Numérico	41
2.5.1 Representação em Ponto Fixo	42
2.5.2 Sinal e Magnitude	43
2.5.3 Representação de Números Negativos em Complemento	47
2.5.4 Ponto Flutuante - Padrão (IEEE 754)	52
3. ÁLGEBRA DE BOOLE	58
3.1 Noções de Teoria dos Conjuntos	58

3.1.1	Complemento, Negação ou Inversão	58
3.1.2	Intersecção ou Produto	59
3.1.3	União ou Adição.....	59
3.1.4	Propriedades.....	59
3.2	Noções de Álgebra Booleana.....	60
4.	CONCEITOS DA LÓGICA DIGITAL.....	64
4.1	Introdução	64
4.2	Portas e Operações Lógicas	65
4.2.1	Operação Lógica ou Porta AND (E)	67
4.2.2	Operação Lógica ou Porta OR (OU)	68
4.2.3	Operação Lógica NOT (Inversor).....	68
4.2.4	Operação Lógica NAND – NOT AND	68
4.2.5	Operação Lógica NOR – NOT OR.....	69
4.2.6	Operação Lógica XOR – EXCLUSIVE OR.....	70
4.3	Minimização de funções booleanas	70
4.3.1	Implementação de funções booleanas	70
4.4	Mapas de Karnaugh	72

1. HISTÓRIA DA COMPUTAÇÃO

1.1 Introdução

Podemos dizer que a origem da Informática remonta a 3.500 anos a.C., quando no Egito surge uma importante ferramenta manual para auxiliar na realização de cálculos, o ábaco. De 3.500 a.C. até o século XVI d.C. não se tem nenhum registro de outro desenvolvimento significativo de ferramentas que auxiliassem na realização de cálculos e manipulação de informações.

Somente em 1617, já na idade moderna, surge uma ferramenta que permite realizar, além de soma e subtração, também multiplicação, mas ainda de forma manual. No decorrer dos séculos XVII, XVIII e XIX outros inventos mecânicos foram criados, como a Pascalina de Blaise Pascal, que realizava soma e subtração (1642), a máquina de multiplicar e dividir de Gottfried Leibnitz (1694) e o aritmômetro de Thomas Colmar (1820) que aperfeiçoava as ideias empregadas nas máquinas anteriores (5).

Em 1822, o lorde inglês Charles Babbage (matemático e engenheiro) criou uma máquina que viria revolucionar a tecnologia utilizada até então, denominada Máquina Diferencial. Esta máquina resolvia polinômios e logaritmos. Em 1833 Babbage aperfeiçoa o projeto da Máquina Diferencial, e define o projeto da Máquina Analítica, que embute importantes conceitos computacionais empregados ainda hoje nos computadores modernos: dados armazenados em memória e programação sequencial. Babbage não conseguiu construir a máquina analítica, mas muito da história subsequente da computação foi baseada em seus trabalhos, pois várias ideias semelhantes foram colocadas em prática posteriormente (7).

Em 1890, Hermann Hollerith desenvolveu nos Estados Unidos um equipamento baseado em cartões perfurados para realizar o censo demográfico americano. Este equipamento foi utilizado ainda por várias décadas como periféricos de computadores eletrônicos.

Uma grande inovação surgiu em 1920, com o desenvolvimento dos comandos elétricos, que permitiram automatizar o controle dessas máquinas. Em 1936 o alemão Konrad Zuse apresentou o Z1, um dispositivo eletromecânico que efetuava as quatro operações aritméticas, mais raiz quadrada e conversão decimal/binário. Em 1941 ele apresentou o Z2, com algumas inovações em relação ao Z1, e depois criou o Z3, com memória para 64 palavras de 22 caracteres cada, que operou até ser destruído no bombardeio de Berlim em 1945. Desenvolveu ainda o Z4 com memória de 32 caracteres por palavra, que operou na Basileia até 1954.

Entre 1943 e 1944 surgiu nos Estados Unidos o MARK I, considerado o primeiro computador regido pelos princípios de Babbage, com tecnologia eletromecânica, seu criador foi Howard Aiken, apoiado pela IBM e a marinha americana. O MARK I continha um gerador de funções, utilizava cartões

perfurados para entrada de dados e máquina de escrever elétrica para a saída das informações.

Na mesma época, Alan Turing desenvolveu na Inglaterra o COLOSSUS, uma máquina com o objetivo de decifrar os códigos de comunicação da Alemanha, que continha 1.500 válvulas e processava 5.000 caracteres por segundo. Os incentivos ao desenvolvimento de pesquisas científicas e tecnológicas, principalmente em física e matemática, ocorridos durante a Segunda Guerra Mundial, contribuíram substancialmente para o surgimento da computação eletrônica.

O desenvolvimento dos componentes eletrônicos proporcionou a fabricação da válvula eletrônica, e em 1946 foi construído por Eckert e Mauchly, na Universidade da Pensilvânia, o ENIAC (*Electronic Numerical Integrator And Computer*), o primeiro computador eletrônico, que continha 18.000 válvulas e fazia 200 operações por segundo. Durante o projeto ENIAC, John von Neumann propõe pela primeira vez o conceito de programa residente, implementado posteriormente no EDVAC e EDSAC. O conceito de programa residente em memória é utilizado até hoje.

A partir de 1946, a evolução da eletrônica passou a ditar a evolução dos computadores. A variedade de modelos passou a ser tão grande, que não é mais possível citá-los um a um, tornando-se necessário agrupá-los em famílias, cujo denominador comum seja o componente eletrônico responsável pelo processamento das informações. Estas famílias, denominadas gerações, têm características comuns de funcionamento.

- A **primeira geração** (1946-1954) utilizava como elemento básico de processamento válvulas eletrônicas. Estas válvulas eram grandes, caras, lentas e queimavam com facilidade. Podemos citar como exemplos típicos: ENIAC, EDVAC, UNIVAC, IBM 650. O uso dos computadores era basicamente científico, instalados apenas em grandes centros de pesquisas.
- A **segunda geração** (1955-1964) passou a utilizar transistores como elemento básico para o processamento de dados. Muito menores que as válvulas, os transistores eram mais rápidos, baratos e duráveis. Podemos citar como exemplos: NCR 501, IBM 70, IBM 94 e CDC 6600. A Informática começava a entrar nas grandes empresas.
- A **terceira geração** (1964-1977) utilizava como elemento básico o circuito integrado (CI), que podia reunir numa pequena cápsula milhares de transistores. Podemos citar como exemplos: IBM 360, PDP 11 e HoneyWell Bull 200. Foi um período de grande expansão da Informática, onde o computador invadiu grande parte das empresas de médio e grande porte (2).
- A partir de 1977 até 1991, com o advento de CI do tipo VLSI (*Very Large Scale Integration*) nasce a **quarta geração**. O elemento básico de processamento passa ser o VLSI e a Informática passa a invadir muitas áreas do conhecimento humano, atingindo vários segmentos da sociedade.
- Os computadores da **quinta geração** usam processadores com milhões de transistores.

No entanto, a acentuada difusão do uso da Informática se deve em grande parte ao advento da computação pessoal, com o lançamento do PC em 1981. O IBM-PC foi lançado inicialmente utilizando o microprocessador 8086 da Intel. Este microprocessador apresentava plena capacidade de processamento de 16 bits e, para a época, grande poder de endereçamento de memória (1 Mb). Por problemas de suporte de periféricos que precisavam interagir com este microprocessador a IBM introduziu o 8088, que limitou o desempenho da máquina, mas permitiu um convívio amigável com a tecnologia de suporte existente no momento.

O verdadeiro marco da revolução na indústria da computação pessoal foi o lançamento do microprocessador 80286, em 1982. Foi o microprocessador 286 que transformou o PC da IBM no padrão a ser igualado por todos os demais fabricantes de compatíveis PC no mundo. Este microprocessador chegou a trabalhar numa frequência de *clock* de até 12 Mhz, com registradores e barramento de dados de 16 bits, barramento de endereço de 24 bits, possibilitando o acesso de até 16 Mb de memória RAM.

Em 1986 a Intel lança o microprocessador 80386, com controle de memória virtual interno, 4 Gb de espaço de endereçamento, 16 Mhz de *clock*, barramento de dados de 32 bits, e capacidade de processamento na casa dos 4 Mips. A partir do microprocessador 386, ambientes gráficos de trabalho, como o Windows 3.0 da Microsoft, passam a encontrar ressonância no mercado por apresentarem um desempenho mais satisfatório em microprocessadores de 32 bits.

Em 1991 a Intel lança o microprocessador 80486, com uma capacidade de processamento 2 vezes maior que o microprocessador 80386. No seu lançamento original o 486 trabalhava com uma frequência de *clock* de 25 Mhz (incrementada para 33 Mhz posteriormente), 8 Kb de cache interno de memória e uma unidade co-processadora de ponto flutuante interna. Utilizando-se de técnicas de duplicação e triplicação de *clock* a Intel evoluiu em pouco tempo o 486DX para 486DX2 (66 Mhz) e 486DX4 (100 Mhz), o que melhorou sensivelmente o desempenho dos computadores pessoais, sem perder a compatibilidade dos aplicativos.

Em 1993 a Intel evolui ainda mais os microprocessadores para uso em PC e lança o Pentium, com capacidade de processamento na casa dos 100 Mips, incorporando 3,5 milhões de transistores. Neste mesmo ano é lançado o PowerPC, um microprocessador que embute tecnologia RISC, com capacidade de processamento muito superior em relação aos microprocessadores Intel, resultado da parceria IBM/Apple/Motorola.

1.2 Época dos Dispositivos Mecânicos (500 a.C. – 1880)

O conceito de efetuar cálculos com algum tipo de equipamento data, pelo menos, do século V a.C. com os babilônios e sua invenção do *ábaco*.

Este dispositivo permitia a contagem de valores, tornando possível aos comerciantes babilônicos registrar dados numéricos sobre suas colheitas. Também os romanos se serviram muito dos ábacos, para efetuar cálculos

aritméticos simples. Ainda hoje há quem use tal tipo de dispositivo, ainda popular na China, por exemplo.

A primeira evolução do ábaco surgiu somente no século XVII (em 1642), quando o filósofo e matemático francês Blaise Pascal construiu um contador mecânico que realizava operações aritméticas de soma e subtração através de rodas e engrenagens dentadas. Este instrumento consistia em seis engrenagens dentadas, com um ponteiro indicando o valor decimal escolhido ou calculado. Cada engrenagem continha dez dentes que, após efetuarem um giro completo, acarretavam o avanço de um dente de uma Segunda engrenagem, exatamente o mesmo princípio de um odômetro de automóvel e base de todas as calculadoras mecânicas. Cada conjunto de ponteiros era usado como um *registrador* para armazenar temporariamente o valor de um número. Um registrador atuava como um *acumulador*, para guardar resultados e uma parcela. O outro registrador era utilizado para se introduzir um valor a ser somado ou subtraído do valor armazenado no acumulador. Quando se acionava a manivela e a máquina era colocada em movimento, os dois valores eram adicionados e o resultado aparecia no acumulador. O calculador de Pascal apresentou duas significativas inovações tecnológicas para sua época:

- a) permitia o uso de “vai 1”, passado automaticamente para a parcela seguinte; e
- b) utilizava o conceito de complemento para realizar operações aritméticas de subtração através de soma de complemento (este conceito é até hoje essencialmente a base de funcionamento dos circuitos de operação aritmética em ponto fixo dos computadores).

A máquina embora rudimentar, era eficaz para a sua época, sendo inteiramente mecânica e não automática (funcionava por comando de uma manivela acionada manualmente). A linguagem de programação Pascal foi assim chamada em honra deste cientista pelo seu trabalho pioneiro em matemática e também devido à sua invenção.

Algum tempo após a invenção de Pascal, outro filósofo e matemático, desta vez alemão, Gottfried Leibniz, construiu uma calculadora mais completa que a de Pascal, porque realizava as quatro operações aritméticas, e não apenas a adição e subtração da de Pascal. O calculador mecânico de Leibniz era uma duplicata do calculador de Pascal acrescido de dois conjuntos adicionais de rodas, as quais permitiam a realização de multiplicação e divisão por meio de um processo de operações sucessivas (sabemos ser possível realizar multiplicações por somas sucessivas e divisão por subtrações sucessivas). Ambas as máquinas, a de Pascal e a de Leibniz, eram manuais.

A primeira utilização prática de dispositivos mecânicos para computar dados automaticamente data do início de 1800. Por esta época, muita contribuição para o desenvolvimento inicial dos computadores foi obtida do aperfeiçoamento de processos em uma área inteiramente diferente de computação – a de tecelagem. Em 1801, Joseph Jacquard produziu com sucesso um retrato em tecelagem, cuja produção foi inteiramente realizada de forma mecânica e controlada automaticamente por instruções registradas em orifícios em cartões perfurados. Um dos mais conhecidos resultados deste processo, de realização de tarefas de

tecelagem por uma máquina controlada por programa armazenado, foi o retrato do próprio Joseph Jacquard em uma tapeçaria, o qual consumiu 24000 cartões.

Um dos últimos e mais importantes trabalhos pioneiros em computação por processos mecânicos foi realizado por um inglês de nome Charles Babbage, que, em 1823, foi contratado pela *Royal Astronomical Society of Great Britain* para produzir uma máquina calculadora programável, com a finalidade de gerar tabelas de navegação para a Marinha britânica. Em seu trabalho, Babbage projetou dois tipos de máquina: a máquina de diferenças e a máquina analítica.

A primeira delas, a *máquina de diferenças*, foi justamente idealizada para atender às necessidades da Marinha real inglesa. Na época, as tabelas de navegação eram escritas manualmente por diversos funcionários, contratados para: (1) realizar sucessivas e repetitivas operações de adição e multiplicação e (2) imprimir os resultados, escrevendo-os. Foi constatado que, devido à natureza permanente e repetitiva do processo realizado por humanos, sempre ocorriam erros (tanto nos cálculos, quanto na ocasião de registrar por escrito os resultados). O que Babbage se propunha (por contrato) era projetar uma máquina que realizasse de forma constante e sem erros o tedioso trabalho de cálculos, e registrasse, de forma também confiável, os resultados.

A *máquina de diferenças* (assim chamada devido ao nome do processo matemático de cálculo utilizado por ela – diferenças finitas) era um dispositivo mecânico, que só realizava adições e subtrações (como a máquina de Pascal) e cujos cálculos matemáticos se baseavam no processo de diferenças finitas, pelo qual é possível calcular fórmulas (até com polinômios e funções trigonométricas) utilizando apenas a operação de adição. Ela era constituída de um conjunto de registradores mecânicos, cada um contendo rodas com dígitos, que serviam para armazenar um valor decimal. A exemplo da calculadora de Pascal, dois registradores adjacentes eram conectados por um mecanismo de efetuar somas. A máquina era acionada por um motor movido a vapor, que realizava uma série de etapas e finalmente apresentava um resultado. Além disso, ela continha um dispositivo de gravação em uma chapa de cobre, uma espécie de agulha que marcava os valores na chapa, a qual servia de matriz para posterior impressão em papel. Este processo de gravação pode ser considerado como pioneiro em termos de dispositivos de armazenamento secundário.

Babbage projetou algumas dessas máquinas, capazes de realizar cálculos com valores de até 15 algarismos e com polinômios de até 3º grau. Ele conseguiu convencer o Governo inglês a financiar a construção de uma máquina mais sofisticada e precisa, capaz de calcular polinômios de até 6º grau e números de até 20 dígitos. Esta máquina funcionou e, após muito dinheiro gasto, o Governo inglês desistiu do projeto.

Nesse ínterim, Babbage passou a se dedicar ao projeto de um novo tipo de máquina, que se chamou *máquina analítica*. A máquina era, na realidade, um computador mecânico capaz de armazenar 1000 números de 20 algarismos e que possuía um programa que podia modificar o funcionamento da máquina, fazendo-a realizar diferentes cálculos. Esta era de fato a sua grande diferença e vantagem sobre as anteriores, o fato de se tornar de uso mais geral por possuir a capacidade de modificar suas operações e assim realizar diferentes cálculos. Pode-se dizer que esta máquina foi a precursora dos primeiros computadores

eletrônicos, inclusive no seu método de introduzir instruções por cartões perfurados (muito usado nas primeiras gerações de computadores eletrônicos).

Embora inteiramente mecânica, a máquina analítica de Charles Babbage essencialmente possuía os mesmos componentes que um computador atual. A memória era constituída de rodas dentadas de contagem; processador: com uma unidade capaz de realizar as quatro operações aritméticas (operando com pares de registradores) e “unidade de controle”, constituída de cartões perfurados convenientemente para realizar esta ou aquela operação; saída: para uma impressora ou para um dispositivo perfurador de cartões.

Além da fundamental característica de realização de programas de emprego geral, o projeto ainda acrescentou a capacidade de desvio da sequência de ações da máquina, isto é, um prenúncio do que mais tarde (nos atuais computadores) seriam as instruções “*Jump*” (desvio incondicional) e “p.ex., *Jump on zero*”(desvio condicional).

Alguns pesquisadores acreditam que Babbage utilizou em seus cartões perfurados a ideia de Jacques Jacquard, anteriormente citado.

O projeto final de Babbage, que pretendia ter a capacidade não de calcular valores com 20 dígitos, mas sim com 50 algarismos, nunca chegou a se tornar uma realidade física, talvez por estar realmente avançado demais para a época, quando a tecnologia de fabricação dos dispositivos necessários ao funcionamento das engrenagens não tinha a devida capacidade.

Alguns outros detalhes históricos interessantes podem ser citados, como a estimativa de Babbage de que sua máquina deveria realizar uma operação de adição em segundo e uma multiplicação em um minuto, e que o programa criado para fazer a máquina funcionar foi desenvolvido por uma moça chamada Ada Lovelace, que pode ser, então, considerada a primeira programadora de computador da história e que serviu de nome para a moderna linguagem de programação ADA, desenvolvida para o departamento de defesa dos EUA.

1.3 Época dos Dispositivos Eletromecânicos (1880 – 1930)

Com a invenção do motor elétrico no fim do século XIX, surgiu uma grande quantidade de máquinas de somar acionadas por motores elétricos, baseados no princípio de funcionamento da máquina de Pascal. Essas máquinas se tornaram dispositivos comuns em qualquer escritório até o advento das modernas calculadoras de bolso em 1970.

Em 1889, Hollerith desenvolveu o cartão perfurado para guardar dados (sempre o cartão perfurado, desde Jacques Jacquard) e também uma máquina tabuladora mecânica, acionada por um motor elétrico, que contava, classificava e ordenava informações armazenadas em cartões perfurados. Por causa desta invenção, o Bureau of Census dos EUA contratou Hollerith em 1890 para utilizar sua máquina tabuladora na apuração de dados do censo de 1890. O censo foi apurado em dois anos e meio, apesar do aumento da população de 50 para 63

milhões de habitantes em relação ao censo de 1880, que consumiu quase dez anos de processamento manual.

O sucesso de Hollerith com a apuração do censo conduziu à criação, em 1896, da *Tabulating Machine Company*, por onde Hollerith vendia uma linha de máquinas de tabulação com cartões perfurados. Em 1914, um banqueiro persuadiu três companhias a se juntarem, entre elas a empresa de Hollerith, formando a Computer Tabulating Recording Corporation, Thomas Watson foi contratado como gerente geral e em 1924 ele mudou o nome da companhia para IBM – *International Business Machines*, logo após ter iniciado no Canadá uma bem-sucedida filial. Atualmente, os cartões perfurados não são mais utilizados (até a década de 1980, eles foram um dos principais elementos de entrada de dados dos computadores digitais, inclusive nos IBM/360/370, e de outros fabricantes). Também eram chamados de cartões de Hollerith, assim como o código de 12 bits por eles usado também se denominava código de Hollerith.

A primeira máquina de calcular eletrônica somente surgiu por volta de 1935 e seu inventor foi um estudante de engenharia alemão, Konrad Zuse, cuja ideia consistia em criar uma máquina que usava relés mecânicos que, atuando como chaves, podiam abrir ou fechar automaticamente, o que levou à utilização de números binários em vez de algarismos decimais, utilizados nas engrenagens da máquina de Babbage.

Em 1936, Zuse deixou de ser estudante e profissionalmente criou sua primeira máquina, chamada Z1, baseada em relés mecânicos, que usava um teclado como dispositivo de entrada e lâmpadas (dispositivo binário – acesa e apagada) como componente de saída (o primeiro microcomputador comercial, o Altair, em 1974 também usava lâmpadas como dispositivo de saída). Zuse realizou alguns aperfeiçoamentos em seu “computador” até concluir, em 1941, o Z3, o qual utilizava relés eletromecânicos e era controlado por programa, sendo talvez o primeiro computador efetivamente operacional do mundo. Seu modelo Z4 foi usado pelos militares alemães para auxiliar no projeto de aviões e mísseis durante a Segunda Guerra Mundial. Provavelmente Zuse teria desenvolvido máquinas de maior capacidade e versatilidade se tivesse sido melhor financiado pelo governo alemão. Os bombardeios aliados na Alemanha destruíram a maior parte dos computadores construídos por Zuse, e por isso o seu trabalho foi praticamente perdido, restando apenas o registro histórico dessas invenções.

Outro “inventor” da época dos dispositivos eletromecânicos foi Howard Aiken, um físico e matemático americano que desenvolveu um “computador”, o Mark I, utilizando os princípios básicos da máquina de Babbage (era um sistema decimal e não binário como os de Zuse), com engrenagens decimais, mas com estrutura computacional baseada em relés eletromecânicos. O projeto, que foi financiado pela IBM, era capaz de armazenar 72 números, e as instruções de dois operandos eram introduzidas na máquina por meio de uma fita de papel perfurado. Ao ser completado em 1944, o Mark I podia realizar uma soma em seis segundos e uma divisão em 12 segundos (Charles Babbage imaginava que sua máquina analítica poderia realizar uma adição em um segundo). No entanto, a eletrônica já começava a substituir elementos eletromecânicos por dispositivos muito mais rápidos, as válvulas, o que já tornava o Mark I obsoleto antes de operar

comercialmente em escala, e o seu sucessor, o Mark II, nem chegou a ser concluído.

1.4 Época dos Componentes Eletrônicos

1.4.1 Primeiras Invenções (1930 – 1945)

O problema dos computadores mecânicos e eletromecânicos residia em dois fatos: *baixa velocidade de processamento*, devido à parte mecânica de seus elementos, e *falta de confiabilidade dos resultados*, já que seu armazenamento e movimento interno eram realizados por engrenagens, incapazes de realizar sempre o mesmo tipo de movimento, principalmente com desgaste causado pelo tempo.

Esses dois problemas só poderiam ser solucionados com a utilização de elementos de armazenamento e chaveamento que não tivessem partes mecânicas e fossem bem mais rápidos. Para tanto, os cientistas dedicados a esse trabalho passaram a explorar o uso de um componente eletrônico, *a válvula*, inventada em 1906.

Na mesma época em que Zuse e Aiken realizavam seus trabalhos com dispositivos eletromecânicos, dois outros cientistas desenvolveram computadores utilizando válvulas.

Um desses cientistas foi John Vincent Atanasoff, que, por volta de 1939, projetou uma máquina calculadora para resolver equações lineares, mas a invenção apenas ficou registrada historicamente, sem que a intenção de seu inventor, de que a máquina se tornasse um dispositivo de emprego geral, fosse realizada. A grande importância dessa invenção foi, no entanto, a atenção que despertou em um outro cientista, John Mauchly, um dos construtores do computador, o ENIAC, que é atualmente reconhecido como o que deu início à computação eletrônica, como veremos logo adiante.

Além de Atanasoff, outro cientista, o matemático inglês Alan Turing, desenvolveu uma máquina com componentes eletrônicos. Turing é bastante conhecido pela teoria de computação que desenvolveu e conhecida como máquina de Turing, descrita em 1937 e que consistia na definição de uma função de computação, pela qual uma máquina poderia simular o comportamento de qualquer máquina usada para computação, se fosse adequadamente instruída para tal (isto é, se recebesse instruções através de uma fita de papel perfurado). Porém, até pouco tempo atrás não havia registro de que ele tivesse desenvolvido outro tipo de trabalho, mais prático, para o desenvolvimento de computadores.

Recentemente, com a divulgação de documentos militares do Governo Britânico, antes sigilosos, é que se tomou conhecimento de que o primeiro computador verdadeiramente eletrônico foi colocado em operação em 1943, com o propósito de quebrar códigos militares secretos de comunicação dos alemães. Esta máquina, construída por Alan Turing com válvulas eletrônicas, foi denominada Colossus, provavelmente devido a seu tamanho. Sua grande desvantagem residia no fato de não ser uma máquina de emprego geral, pois não

podia resolver outros problemas a não ser a quebra de códigos militares. Ela era, então, um sistema de computação com programa único.

1.4.2 A Evolução dos Computadores Eletrônicos - (1945 – ?)

Apesar dos primeiros projetos de Atanasoff e Turing, reconhece-se outra máquina eletrônica como o primeiro computador, realmente falando, projetado como uma máquina de emprego geral, eletrônica e automática (!!!).

Costumava-se dividir (e não há razão para fazermos diferente) a evolução cronológica do desenvolvimento dos computadores até nossos dias de acordo com o elemento básico utilizado na fabricação dos componentes do processador central, o primeiro deles já citado como sendo a válvula eletrônica.

1.4.2.1 Primeira Geração: Válvula (1946-1954)

O primeiro computador eletrônico e digital, construído no mundo para emprego geral, isto é, com programa de instruções que podiam alterar o tipo de cálculo a ser realizado com os dados, foi denominado ENIAC (*Electronic Numerical Integrator And Computer*) e foi projetado por John Mauchly e John P. Eckert, de 1943 a 1946, tendo funcionado daí em diante até 1955, quando foi desmontado.

Em agosto de 1942, John Mauchly, inspirado no projeto de Atanasoff, propôs ao Exército americano o financiamento para a construção de uma máquina que auxiliasse os militares do Ballistics Research Laboratory (um departamento do Exército americano responsável pela elaboração de tabelas de alcance e trajetória para novas armas balísticas) em seu trabalho, reduzindo o tempo de elaboração das tabelas balísticas. Na época, o laboratório empregava mais de 200 pessoas para o cálculo das tabelas, as quais, usando máquinas calculadoras de mesa, resolviam repetidamente equações balísticas para gerar os dados necessários à formação das tabelas. Tabelas para uma simples arma poderiam levar até dias para serem completadas, e isto atrasava consideravelmente a entrega dos artefatos.

O ENIAC era uma máquina gigantesca, contendo mais de 17.000 válvulas e 800 quilômetros de cabos. Pesava cerca de 30 toneladas e consumia uma enorme quantidade de eletricidade, além do consumo de válvulas, que queimavam com grande frequência devido ao calor.

De qualquer modo, e apesar de ter ficado pronto após o término da guerra e, portanto, sem poder ser utilizado para o propósito inicial de seu financiamento, o ENIAC era extremamente rápido para sua época, realizando cerca de 10.000 operações por segundo. Ele possuía 20 registradores, cada um deles podendo armazenar um valor numérico de 10 dígitos; era uma máquina decimal (não binária) e, por isso, cada dígito era representado por um anel de 10 válvulas, uma das quais estava ligada em cada instante, indicando o algarismo desejado. O ENIAC era programado através da redistribuição de cabos em tomadas diferentes e rearranjo de chaves (possuía cerca de 6.000), tarefa que poderia levar muitos

dias (pode-se imaginar a redistribuição de cabos como uma tarefa análoga à das telefonistas em antigas mesas telefônicas).

O ENIAC provou, com sucesso, que era uma máquina de emprego geral ao ser utilizado para realização de complexos cálculos em relação ao uso da bomba-H, uma tarefa bem diferente daquela para a qual ele tinha sido construído. No entanto, era uma máquina de difícil operação e de manutenção dispendiosa devido às sucessivas queimas de válvulas.

De qualquer modo, a divulgação das características do ENIAC despertou o interesse de numerosos cientistas da área e vários projetos tiveram início na mesma época.

Enquanto Mauchly e Eckert iniciaram a construção de um novo computador, o EDVAC (*Electronic Discret Variable Automatic Computer*), um dos colaboradores do projeto ENIAC, o matemático John von Neumann, também iniciou outro projeto de aperfeiçoamento do computador inicial, denominado IAS, nome do local onde von Neumann foi trabalhar, o Institute for Advanced Studies da Universidade de Princeton.

O EDVAC de Mauchly e Eckert não foi adiante devido à saída de ambos da Universidade da Pennsylvania, para constituírem sua própria empresa, que mais tarde se tornou a UNIVAC. Recentemente, a UNIVAC uniu-se à Burroughs, constituindo-se na atual Unysis Corporation.

A outra vertente do aperfeiçoamento do ENIAC, pelo desenvolvimento do EDVAC, é atribuída, como já mencionado, a John von Neumann e é a ele que se credita de um modo geral a definição de uma arquitetura de computadores com *programa armazenado*, e que até os dias atuais é empregada nas máquinas modernas.

Em 1945, von Neumann divulgou seu conceito ao publicar a especificação básica do EDVAC, isto é, da sua versão do EDVAC, no trabalho "*First Draft of a Report on the EDVAC*" (primeiro rascunho de um relatório sobre o EDVAC). O relatório definia as características essenciais de uma máquina sequencial de programa armazenado. Nele foram introduzidos os aperfeiçoamentos desejados para reduzir os inconvenientes do ENIAC, tais como: a dificuldade de programar a recolocação da fiação (isto poderia ser realizado com o mesmo tipo de elementos que representavam os dados no ENIAC, eletronicamente) e o tipo de aritmética (substituindo a aritmética decimal pela binária devido à dificuldade e ao custo de construir uma máquina capaz de representar confiavelmente 10 níveis de tensão em vez de apenas dois).

Em 1946, von Neumann e vários outros cientistas em Princeton iniciaram a construção de uma nova máquina, um computador eletrônico de programa armazenado, o IAS, que se utilizava dos mesmos princípios descritos no referido relatório do EDVAC.

O IAS possuía as seguintes características básicas extraídas de [STAL 87] (embora pertença à primeira geração de computadores e tenha sido, para os padrões atuais, uma máquina limitada, o IAS é fundamental no estudo da arquitetura de computadores, pois a grande maioria de suas especificações permanece válida até o momento):

- era constituído de quatro unidades principais, a memória, a CPU, a UC e a parte de entrada e saída;

- possuía memória com 1.000 posições, chamadas palavras, cada uma podendo armazenar um valor com 40 dígitos binários (bits);
- tanto os dados (valores numéricos) quanto as instruções eram representados da mesma forma binária e armazenados na mesma memória;
- possuía 21 instruções de 20 bits cada uma, constituídas de dois campos, um com 8 bits, denominado código de operação (C.Op.), e o outro com 12 bits, denominado endereço, para localizar cada uma das 1.000 palavras, endereços de 000 a 999 (embora pudesse endereçar 4096 (4K) posições de memória, pois $2^{12} = 4096$, o IAS somente possuía 1.000 endereços);
- operava de modo repetitivo, executando um *ciclo de instrução* em seguida ao outro. Cada ciclo consistia em dois subciclos: o *ciclo de busca* (“fetch cycle”), onde o C.Op. da próxima instrução era trazido da memória para o IR e a parte de endereço da instrução era armazenada no MAR (Memory Address Register). Tão logo o C.Op. estivesse armazenado no IR, então se iniciava o outro subciclo, o *ciclo de execução*. O circuito de controle interpretava o código de operação e gerava os sinais apropriados para acarretar o movimento de dados ou a realização de uma operação na UAL – Unidade Aritmética e Lógica.

Conforme pode ser observado dessas especificações resumidas, o IAS possuía características de arquitetura que permaneceram ao longo do tempo. As máquinas evoluíram consideravelmente em velocidade, capacidade de armazenamento, miniaturização, consumo de energia e calor, e outras inovações, mas a arquitetura básica permaneceu.

Em 1949, a empresa fundada por Mauchly e Eckert construiu com sucesso o primeiro computador para fins comerciais, o UNIVAC I (*Universal Automatic Computer*), adquirido pelo Bureau of Census dos EUA, para processar os dados do censo de 1950. Pouco mais tarde, a Mauchly – Eckert Computer Corporation foi absorvida pela Sperry-Rand Corporation, como uma de suas subsidiárias, com o nome de UNIVAC.

A UNIVAC fabricou diversos outros tipos de computadores, a começar pelo UNIVAC II e, em seguida, a série 1100, mais voltada para a computação científica.

Em 1953, a IBM, até então mais voltada, e com sucesso, para a construção e comercialização de equipamentos de processamento por cartão perfurado, lançou o seu primeiro computador eletrônico de programa armazenado, o IBM-701, voltado para o processamento científico. Esta máquina possuía uma memória com 2K palavras de 36 bits. Em 1955, a IBM modificou o hardware do 701 para adaptá-lo ao uso comercial, lançando o IBM-702 e em 1956 foi lançado o IBM-704, com 4K palavras de memória e, finalmente, em 1958 a IBM lançou outra máquina, mais aperfeiçoada, o IBM-709. Nesta ocasião, a IBM já se destacava no mercado em relação à UNIVAC, que vinha senso a nº 1 desde 1950.

1.4.2.2 Segunda Geração – Transistores (1955-1964)

A eletrônica moderna surgiu em 23 de dezembro de 1947, quando três cientistas do Bell Laboratories – John Bardeen, Walter Bratain e William Schockley – produziram pela primeira vez o *efeito transistor*. Eles descobriram que as propriedades condutoras de um diodo semicondutor poderiam ser controladas por um terceiro elemento. Os transistores se tornaram não só um sucesso em toda a indústria eletrônica (custo, tamanho e desempenho melhores que os dispositivos a válvula), como também formaram a base de todos os computadores digitais até os dias atuais. O fato de que se pode ligar e desligar (dois estados) a corrente elétrica em um dispositivo é a base de toda a lógica digital.

A primeira companhia a lançar comercialmente um computador transistorizado foi a NCR, e logo em seguida a RCA. As vantagens dessas máquinas sobre suas antecessoras a válvula eram várias: eram mais baratas, menores e dissipavam menos calor, além do menor consumo de energia elétrica.

Esta nova geração de computadores também teve, e muito, a participação ativa da IBM, já se firmando como a mais importante companhia na produção de máquinas científicas. Ela transformou a série 700 em série 7000, esta transistorizada. O primeiro deles, o 7090, a mais tarde o 7094, que possuía um ciclo de instrução de dois microssegundos e 32K palavras, ainda de 36 bits. Além do domínio na computação científica, a IBM também produziu uma máquina comercial de enorme sucesso, o IBM-1401 (quatorze zero um, como era conhecido).

Com esta geração de computadores, outros fatos historicamente importantes também aconteceram. Entre eles:

- a) O aparecimento de outra companhia fabricante de computadores, a DEC – *Digital Equipment Corporation*, que viria mais tarde a se tornar o segundo maior fabricante do mundo, após a IBM. A DEC foi fundada em 1957 por Kenneth Olsen, um dos engenheiros do Lincoln Laboratory, do MIT (Massachusetts Institute of Technology), órgão que realmente desenvolveu o primeiro computador transistorizado, o TX-0 (embora o da NCR tenha sido o primeiro do tipo comercial, o TX-0 foi o primeiro de todos, embora apenas em nível experimental). No mesmo ano de 1957, a DEC lançou seu primeiro computador, o PDP-1, início de uma longa série de máquinas extraordinariamente eficazes e tecnologicamente avançadas, até o famoso PDP-11. Por ser uma máquina de pequeno porte, comparada com os computadores de até então, o PDP-1 também custava muito menos. Por essa razão e devido ao excelente desempenho para a sua faixa de preço, o computador da DEC teve grande aceitação do mercado, tornando-se um marco inicial da indústria de minicomputadores, da qual a DEC foi líder por um longo período (primeiro com os PDP e, em seguida, com a família VAX).
- b) O aparecimento de unidades aritméticas e lógicas mais complexas assim como unidades de controle.
- c) O aparecimento de linguagens de programação de nível superior ao das linguagens Assembly da época (na realidade, o FORTRAN para o IBM-704, era ainda de primeira geração).
- d) O surgimento de outra companhia importante, a Control Data Corporation, que lançou, em 1964, o sistema CDC-6000, voltado

primariamente para o processamento científico (a CDC sempre construiu computadores com uma maior vocação para o processamento numérico). Era uma máquina com palavra de 60 bits (apesar de não ser múltipla da base 2, possuía um valor grande, apropriado para o processamento numérico) e vários processadores independentes de entrada/saída, um total de 10, denominados PPU – *Peripheral Processing Unit*, que liberavam a UCP de várias tarefas, tornando o sistema ainda mais rápido.

1.4.2.3 Terceira Geração – Circuitos Integrados (1964-1977)

Em outubro de 1958, Jack Kilby, da Texas Instruments Co., colocou dois circuitos em uma peça de germânio. O dispositivo resultante era rudimentar e as interconexões tinham que ser realizadas por fios externos, mas este dispositivo é, em geral, reconhecido como o primeiro circuito integrado fabricado no mundo. Logo em seguida, Robert Noyce, da Fairchild Semiconductor Inc., utilizou-se de técnicas recém-criadas na mesma companhia e integrou múltiplos componentes em um substrato de silício. Os dispositivos comerciais que se sucederam mostraram a vantagem do silício sobre o germânio e permitiram o surgimento de uma nova geração de máquinas, mais poderosas e menores, devido à integração em larga escala (LSI – *Large Scale Integration*) que os circuitos integrados proporcionaram.

Em 1964, a IBM se utilizou das recentes inovações tecnológicas na área da microeletrônica (os circuitos integrados) e lançou a sua mais famosa “família” de computadores, a série 1360. Este sistema incorporou diversas inovações, que se tornaram um marco histórico em termos de computação e consolidaram a posição já obtida pela IBM, como a primeira fabricante de computadores do mundo.

Entre essas inovações, podemos citar:

- a) conceito de família de computadores, em vez de máquina individual, como até então. Este conceito permite que o fabricante ofereça o mesmo tipo de máquina (arquitetura igual – linguagem de máquina semelhante, etc.) com diferentes capacidades e preços, o que garante uma maior quantidade de clientes. O sistema 1360 foi lançado inicialmente com cinco modelos, modelo 30, 40, 50, 65 e 75, cada um com características próprias de ciclo de instrução, capacidade de memória instalável, quantidade de processadores de E/S, embora todos os modelos tivessem o mesmo conjunto básico de instruções (e, com isso, um programa criado em um modelo poderia, em princípio, ser executado em outro).
- b) A utilização de unidade de controle com microprogramação em vez de tradicionais unidades de controle no hardware.
- c) emprego de uma técnica chamada multiprogramação, pela qual vários programas compartilham a mesma memória principal e dividem o uso da UCP, dando a impressão ao usuário de que estão sendo executados simultaneamente.

- d) A elevada capacidade de processamento (para a época), com palavra de 32 bits e ciclo de instrução de até 250 nanossegundos, bem como a grande capacidade de armazenamento na memória principal, 16 Mbytes;
- e) A memória principal orientada a byte, isto é, cada célula de MP armazena oito bits de informação, independentemente do tamanho de bits definido para a palavra de dados. Esta característica tornou-se comum para quase todo o mercado (exceto máquinas científicas) e até hoje os computadores continuam com a MP orientada a byte.
- f) lançamento de um programa (conjunto de programas é o melhor termo) gerenciador dos recursos de hardware, de modo mais integrado e eficaz, o sistema operacional OS/360.

Além da família /360, esta época de LSI presenciou também o lançamento de outro minicomputador DEC, com circuitos integrados, memória principal orientada a byte e palavra de 16 bits, o PDP-11, uma das máquinas mais famosas em sua categoria. Seu sucessor, o sistema VAX-11, também teve o mesmo sucesso, especialmente no ambiente universitário.

1.4.2.4 Quarta Geração: A era do Chip (1977-1991)

O termo VLSI (*Very Large Scale Integration*), integração em larga escala, caracteriza uma classe de dispositivos eletrônicos capazes de armazenar, em um único invólucro, milhares e até milhões de diminutos componentes. Este dispositivo, denominada pastilha (“chip”), vem constituindo a base da estrutura de todos os principais sistemas de computação modernos.

A técnica de miniaturização de componentes eletrônicos conduziu, por volta de 1972, ao desenvolvimento de um outro tipo de computadores até então inexistente no mercado – os computadores pessoais ou microcomputadores.

Didaticamente os circuitos integrados são categorizados de acordo com a quantidade de integração que eles possuem:

- **LSI** (*Large Scale Integration* - 100 transistores): computadores da terceira geração
- **VLSI** (*Very Large Scale Integration* - 1.000 transistores): computadores da quarta geração
- **ULSI** (*Ultra-Large Scale Integration* - milhões de transistores): computadores da quinta geração.

Os computadores da quarta geração são reconhecidos pelo surgimento dos processadores—unidade central de processamento. Os sistemas operacionais como MS-DOS, UNIX, Apple’s Macintosh foram construídos. Linguagens de programação orientadas a objeto como C++ e Smalltalk foram desenvolvidas. Discos rígidos eram utilizados como memória secundária. Impressoras matriciais, e os teclados com os layouts atuais foram criados nesta época. Os computadores eram mais confiáveis, mais rápidos, menores e com maior capacidade de armazenamento. Esta geração é marcada pela venda de computadores pessoais.

1.4.2.5 Quinta Geração (1991- até ?)

Os computadores da quinta geração usam processadores com milhões de transistores. Nesta geração surgiram as arquiteturas de 64 bits, os processadores que utilizam tecnologias RISC e CISC, discos rígidos com capacidade superior a 600GB, pen-drives com mais de 1GB de memória e utilização de disco ótico com mais de 50GB de armazenamento.

A quinta geração está sendo marcada pela inteligência artificial e por sua conectividade. A inteligência artificial pode ser verificada em jogos e robôes ao conseguir desafiar a inteligência humana. A conectividade é cada vez mais um requisito das indústrias de computadores. Hoje em dia, queremos que nossos computadores se conectem ao celular, a televisão e a muitos outros dispositivos como geladeira e câmeras de segurança.

1.4.3 Evolução dos Computadores de Grande Porte (*Main Frames*)

Os computadores de grande porte se constituíram nas principais máquinas das empresas, desde os primórdios da computação, com o lançamento do UNIVAC I e do IBM-701, e até cerca de três a quatro anos atrás, quando a capacidade sempre crescente e o custo bem menor dos microcomputadores orientaram as intenções dos usuários na ocasião de implantar novos sistemas ou atualizar os antigos, grande parte deles substituindo os “main frames” por estações de trabalho em rede ou mesmo redes locais de microcomputadores.

Um dos principais representantes desta categoria, o sistema IBM/360, teve uma evolução tecnológica acentuada e permanente, desde 1964 até 1988, já como o nome de IBM/370, tendo continuado o desenvolvimento com novos sistemas, porém sempre com a mesma arquitetura básica, como os IBM-43xx, IBM-308x e IBM-309x.

Outra classe de computadores bastante específica e com aplicações científicas definidas é a de supercomputadores, entre os quais são mais significativos: a família CRAY (CRAY-1, CRAY-2, CRAY-X/MP, CRAY-Y/MP), a família IBM-90xx, com processamento vetorial, e a família CDC-CYBER.

1.4.4 Computadores Pessoais: Microcomputadores

Em 1971, uma companhia criada para produzir componentes eletrônicos, a Intel Corporation, sediada na Califórnia, EUA, produziu a primeira UCP em uma só pastilha de circuito integrado, denominada Intel-4004. Esta UCP, que se destinava a uma calculadora, possuía palavra de 4 bits e tinha cerca de 2.300 transistores na pastilha. Logo em seguida, a Intel lançou um novo microprocessador, desta vez com 8 bits de palavra e 16K de memória, o Intel-8008.

Tanto o 4004 quanto o 8008 eram UCP destinadas a uma aplicação específica (o 8008 destinava-se à Display Terminals Corporation, para servir de controlador de um monitor de vídeo). Embora a empresa solicitante da pastilha

nunca tivesse usado o 8008, a Intel vendeu uma quantidade não esperada dessa pastilha, mesmo com os problemas de pouca memória e pequeno conjunto de instruções. Então, em 1973, a Intel lançou o seu grande sucesso da época, o primeiro microprocessador de emprego geral do mundo, o Intel 8080. O 8008 possuía cerca de 3.500 transistores encapsulados na pastilha, enquanto que o 8080 tinha em torno de 5.000 transistores. Este último possuía também 8 bits de tamanho de palavra, capacidade maior de memória (cada endereço tinha 16 bits e, então, a memória podia conter até 64Kbytes) e um grande conjunto de instruções (78 instruções). O 8080 vendeu aos milhões e, desde então, a Intel não parou mais de crescer e desenvolver novos produtos, até o seu mais recente lançamento, o microprocessador Pentium III, contendo mais de 4 milhões de transistores na pastilha.

1.5 Softwares

Assim como o hardware (parte física de um sistema de computação) vem evoluindo desde a criação do primeiro computador eletrônico, os softwares e a tecnologia de comunicação de dados utilizadas por sistemas de computação, também sofreram sensível evolução.

1.5.1 A História dos Sistemas Operacionais

1.5.1.1 Primeira Geração - Válvulas e Painéis

Nesta época, um único grupo de pessoas era responsável pelo projeto, construção, programação, operação e manutenção de cada máquina. Toda a programação era feita em código absoluto, muitas vezes através da fiação para controlar as funções básicas da máquina. O conceito de linguagem de programação ainda não existia. Os sistemas operacionais também não. O acesso ao computador por parte do usuário era feito através da reserva antecipada de tempo de máquina. Ao chegar sua vez de usar o computador, o usuário fazia sua própria programação nos painéis da máquina. Ao chegar sua vez de usar o computador, o usuário fazia sua própria programação nos painéis da máquina e passava a torcer para que nenhuma das 20.000 válvulas do computador viesse a queimar enquanto ele estivesse trabalhando. Nessa época os programas processados pelos computadores eram constituídos essencialmente por cálculos numéricos repetitivos, como por exemplo a geração de tabelas de funções trigonométricas.

No início dos anos 50, houve uma melhora no uso de tais máquinas com o advento do cartão perfurado que tornou possível a codificação de programas em cartões e sua leitura pela máquina, dispensando a programação através de painéis.

1.5.1.2 Segunda Geração - Transistores

Com o emprego desta nova tecnologia em meados dos anos 50 os computadores tornaram-se confiáveis a ponto de serem comercializados.

Em vista do alto custo de tais equipamentos, não foi surpresa o fato de se encontrar uma solução que reduzisse o tempo de máquina desperdiçado. A solução encontrada, denominada de sistema batch (lote), consistia em coletar na recepção de um conjunto de *jobs* e fazer a leitura dos mesmos para uma fita magnética empregando um computador pequeno e relativamente barato, tal como o IBM 1401, que era muito bom na leitura de cartões, na cópia em fita e na impressão de resultados.

Os computadores da segunda geração eram usados maciçamente na realização de cálculos científicos e de engenharia, tal como a obtenção da solução de equações diferenciais parciais. A programação era feita em linguagem

FORTTRAN ou em linguagem de montagem. Os sistemas operacionais típicos da época eram o FMS (Fortran Monitor System) e o IBSYS, ambos sistemas operacionais desenvolvidos pela IBM para rodar no 7094.

1.5.1.3 Terceira Geração - CI's e Multiprogramação

Os sistemas operacionais da terceira geração nos reservam três técnicas importantes de acordo com os avanços da tecnologia, nas quais são:

MULTIPROGRAMAÇÃO / SPOOL / TIME-SHARING

MULTIPROGRAMAÇÃO: No 7094, quando o job corrente parava para aguardar a conclusão de operações de entrada/saída, o processador também permanecia inativo até a conclusão da operação. Nos casos das aplicações científicas, as operações de entrada/saída não são muito freqüentes, de tal maneira que o tempo perdido aguardando sua conclusão não chega a ser significativa. Já no caso do processamento comercial, o tempo de espera por entrada/saída pode chegar a 80 ou 90 por cento do tempo total de processamento, de maneira que algo precisava ser feito para evitar desperdício. A solução inicial foi dividir a memória em diversas partes, com um job alocado a cada uma delas.

SPOOL (*Simultaneous Peripheral Operation On Line*): Outra característica dos sistemas operacionais de terceira geração era sua capacidade de ler jobs de cartão direto para o disco, tão logo a massa de cartões tivesse chegado à sala do computador. Desta forma, assim que um job ativo terminasse, o sistema operacional carregaria um novo job na partição livre de memória, proveniente do disco.

TIME-SHARING : Compartilhamento de tempo, ou seja, uma variação dos sistemas operacionais, onde cada usuário tinha um terminal on-line à sua disposição.

1.5.1.4 Quarta Geração - CI's e Multiprogramação

Com o desenvolvimento da integração de circuitos em grande escala (LSI), apareceram chips com milhares de transistores encapsulados em um centímetro quadrado de silício, nascendo daí a ideia do computador pessoal. Atualmente, os mais poderosos computadores pessoais são denominados estações de trabalho (*wokstations*). Tais máquinas são usadas nas mais diferentes atividades e usualmente estão conectadas a uma rede pública ou privada, que permite a troca de informações entre todas as máquinas ligadas a ela.

A grande disponibilidade de poder computacional, levou ao crescimento de uma indústria voltada para a produção de software para estas máquinas. A maioria destes softwares é "ameno ao usuário" (user-friends), significando que

eles são voltados para pessoas que não tem nenhum conhecimento de computadores, e mais que isto, não tem nenhuma vontade de aprender nada sobre este assunto.

Um desenvolvimento interessante que começou a tomar corpo em meados dos anos 80 foi o dos sistemas operacionais para redes e o dos sistemas operacionais distribuídos.

Em uma rede de computadores, os usuários estão conscientes da existência de um conjunto de máquinas conectadas à rede, podendo, portanto, ligar-se a máquinas remotas e solicitar serviços das mesmas. Cada uma destas máquinas roda seu próprio sistema operacional e tem seu próprio usuário.

Os sistemas operacionais de rede não diferem fundamentalmente daqueles usados em máquinas monoprocessadoras. Obviamente, eles precisam de uma interface controladora de rede e de um software específico para gerenciar tal interface, além de programas que permitam a ligação de usuários a máquinas remotas e seu acesso a arquivos também remotos. Tais características não chegam a alterar a estrutura básica do sistema operacional usado para máquinas com um único processador.

Já os sistemas operacionais distribuídos precisam de mais do que a simples adição de poucas linhas de código a um sistema usado em máquinas monoprocessadoras, pois os sistemas ditos distribuídos diferem dos centralizados em pontos bastante críticos. Por exemplo, os sistemas operacionais distribuídos permitem que programas rodem em vários processadores ao mesmo tempo, necessitando, portanto, de algoritmos de escalonamento de processador bem mais elaborados, de forma a otimizar o grau de paralelismo disponível no sistema.

A seguir serão apresentadas duas tabelas que procuram resumir, respectivamente, a evolução da tecnologia de software e de comunicação de dados.

Tabela: Evolução do software

Data	Acontecimento
1946	Konrad Zuse desenvolve Plankalkul, aplicada, entre outras coisas, em jogo de xadrez
1949	Aparece a primeira linguagem de programação realmente usada em computadores eletrônicos, denominada Short Code
1951	Grace Hopper, trabalhando para Remington Rand, inicia o trabalho do primeiro compilador amplamente divulgado, denominado A-0
1952	Alick E. Glennie projeta um sistema de programação chamado AUTOCODE, um compilador rudimentar
1954	Surgimento do primeiro Assembler
1955	UNIVAC da General Electric é colocado para trabalhar em folha de pagamento
1957	Surgimento da linguagem de programação Fortran, projeto liderado por John Backus
1958	Surgimento da linguagem de programação Lisp (utilizada em estudos de Inteligência Artificial)

	Surgimento da linguagem de programação Algol 58 Surgimento do Fortran II
1960	Surgimento da linguagem de programação Cobol, criada pela Codasyl (Conference on Data Systems and Languages)
1962	Surge o conceito de sistema operacional
1964	Surgimento da linguagem de programação PL/1
1969	Surgimento do sistema operacional Unix
1971	Surgimento da linguagem de programação Pascal
1972	Surgimento da linguagem de programação Smalltalk, desenvolvida pela Xerox PARC (programação orientada a objetos) Dennis Ritchie desenvolve o compilador C
1975	Surgimento da linguagem de programação Basic (residente em microcomputadores) Bill Gates e Paul Allen escrevem uma versão do Basic, a qual é vendida para MITS (Micro Instrumentation and Telemetry Systems), na base de royalty por cópia
1978	Surgimento do Visicalc (primeira planilha eletrônica de sucesso comercial)
1979	Surgimento do Dbase II, desenvolvido pela Ashton Tate
1980	Surgimento da linguagem de programação ADA (o nome da linguagem é em homenagem a Augusta Ada Byron, considerada a primeira programadora de computadores) Surgimento do primeiro processador de textos
1981	Surgimento do sistema operacional MS-DOS (para computadores pessoais padrão IBM-PC)
1983	Surgimento do Unix System V
1985	Surgimento da linguagem de programação C++ (orientada a objetos) Microsoft lança o Windows (ambiente de sistema operacional) Surgimento do Aldus Pagemaker para Macintosh
1986	Borland lança o Turbo Prolog (utilizada para o desenvolvimento de sistemas especialistas) Surgimento da linguagem de programação Eiffel (linguagem orientada a objetos)
1987	A IBM lança o sistema operacional OS/2 A versão 4.0 do Turbo Pascal é lançada
1988	A especificação para CLOS é publicada Nikolas Wirth termina Oberon (outra linguagem orientada a objetos)
1989	A especificação C ANSI é publicada
1990	Lançamento do C++ 2.1 Lançamento do Fortran 90
1991	Surgimento da linguagem de programação Visual Basic
1993	Primeira proposta para Cobol orientado a objetos
1994	Microsoft incorpora Visual Basic para aplicações no Excel

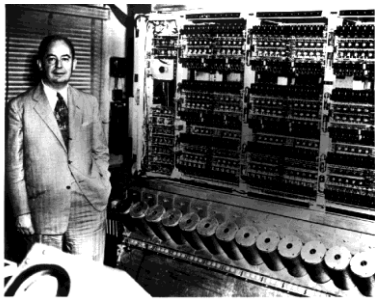
1995	Surgimento do sistema operacional Windows 95 (Microsoft) ISO aceita a revisão 1995 da linguagem de programação ADA, chamada ADA 95, que inclui programação orientada a objetos e suporte para sistemas em tempo real
1996	Antecipada a release do primeiro padrão C++ ANSI

Tabela: Evolução da comunicação de dados

Data	Acontecimento
1956	URSS lança o Sputnik, primeiro satélite artificial da Terra. Em resposta os Estados Unidos formam uma agência de pesquisa (ARPA - Advanced Research Projects Agency) dentro do Departamento de Defesa para estabelecer uma liderança norte-americana em ciência e tecnologia, aplicáveis para fins militares
1968	Apresentação da rede de trabalho para ARPA
1969	ARPAnet é comissionada pelo Departamento de Defesa Americano para desenvolvimento de pesquisas em redes
1970	ALOHAnet é desenvolvida por Norman Abrahamson, Universidade do Hawaii
1972	Conferência internacional em comunicação de computadores, com demonstração de 40 máquinas (ARPAnet) interconectadas
1973	Primeira conexão internacional realizada na ARPAnet (entre Inglaterra e Noruega)
1974	Vint Cerf e Bob Kahn publicam "Um Protocolo para Interoperação de Pacotes" que especificava em detalhes o projeto de um Programa de Controle de Transmissão (TCP - Transmission Control Program)
1975	Gerenciamento operacional da Internet é transferido para o DCA
1976	AT&T desenvolve o uucp (unix-to-unix copy), distribuído junto com o Unix um ano mais tarde
1977	Theorynet é criada na Universidade de Wisconsin, oferecendo correio eletrônico para mais de 100 pesquisadores em ciência da computação
1979	USENET é estabelecida usando uucp
1982	INWG estabelece o Protocolo de Controle de Transmissão (TCP - Transmission Control Protocol) e o Protocolo Internet (IP - Internet Protocol), comumente conhecido como TCP/IP
1984	É introduzido o conceito de DNS (Domain Name Server)
1986	É criada a NSFNET (rede da National Scientific Foundation)
1989	Internet ultrapassa a casa dos 100.000 hosts
1992	Criação da Internet Society
1994	Expansão a nível mundial da Internet

1.6 Biografia de Cientistas da Área

JOHN VON NEUMAN



Considerado um gênio à altura de Leonardo Da Vince, o húngaro John Von Neuman (1903-1957) era poliglota e especialista em ciências físicas e matemáticas (Matemática Aplicada, Física, Meteorologia, Economia e Computação) Durante a Segunda Guerra Mundial, Von Neuman participou do projeto de desenvolvimento da bomba atômica para o governo americano. é considerado o maior matemático deste século depois de Einstein . Além disso tinha uma memória fantástica, sendo capaz de lembrar, integralmente, o texto dos livros que lia. Já era o matemático mais respeitado no mundo quando se interessou por computadores.

John definiu o esquema básico de funcionamento dos computadores, a chamada Arquitetura de Von Neuman, que é usada até hoje no projeto de novas máquinas. Ele foi uma das pessoas envolvidas no projeto do primeiro computador eletrônico, o ENIAC, em 1943, depois dedicou-se a construir sua própria versão do ENIAC, o EDVAC, que ficou pronto em 1951 no Instituto de Estudos Avançados de Princeton e tinha maior capacidade de armazenamento que o ENIAC.

A arquitetura a proposta por Von Neuman consiste em quatro estruturas básicas que se relacionam entre si : Dispositivos de Entrada, CPU, Memória e Dispositivos de Saída. isto possibilita que os programas estejam armazenados na própria memória do computador, podendo ser modificados facilmente. Von Neuman já previa a maneira como os computadores revolucionariam todos os campos da vida moderna, inclusive com a explosão de utilização de computadores pessoais. Na sua visão, os computadores seriam utilizados principalmente em aplicações científicas e para processamento de grandes volumes de dados, como censo ou outros.

ALAN TURING



Um dos grandes pioneiros no campo da computação, o inglês Alan Mathison Turing começou a se interessar sobre o assunto imaginando os possíveis usos do computador. O primeiro resultado de seus estudos foi uma máquina teórica conhecida como Turing Universal Machine, que podia calcular qualquer número e função, de acordo com instruções apropriadas.

Com a intenção de descobrir os limites do uso dos computadores, Turing elaborou um teste para descobrir se podíamos atribuir à máquina a noção de inteligência. O Teste de Turing consistia

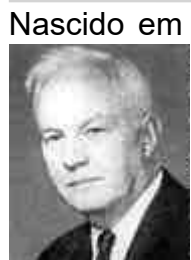
em colocar uma pessoa isolada em uma sala e ver se ela seria capaz de distinguir se quem respondia às suas perguntas era um homem ou uma máquina.

Durante a II Guerra Mundial, Turing trabalhou no Departamento de Comunicações da Gran Bretanha (Government Code and Cypher School) em Buckinghamshire, O Governo tentava quebrar o código das comunicações alemãs, que era trocado constantemente por um computador chamado Enigma, de modo que os inimigos não tivessem tempo de decifrá-lo. Para isso a equipe de Turing elaborou um sistema chamado de Colossus que possuía 1500 válvulas e processava 5000 caracteres por segundo e é considerado um dos precursores dos computadores digitais. Depois disso, nos Estados Unidos, Turing foi chamado para fazer o inverso, ou seja, estabelecer códigos seguros para comunicações transatlânticas entre os aliados. Também nos EUA Turing participou junto com Von Neuman do projeto ENIAC.

Depois da guerra Turing participou de um projeto totalmente inglês que seria chamado de ACE (Automatic Computing Engine) mas depois o abandonou decepcionado com a demora da construção. Alan Turing suicidou-se em Manchester, no dia 7 de junho de 1954, durante uma crise de depressão.

Disciplinas relacionadas ao trabalho de Alan Turing ALGORITMOS, FUNDAMENTOS DE INTELIGENCIA ARTIFICIAL (Teste de Turing), LINGUAGENS de PROGRAMAÇÃO, REDES NEURAIS, GERENCIAMENTO DE REDES.

HASKELL CURRY



Nascido em 1900, em Millis, Estados Unidos, Haskell Brooks Curry, Estudou em Harvard e fez doutorado em Göttingen em 1929, Sua especialidade era lógica matemática, com interesse especial em Teoria dos Sistemas formais e Processos.

Quando foi trabalhar em um projeto complexo no ENIAC, Curry percebeu que a linguagem utilizada, concebida por Neuman e sua equipe, era muito complicada podendo confundir o próprio programador. Por isso sugeriu uma notação mais compacta, mas, a

linguagem que Curry criou pecava pela maneira estranha com que analisava e dividia os problemas.

A principal contribuição de Curry para a informática foi a conversão de algoritmos para código de máquina de modo que se pudesse criar uma descrição recursiva de um procedimento para a conversão de expressões aritméticas claras em um código de máquina apropriado. Por causa disso Haskell foi o primeiro a descrever a geração de código de um compilador.

A Linguagem que leva seu nome (Haskell) é, estudada em LP2.

Curry procurou simplificar as linguagens de programação por isto se encaixa em SEMANTICA DE LINGUAGENS.

ALONZO CHURCH

O matemático norte americano Alonzo Church Professor de filosofia e matemática na University of California, em Los Angeles, escreveu Introduction of Mathematical Logic, foi um dos pioneiros da computação, sua principal contribuição foi o cálculo Lambda, relacionado à funções, que serviu de base para a linguagem de programação funcional LISP.

O cálculo lambda consiste, principalmente, em permitir que funções possam ser utilizadas como um dado qualquer em argumento de outras funções e combinar funções formando uma nova função. Isto é utilizado nas linguagens funcionais principalmente a LISP, que é considerada a única linguagem funcional verdadeira.

A utilização do cálculo Lambda permite uma linguagem simples, com sintática e semântica de fácil compreensão, pois contém apenas três tipos de expressão. Os cálculos desenvolvidos por Church estão presentes em LOGICA TEORIA DA PROVA, METODOS NUMÉRICOS e METODOS FORMAIS.

KURT GODEL



Kurt Gödel era checo naturalizado americano, ficou famoso por um trabalho sobre lógica matemática conhecido como Prova de Gödel. Doutor pela Universidade de Viena realizou trabalhos importantes em filosofia e matemática, entre eles "the Axiom of Choice and of the Generalized Continuum-Hypothesis with the Axioms of Set Theory (1940)" e "Rotating Universes in General Relativity Theory (1950). Gödel mudou-se para os Estados Unidos em 1940 onde trabalhou no Instituto de Estudos Avançados de Princeton (Institute for Advanced Studies, Princeton) em New Jersey onde ficou até 1953 quando foi ensinar matemática na Universidade de Princeton onde ficou até sua morte. Por ser bastante abstrato o trabalho de Gödel é empregado em disciplinas ligadas a matemática como Estatística e probabilidade.

2. SISTEMAS DE NUMERAÇÃO E PADRÕES DE CARACTERES

2.1 Notação posicional – base decimal

Desde os primórdios da civilização o homem vem adotando formas e métodos específicos para representar números, tornando possível, com eles, contar objetos e efetuar operações aritméticas (de soma, subtração etc).

A forma mais empregada de representação numérica é a chamada notação posicional. Nela, os algarismos componentes de um número assumem valores diferentes, dependendo de sua posição relativa ao número. O valor total do número é a soma dos valores relativos de cada algarismo. Desse modo, é a posição do algarismo ou dígito que determina seu valor.

A formação de números e as operações com eles efetuadas dependem, nos sistemas posicionais, da quantidade de algarismos diferentes disponíveis no referido sistema. Há muito tempo a cultura ocidental adotou um sistema de numeração que possui dez diferentes algarismos – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 – e, por essa razão, foi chamado de sistema decimal.

A quantidade de algarismos disponíveis em um dado sistema de numeração é chamada de base: a base serve para contarmos grandezas maiores indicando a noção de grupamento. O sistema de dez algarismos, acima mencionado, tem base 10; um outro sistema que possua apenas dois algarismos diferentes (0 e 1) é de base 2, e assim por diante.

Vamos exemplificar o conceito de sistema posicional. Seja o número 1303, representado na base 10, escrito da seguinte forma:

1303₁₀

em base decimal, por ser a mais usual, costuma-se dispensar o indicador da base, escrevendo-se apenas o número:

1303

neste exemplo, o número é composto de quatro algarismos:

1, 3, 0, 3

e cada algarismo possui um valor correspondente à sua posição no número.

Assim, o primeiro 3(algarismo mais à direita) representa 3 unidades. Neste caso, o valor absoluto do algarismo (que é 3) é igual ao seu valor relativo (que também é 3), por se tratar da 1ª posição (posição mais à direita, que é a ordem das unidades). Considerando-se o aspecto três vezes a potência 0 da base 10 ou **$3 \times 10^0 = 3$**

Enquanto o segundo 3, vale três vezes a potência 2 da base 10 ou **$3 \times 10^2 = 300$**

O valor total do número seria então:

$$1000 + 300 + 0 + 3 = 1303_{10}$$

$$1 \times 10^3 + 3 \times 10^2 + 0 \times 10^1 + 3 \times 10^0 = 1303_{10}$$

Generalizando, num sistema qualquer de numeração posicional, um número **N** é expresso da seguinte forma:

$$N = (d_{n-1} d_{n-2} d_{n-3} \dots d_1 d_0)_b$$

Onde

D indica cada algarismo do número;

n-1, n-2, 1, 0 índice, indicam a posição de cada algarismo;

b indica a base de numeração; **n** indica o número de dígitos inteiros.

O valor do número pode ser obtido do seguinte somatório:

$$N = d_{n-1} \cdot b^{n-1} + d_{n-2} \cdot b^{n-2} + \dots + d_1 \cdot b^1 + d_0 \cdot b^0$$

Desse modo, na base 10, podemos representar um número:

$$N = 3748$$

Onde

$$N = 4 \text{ (quatro dígitos inteiros)}$$

Utilizando a fórmula indicada em (1.1):

$$d_{n-1} = 3 \text{ ou } d_3 = 3; \quad d_2 = 7; \quad d_1 = 4; \quad d_0 = 8$$

ou obtendo seu valor de acordo com a fórmula mostrada

$$N = 3 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 8 \times 10^0 = 3000 + 700 + 40 + 8 = 3748$$

2.2 Outras Bases de Numeração

Vejamos, em seguida, como representar números em outra base de numeração. Entre as bases diferentes da 10, consideremos apenas as bases 2 e potências de 2, visto que todo computador digital representa internamente as informações em algarismos binários, ou seja, trabalha em base 2. Como os números representados em base 2 são muito extensos (quanto menor a base de numeração, maior é a quantidade de algarismos necessários para indicar um dado valor) e, portanto, de difícil manipulação visual, costuma-se representar externamente os valores binários em outras bases de valor mais elevado. Isso

permite maior compactação de algarismos e melhor visualização dos valores. Em geral, usam-se as bases octal ou hexadecimal, em vez da base decimal, por ser mais simples e rápido converter valores binários (base 2) para valores em bases múltiplas de 2.

Utilizando-se a notação posicional indicada na expressão de N, representam-se números em qualquer base:

(1011)₂ - base 2
(342)₅ - base 5
(257)₈ - base 8

Sobre o assunto, podemos concluir:

- a) O número máximo de algarismos diferentes de uma base é igual ao valor da base.

Exemplo:

Na base 10 temos dez dígitos: de 0 a 9;

Na base 2 temos apenas dois dígitos: 0 e 1;

Na base 5 temos cinco dígitos: de 0 a 4.

- b) o valor do algarismo mais à esquerda (mais significativo) de um número de **n** algarismos inteiros é obtido pela multiplicação de seu valor absoluto (algarismo **d_{n-1}**) pela base elevada à potência (**n-1**), ou seja: (**d_{n-1} x bⁿ⁻¹**)

- c) o valor total do número é obtido somando-se n valores, cada um expressando o valor relativo de um dos n algarismos componentes do número.

A base do sistema binário é 2 e, conseqüentemente, qualquer número, quando representado nesse sistema, consiste exclusivamente em dígitos 0 e 1. Por exemplo, o número binário 11011 possui cinco dígitos, ou algarismos binários. Diz-se que o referido número é constituído de 5 bits.

A base 2, ou sistema binário, é um sistema de numeração posicional que utiliza apenas dois símbolos: 0 e 1. Cada posição de um número binário representa uma potência de 2, diferentemente do sistema decimal, onde cada posição representa uma potência de 10. No contexto da computação, o sistema binário é fundamental porque os circuitos eletrônicos operam de forma eficiente com dois estados físicos distintos, permitindo a representação dos valores 0 e 1.

- **Exemplo de Representação Binária**

O número binário 1011 equivale, em decimal, a:
 $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11.$

- **Bit**

O bit é a menor unidade de informação digital, resultante da junção das palavras em inglês 'binary digit'. Um bit pode assumir apenas dois valores: 0 ou 1.

Em sistemas digitais, esses valores podem representar estados como ligado/desligado, verdadeiro/falso, sim/não, etc.

- **Byte**

O byte é um conjunto de 8 bits. Esse agrupamento padrão foi adotado porque 8 bits permitem representar até 256 combinações diferentes ($2^8 = 256$). Isso é suficiente para representar caracteres, números e outros tipos de dados simples em computadores. Por exemplo, o valor binário 01100001 equivale ao número decimal 97, que representa a letra 'a' no código ASCII.

Tabela dos Múltiplos do Byte (KB até Quettabyte)

A tabela abaixo apresenta os principais múltiplos do byte, utilizando o padrão binário, do kilobyte (KB) ao quettabyte (QB), incluindo a sigla, o nome, a equivalência em bytes e a potência de 2 correspondente.

Sigla	Nome	Equivalência em Bytes	Potência de 2
KB	Kilobyte	1.024 bytes	2^{10}
MB	Megabyte	1.048.576 bytes	2^{20}
GB	Gigabyte	1.073.741.824 bytes	2^{30}
TB	Terabyte	1.099.511.627.776 bytes	2^{40}
PB	Petabyte	1.125.899.906.842.624 bytes	2^{50}
EB	Exabyte	1.152.921.504.606.846.976 bytes	2^{60}
ZB	Zettabyte	1.180.591.620.717.411.303.424 bytes	2^{70}
YB	Yottabyte	1.208.925.819.614.629.174.706.176 bytes	2^{80}
RB	Ronnabyte	1.237.940.039.285.380.274.899.124.224 bytes	2^{90}
QB	Quettabyte	1.267.650.600.228.229.401.496.703.205.376 bytes	2^{100}

Observação: Em alguns contextos comerciais, o uso decimal (1 KB = 1.000 bytes) é comum, mas o padrão binário, com potências de 2, é o mais utilizado na computação.

Em bases de valor superior a 10, usam-se letras do alfabeto para a representação de algarismos maiores que 9. Uma dessas bases é especialmente importante em computação; trata-se da base 16 ou hexadecimal, por ser de valor múltiplo de 2 (como a base 8).

Nessa base, os “algarismos” A, B, C, D, E e F representam, respectivamente, os valores (da base 10): 10, 11, 12, 13, 14 e 15.

Na base 16 (hexadecimal), dispomos de 16 algarismos (não números) diferentes:

0,1,2,3 , 9, A, B, C, D, E e F

Um número nessa base é representado na forma da expressão de N:

(1^A7B)₁₆

O seu valor na base 10 será obtido usando-se a expressão de N:

$$1 \times 16^3 + 10 \times 16^2 + 7 \times 16^1 + 11 \times 16^0 = 4096 + 2560 + 112 + 11 = 6779_{10}$$

Observemos que na fórmula foram usados os valores 10(para o algarismo A) e 11 (para o algarismo B) para multiplicar as potências de 16. por isso, obtivemos o valor do número na base 10.

Em outras palavras, utilizamos valores e regras de aritmética da base 10 e, por isso, o resultado encontrado é um valor decimal. A tabela 1 mostra a representação de números nas bases 2, 8, 10 e 16.

Base 2	Base 8	Base 10	Base 16
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
10001	21	17	11

Tabela 1: Correlação entre bases

Pela tabela, podemos observar que os dígitos octais e hexadecimais correspondem a combinações de 3 (octais) e 4 (hexadecimais) bits (algarismos binários). Sendo a base desses sistemas de valor maior que a base 2 e tendo em vista essa particularidade na representação de números nas bases 8 e 16 em relação à base 2, verifica-se que é possível converter rapidamente números da base 2 para as bases 8 ou 16, ou vice-versa.

Por exemplo, o número (101111011101)₂, na base 2, possui 12 algarismos (bits), mas pode ser representado com quatro algarismos octais ou com apenas tres algarismos hexadecimais.

$$(101111011101)_2 = (95735)_8$$

$$\text{porque: } 101 = 5; \quad 111 = 7; \quad 011 = 3 \text{ e } 101 = 5$$

$$(101111011101)_2 = (BDD)_8$$

porque: $1011 = B$; $1101 = D$; $1101 = D$

2.3 Conversão de bases

Uma vez entendido como representar números em notação posicional, e como esta notação é aplicável em qualquer base inteira, podemos exercitar a conversão de números de uma base para outra.

Interessa-nos, principalmente, verificar o processo de conversão entre bases múltiplas de 2, e entre estas e a base 10, e vice-versa.

2.3.1 Conversão entre bases Potência de 2

2.3.1.1 Entre as bases 2 e 8

Como $8 = 2^3$, um número binário (base 2) pode ser facilmente convertido para o seu valor equivalente na base 8 (octal). Se o número binário for inteiro, basta dividi-lo, da direita para a esquerda, em grupos de 3 bits (o último grupo não sendo múltiplo de 3, preenche-se com zeros à esquerda). Então, para cada grupo, acha-se o algarismo octal equivalente, conforme mostrado na tabela 1.

A conversão de números da base 8 para a base 2 é realizada de forma semelhante no sentido inverso.

2.3.1.2 Entre as bases 2 e 16

O procedimento de conversão entre números binários e hexadecimais (base 16) é idêntico ao da conversão entre as bases 2 e 8, exceto que, neste caso, a relação é $16 = 2^4$

Desse modo, um algarismo hexadecimal é representado por 4 bits (ver tabela 1); converte-se um número binário em hexadecimal, dividindo-se este número em grupos de 4 *bits* da direita para esquerda.

A conversão de hexadecimal para binário é obtida substituindo-se o algarismo hexadecimal pelos 4 bits correspondentes, de acordo com os valores indicados na tabela 1.

2.3.1.3 Entre as bases 8 e 16

O procedimento de conversão utiliza os mesmos princípios antes apresentados. No entanto, como a base de referência para as substituições de valores é a base 2, esta deve ser empregada como intermediária no processo. Ou seja, convertendo-se da base 8 para a base 16, deve-se primeiro efetuar a conversão para a base 2 (como mostrado nos subitens anteriores) e depois para a base 16. O mesmo ocorre se a conversão for da base 16 para a base 8.

2.3.2 Conversão de números de uma base B para a base 10

A conversão de um número, representado em uma base B qualquer, para seu correspondente valor na base 10 é realizada empregando-se a fórmula de N.

2.3.3 Conversão de Números Decimais para uma Base B

A conversão de números, representados na base 10, para seus valores equivalentes em uma base B qualquer é efetuada através de um processo inverso ao do subitem anterior (base B para base 10).

A conversão é obtida dividindo-se o número decimal pelo valor da base desejada; o resto encontrado é o algarismo menos significativo do valor na base B (mais à direita). Em seguida, divide-se o quociente encontrado pela base B; o resto é o outro algarismo (à esquerda); e assim, sucessivamente, vão-se dividindo os quocientes pelo valor da base até se obter quociente de valor zero. Em cada divisão, o resto encontrado é um algarismo significativo do número na nova base; o primeiro resto encontrado é o valor do algarismo menos significativo e o último resto será o algarismo mais significativo (mais à esquerda).

Na realidade, o algoritmo de conversão pode ser definido com vários critérios de parada, tais como:

- a) enquanto quociente for diferente de zero:
 - dividir dividendo por divisor
 - extrair resto como algarismo e colocá-lo à esquerda do anterior
 - repetir
 - quando quociente for igual a zero, parar.

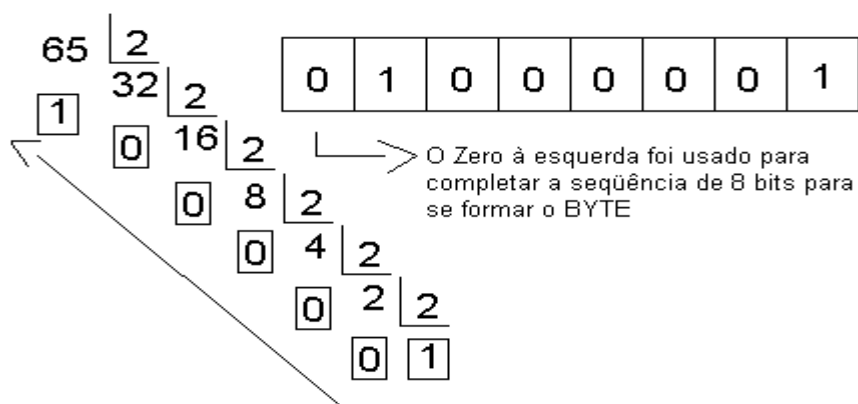
- b) enquanto dividendo for maior que divisor:
 - dividir dividendo por divisor
 - extrair resto como algarismo e colocá-lo à esquerda do anterior
 - repetir

usar o dividendo (que agora é menor que o divisor) como último algarismo à esquerda (algarismo mais significativo).

Vamos seguir os passos para fazer a conversão de decimal para binário:

- 1) Com base no número decimal faz-se divisões sucessivas por 2 (sistema binário) até o número final ser menor que 2 (0 ou 1).

- 2) De cada divisão, o número restante será 0 (zero) ou 1 (um). Escreva estes números logo abaixo do divisor
- 3) O resultado final é obtido da direita (parte inferior) para a esquerda (parte superior)
- 4) Caso o número encontrado não ocupe todos os 8 (oito) bits completa-se o byte com 0 (zeros) a esquerda



Para verificar se a conversão está correta siga os procedimentos abaixo:

- 1) Começando da direita para a esquerda, coloque sobre cada bit o número 2 elevado sucessivamente a cada potência começando do 0 (zero) até 7 (sete).
- 2) Multiplica-se então o valor de cada potência pelo bit correspondente.
- 3) A soma de todos os resultados tem que ser igual ao número que foi dividido: no caso 65.

$$\begin{array}{cccccccc}
 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 \hline
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
 \hline
 0 + 64 + 0 + 0 + 0 + 0 + 0 + 1 = 65
 \end{array}$$

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

Tabela decimal	
10^0	1
10^1	10
10^2	100
10^3	1.000
10^4	10.000
10^5	100.000
10^6	1.000.000
10^7	10.000.000
10^8	100.000.000
10^9	1.000.000.000

Tabela Octal	
8^0	1
8^1	8
8^2	64
8^3	512
8^4	4.096
8^5	32.768
8^6	262.144
8^7	2.097.152

Tabela Hexadecimal	
16^0	1
16^1	16
16^2	256
16^3	4.096
16^4	65.536
16^5	1.048.576
16^6	16.777.216
16^7	268.435.456
16^8	4.294.967.296
16^9	68.719.476.736
16^{10}	1.099.511.627.776
16^{11}	17.592.186.044.416
16^{12}	281.474.976.710.656
16^{13}	4.503.599.627.370.500
16^{14}	72.057.594.037.927.900
16^{15}	1.152.921.504.606.850.000

Tabelas de Conversões
Regra Prática

DE OCTAL PARA BINÁRIO	
OCTAL	BINÁRIO
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

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

Exemplos de Conversão de Bases Numéricas

- Converta o número decimal 25 para binário.

Para converter o número decimal 25 para binário, dividimos o número por 2 repetidamente até o quociente ser 0, anotando o resto a cada divisão. Depois, lemos os restos na ordem inversa.

1. $25 \div 2 = 12$ resto 1
2. $12 \div 2 = 6$ resto 0
3. $6 \div 2 = 3$ resto 0
4. $3 \div 2 = 1$ resto 1
5. $1 \div 2 = 0$ resto 1

Lendo os restos de baixo para cima, obtemos: 11001_2 .

****Resposta:**** $25_{10} = 11001_2$

- Converta o número decimal 45 para hexadecimal.

Para converter o número decimal 45 para hexadecimal, dividimos o número por 16 e anotamos o resto. Continuamos dividindo o quociente até que ele seja 0, e então lemos os restos na ordem inversa.

1. $45 \div 16 = 2$ resto 13 (13 é representado por 'D' no hexadecimal)

O quociente é 2 e o resto é 13. Em hexadecimal, 13 é representado como 'D'. Lendo de baixo para cima, temos $2D_{16}$.

****Resposta:**** $45_{10} = 2D_{16}$

- Converta o número binário 101101 para decimal.

Para converter o número binário 101101 para decimal, multiplicamos cada dígito pelo valor de 2 elevado à posição correspondente, começando do zero da direita para a esquerda:

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 32 + 0 + 8 + 4 + 0 + 1 = 45$$

****Resposta:**** $101101_2 = 45_{10}$

- Converta o número hexadecimal 7F para binário.

Para converter hexadecimal para binário, substituímos cada dígito hexadecimal pelo equivalente de 4 bits binários.

- $7_{16} = 0111_2$
- $F_{16} = 1111_2$

Portanto, $7F_{16} = 01111111_2$.

****Resposta:**** $7F_{16} = 01111111_2$

- Converta o número binário 11011011 para hexadecimal.

Para converter binário para hexadecimal, dividimos o número binário em grupos de 4 bits começando da direita. Se necessário, adicionamos zeros à esquerda.

1101 1011

- $1101_2 = D_{16}$

- $1011_2 = B_{16}$

Portanto, $11011011_2 = DB_{16}$.

****Resposta:**** $11011011_2 = DB_{16}$

Converta o número hexadecimal A3 para decimal.

Para converter hexadecimal para decimal, multiplicamos cada dígito pelo valor de 16 elevado à posição correspondente, da direita para a esquerda.

$$A3_{16} = 10 \times 16^1 + 3 \times 16^0 = 160 + 3 = 163_{10}$$

****Resposta:**** $A3_{16} = 163_{10}$

- Converta o número decimal 255 para binário.

Para converter o número decimal 255 para binário, dividimos o número por 2 repetidamente até o quociente ser 0, anotando o resto a cada divisão.

1. $255 \div 2 = 127$ resto 1

2. $127 \div 2 = 63$ resto 1

3. $63 \div 2 = 31$ resto 1

4. $31 \div 2 = 15$ resto 1

5. $15 \div 2 = 7$ resto 1

6. $7 \div 2 = 3$ resto 1

7. $3 \div 2 = 1$ resto 1

8. $1 \div 2 = 0$ resto 1

Lendo os restos de baixo para cima, obtemos: 11111111_2 .

****Resposta:**** $255_{10} = 11111111_2$

- Converta o número binário 100101 para hexadecimal.

Para converter binário para hexadecimal, dividimos o número em grupos de 4 bits começando da direita. Se necessário, adicionamos zeros à esquerda.

0100 0101

$$- 0100_2 = 4_{16}$$

$$- 0101_2 = 5_{16}$$

Portanto, $100101_2 = 45_{16}$.

****Resposta:**** $100101_2 = 45_{16}$

- Converta o número hexadecimal F4 para decimal.

Para converter hexadecimal para decimal, multiplicamos cada dígito pelo valor de 16 elevado à posição correspondente, da direita para a esquerda.

$$F4_{16} = 15 \times 16^1 + 4 \times 16^0 = 240 + 4 = 244_{10}$$

****Resposta:**** $F4_{16} = 244_{10}$

- Converta o número decimal 1234 para hexadecimal.

Para converter o número decimal 1234 para hexadecimal, dividimos o número por 16 e anotamos o resto. Continuamos dividindo o quociente até que ele seja 0, e então lemos os restos na ordem inversa.

$$1. 1234 \div 16 = 77 \text{ resto } 2$$

$$2. 77 \div 16 = 4 \text{ resto } 13 \text{ (13 é representado por 'D' no hexadecimal)}$$

$$3. 4 \div 16 = 0 \text{ resto } 4$$

Lendo os restos de baixo para cima, temos $4D2_{16}$.

****Resposta:**** $1234_{10} = 4D2_{16}$

2.4 Aritmética Binária e Hexadecimal

Neste item, serão apresentados os procedimentos de adição e subtração de números binários e hexadecimais, inteiros e sem sinal.

2.4.1 Soma Binária

A operação de soma de dois números em base 2 é efetuada de modo semelhante à soma decimal, levando-se em conta, apenas, que só há dois algarismos disponíveis (0 e 1). Assim:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0, \text{ com "vai 1"}$$

2.4.2 Subtração Binária

A subtração em base 2, na forma convencional, usada também no sistema decimal (minuendo – subtraendo = diferença), é relativamente mais complicada por dispormos apenas dos algarismos 0 e 1 e, dessa forma, 0 menos 1 necessita de “empréstimo” de um valor igual à base (no caso é 2), obtido do primeiro algarismo diferente de zero, existente à esquerda. Se estivéssemos operando na base decimal, o “empréstimo” seria de valor igual a 10.

2.4.3 Aritmética Octal (em Base 8)

Consiste em processo semelhante ao da aritmética binária, com exceção do fato de que, neste caso, tem-se 8 algarismos disponíveis. Ocorrerá “vai 1” quando a soma de 2 algarismos for igual ou ultrapassar o valor da base, isto é, 8.

2.4.4 Aritmética Hexadecimal (em Base 16)

Já mencionamos anteriormente que a aritmética com valores expressos em algarismos hexadecimais segue as mesmas regras para qualquer base: somar ou subtrair algarismos por algarismo, utilizando-se de “vai x” na casa à esquerda (e somando-o com as parcelas seguintes à esquerda), ou de “empréstimo” (como nas subtrações em qualquer outra base), e assim por diante.

2.4.5 Conversão com Números Fracionários

A conversão de binário para decimal segue o mesmo processo já estudado, ou seja, efetua-se a soma da série de potências (inclusive as de expoente negativo e que compõem a parte fracionária) para obter-se o resultado.

Observe-se:

$$\begin{aligned} 101.01_{(2)} &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = \\ &= 4 + 1 + \frac{1}{4} = 5,25_{(10)} \end{aligned}$$

$$101.01_{(2)} = 5,25_{(10)}$$

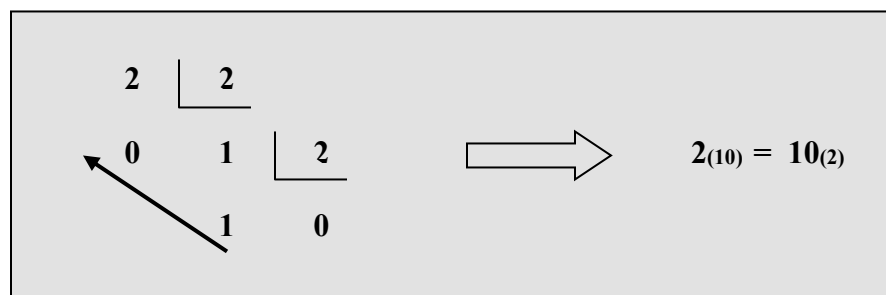
Quando desejamos converter um número decimal fracionário em binário é necessário utilizar um artifício. Para melhor compreender, vamos dividir a conversão em duas etapas e trabalhar com o número $2,6875_{(10)}$ como exemplo:

Separamos a parte inteira do número:

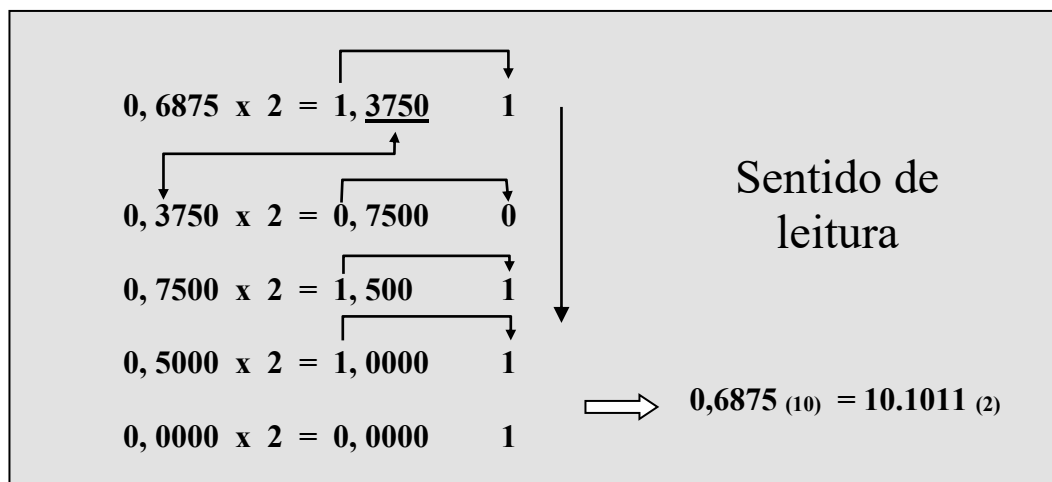
$$2,6875 = 2 + 0,6875$$

↑
parte inteira de 2,6875

Efetuamos a conversão da parte inteira do número decimal para binário.



Vamos agora trabalhar com a parte fracionária do número, efetuando uma série de multiplicações por 2 e isolando as partes inteiras dos resultados das multiplicações.



Convertemos, desta forma, tanto a parte inteira quanto a fracionária do nosso número decimal para binário.

Assim: $2,6875_{(10)} = 10.1011_{(2)}$

2.5 Tipo Numérico

Como os computadores são elementos binários, a forma mais eficiente de representar números deve ser a binária, isto é, converter o número diretamente de decimal para seu correspondente valor binário. A unidade aritmética e lógica (UAL) dos computadores executa operações mais rapidamente se os valores estiverem representados desse modo.

Um dos primeiros problemas ocorridos quando da escolha da representação de números nos computadores consistiu na indicação do sinal do número. Isso foi resolvido com o acréscimo de mais um bit na representação do número; esse bit adicional indica o sinal do número. A convenção adotada de forma universal é:

Valor positivo: bit de sinal igual a zero;
valor negativo: bit de sinal igual a um.

A figura 1 apresenta alguns exemplos de números binários representados com sinal.

O modo mais simples e rápido de efetuar operações sobre números seria, então, transformá-los internamente em números binários, acrescentando-se um bit para o sinal. Um número inteiro com n bits teria disponíveis $n-1$ bits para o seu valor e o restante bit para indicar o sinal. Esse modo de representação é denominado sinal de magnitude.

Valor decimal	Bit de sinal - Valor binário	
+12	01100	(com 5 bits)
-12	11100	(com 5 bits)
-47	100101111	(com 9 bits)
+47	000101111	(com 9 bits)

Figura 1 - Exemplo de números com sinal

Um outro problema reside na forma de representação de números fracionários, devido à dificuldade de representar-se a vírgula (ou ponto de separação entre a parte inteira e fracionária do número) internamente, entre a posição de dois bits.

A solução para esse problema pode ser encontrada através da escolha entre dois modos:

- Representação em ponto fixo (ou vírgula fixa);
- Representação em ponto flutuante (vírgula flutuante).

De modo geral, os sistemas de computação somente representam a vírgula de separação entre as partes inteira e fracionária dos números (ou ponto fracionário, como adotado nos EUA) quando o número é introduzido no sistema como um texto livre. A figura 2 mostra um exemplo de representação de um número pela codificação de cada um de seus algarismos no código ASCII.

+ 37,5 ₁₀	→	00101011	00110011	00110111	00101100	00110101
		+	3	7		5

Figura 2 – Representação de um número em código ASCII

Quando o número é convertido para uma forma binária pura, onde sua magnitude é um valor binário correspondente ao decimal de entrada, então a vírgula fracionária (ou ponto) é assumida em uma determinada posição (definida na declaração do tipo da variável efetuada no corpo do programa-fonte); não há espaço nas células de memória para armazenar um bit de vírgula.

2.5.1 Representação em Ponto Fixo

Esse método consiste na assunção de uma posição fixa para a vírgula (ou ponto). As posições mais adotadas são:

- Na extremidade esquerda do número – nesse caso, o número é totalmente fracionário;
- Na extremidade direita do número – nesse caso, o número é inteiro.

Em qualquer desses casos, no entanto, a vírgula fracionária não estará fisicamente representada na memória; sua posição é determinada na definição da variável, realizada pelo programador (ou pelo compilador), e o sistema memoriza essa posição, mas não a representa fisicamente.

Em geral, os sistemas de computação (e os compiladores da maior parte das linguagens de programação) empregam a representação de números em ponto fixo para indicar apenas valores inteiros (a vírgula fracionária é assumida na posição mais à direita do número); números fracionários são, nesses casos, representados apenas em ponto flutuante.

A figura 3 mostra alguns exemplos de declarações de dados, onde se observa a palavra-chave da linguagem (indica o tipo de dados para o compilador) e a correspondente representação utilizada pelo compilador para o armazenamento e as operações aritméticas a serem eventualmente realizadas com o referido dado.

Na representação de números em ponto fixo, os valores positivos são representados pelo bit zero de sinal. Normalmente esse bit posicionado como o algarismo mais significativo do número, o mais à esquerda, e pelos **n-1** restantes bits, que correspondem ao valor absoluto do número (magnitude).

Linguagem	Tipos de dados	Representação interna
Basic	INTEGER	Ponto fixo (inteiro com 16 bits)
	SINGLE-PRECISION	Ponto flutuante (real com 32 bits)
	DOUBLE-PRECISION	Ponto flutuante (real com 64 bits)
Fortran	INTEGER	Ponto fixo(inteiro)
	REAL	Ponto flutuante (real)
Cobal	COMP	Ponto fixo(inteiro)
	COMP 1	Ponto flutuante (real)
	COMP 2	Ponto flutuante (real)
	COMP 3	Decimal compactado(decimal)
Pascal	INTEGER	Ponto fixo (inteiro)
	REAL	Ponto flutuante (real)

Figura 3– Exemplos de tipos de dados em algumas linguagens

Quanto aos números negativos, o sinal é representado pelo bit um e a magnitude pode ser representada por um dos três seguintes modos:

Sinal e magnitude; Complemento a 1; Complemento a 2;

Conforme verificaremos a seguir, as operações com números negativos (soma com uma das parcelas negativa, ou subtração de números) são difíceis e demoradas, quando realizadas no modo sinal e magnitude. Isso porque é necessário efetuar várias comparações e decisões, em vista da manipulação dos sinais das parcelas para determinação do sinal do resultado

Também mostraremos que as mesmas operações se tornam mais simples e rápidas se realizadas através da aritmética de complemento. Essa é a razão básica do seu emprego em computadores digitais.

Em cada um dos três métodos, não só mostraremos como os números são representados, mas também descreveremos um algoritmo básico utilizado para realizar operações de soma e subtração em cada caso.

2.5.2 Sinal e Magnitude

A representação de números com n algarismos binários (n bits) em sinal e magnitude é obtida atribuindo-se **1 bit** (em geral, na posição mais à esquerda do

número) para indicar o valor do sinal, e os **n-1 bits** restantes para indicarem a magnitude do número, como mostrado na figura 4 (o método de posicionamento dos *bits* é próprio da representação em ponto fixo e, portanto, será também igual em complemento a 1 e complemento a 2).

Nesse tipo de representação, o valor dos bits usados para representar a magnitude (seu valor absoluto) do número é o mesmo, seja o número positivo ou negativo; o que varia é apenas o valor do bit de sinal.

Essa forma de representar números é idêntica ao modo que usamos na vida cotidiana, com a diferença de que usamos símbolos gráficos diferentes dos bits do computador (ver figura 4 e figura 5).



Figura 4 – Exemplo de uma representação de dado em sinal e magnitude

Para representar internamente cada número, a máquina converte o valor absoluto (a magnitude) do número (que deverá estar representado em base decimal) para seu valor correspondente em base 2 e acrescenta um bit à esquerda (que passa a ser o algarismo mais significativo), cujo valor será igual a 0 se o número for positivo (+) ou igual a 1 se o número for negativo (-).

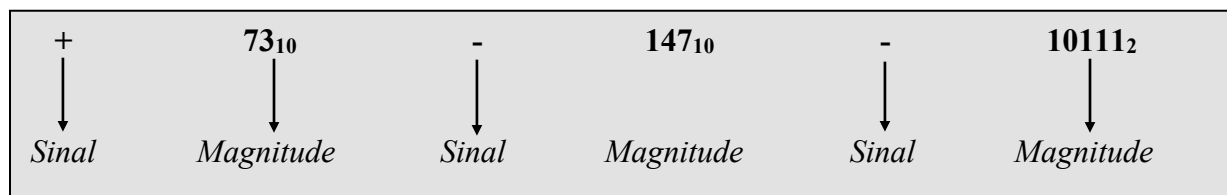


Figura 5 – Representação de valores em sinal e magnitude

2.5.2.1 Limites de Representação

Se os registradores que irão armazenar os valores possuem capacidade para receber **n** algarismos, então a faixa limite de números inteiros, que pode ser armazenada nos referidos registradores, é obtida da expressão.

$$-(2^{n-1}-1) \text{ a } +(2^{n-1}-1)$$

- Se n é a quantidade limite de algarismos, então a magnitude é calculada a partir de $n-1$ algarismos, visto que **1** algarismo é reservado para indicar o sinal do número.
- Do valor obtido (2^{n-1}) subtrai-se **1** (para os valores negativos e positivos) porque o primeiro valor a ser representado é o valor zero (0).

Ainda sobre o modo de representação sinal e magnitude (ver tabela 1), a figura 6 apresenta alguns exemplos da representação binária em sinal e magnitude:

- Possui em duas representações para o zero (matematicamente incorreto), o que é uma desvantagem em relação a outros métodos de representação.
- A representação de números é simétrica entre positivos e negativos, limitada à quantidade permitida de bits dos registradores internos.
- Conforme se observa no exemplo da figura 2, se os números são fracionários, a faixa de representação é mais reduzida, porque o valor $n-1$ refere-se tão-somente à quantidade de algarismos da parte inteira.

2.5.2.2 Aritmética em Sinal e Magnitude

Em sistemas de computação, o sinal é o símbolo de forma idêntica à dos algarismos representativos do seu valor (é um bit igual aos demais), diferentemente da linguagem humana, onde o sinal (= ou -) é um símbolo de forma diversa daquela utilizada para os algarismos que representam o valor do número (0,1,2,5). Além disso, ele é colocado separado da magnitude (+31), enquanto nos computadores o bit de sinal está incorporado aos bits da magnitude, formando um número só.

a)	para um registrador de 6 bits, $n = 6$, e os limites de representação serão:				
de	$-(2^{6-1} - 1)$	a	$+(2^{6-1} - 1)$	ou	
	$-(2^5 - 1)$	a	$+(2^5 - 1)$	-31 a	+31
b)	para um registrador de 16 bits, $n = 16$, e teremos:				
de	$-(2^{16-1} - 1)$	a	$+(2^{16-1} - 1)$	= -32.767	a 32.767
c)	para um registrador de 16 bits, porém representando um número fracionário tendo, por exemplo, 10 bits parte a parte inteira, 5 bits para a parte fracionária e 1 bit para o sinal.				
de	$-(2^{10-1} - 1)$	a	$+(2^{10-1} - 1)$	= $-(2^9 - 1) + (2^9 - 1)$	= - 511 a + 511

Figura 6 – Exemplos de Limites de Representação

Por essa razão, parece que a maneira mais simples de efetuar uma soma seria considerar o número como um inteiro sem sinal; caso contrário, o resultado será incorreto:

$\begin{array}{r} 0100 + 4 \\ 1010 - 2 \\ \hline 1110 - 6 \end{array}$	$\begin{array}{r} 0110 + 2 \\ +1010 - 2 \\ \hline 1100 - 4 \end{array}$
incorreto,	incorreto

Esses exemplos mostram claramente que o bit de sinal não pode, pelo menos nessa forma de representação (sinal e magnitude), ser considerado na operação; ele será necessário apenas para identificar o tipo da operação e o sinal do resultado.

Por isso, as operações aritméticas em sinal e magnitude são efetuadas de modo idêntico ao que fazemos com lápis e papel, isto é, informalmente, executamos um algoritmo para determinar o sinal do resultado e, depois, para efetuar a operação propriamente dita, apenas com as magnitudes.

Em seguida, são descritos os passos de um possível algoritmo para operações aritméticas de soma e subtração, utilizando-se números representados em sinal e magnitude:

Soma

Se ambos os números têm o mesmo sinal, somam-se as magnitudes; o sinal do resultado é o mesmo das parcelas.

- Se os números têm sinais diferentes:
- Identifica-se a maior das magnitudes e registra-se o seu sinal;
- Subtrai-se a magnitude menor da maior (apenas as magnitudes);
- Sinal do resultado é igual ao sinal da maior magnitude.

O problema encontrado pelos fabricantes de computadores na implementação da UAL que efetuasse operações aritméticas com valores representados em sinal e magnitude residiu, principalmente, em dois fatores: *custo* e *velocidade*.

Custo, devido à necessidade de construção de dois elementos diferentes, um para efetuar somas e outro para efetuar subtrações (dois componentes eletrônicos custam mais caro do que um); e *velocidade*, ocasionada pela perda de tempo gasto na manipulação dos sinais, de modo a determinar o tipo da operação e o sinal do resultado.

Além disso, há também a inconveniência da dupla representação para o zero, o que requer um circuito lógico específico para evitar erros de má interpretação. O sistema pode ser programado, por exemplo, para executar uma determinada ação se e somente se o resultado de que uma operação aritmética for igual a zero. Para realizar essa verificação, é feito um teste do sinal do resultado que, sendo -0, bit de sinal igual a 1, redundaria, erroneamente, na não execução da ação desejada.

Embora atualmente a construção de circuitos lógicos complexos seja bastante barata e, considerando também que é desprezível o custo de fabricação de um circuito lógico para efetuar uma operação de subtração em uma UAL, nenhum sistema moderno emprega aritmética em sinal e magnitude, a qual foi definitivamente substituída pela aritmética em complemento a 2 (no caso, é claro, de representação em ponto fixo), cujo descrição será efetuada a seguir.

2.5.3 Representação de Números Negativos em Complemento

O conceito matemático de complemento é válido para qualquer base de numeração. Há dois tipos de complemento: *complemento a base* e *complemento a base menos um*.

Substituindo B pelo valor de uma determinada base, podemos ter, por exemplo:

- Complemento a 10 e complemento a 9 (na base10);
- Complemento a 2 e complemento a 1 (na base2);
- Complemento a 16 e complemento a 15 (na base hexadecimal), e assim por diante.

2.5.3.1 Complemento a Base

Em matemática, o termo complemento significa a quantidade que falta para ‘completar’ um valor, torná-lo completo.

Por exemplo, o complemento de um ângulo agudo é o valor de graus que precisa ser adicionado ao ângulo para se obter 90°, considerando-se, nesse caso, que um ângulo de 90° é completo.

Em operações aritméticas, o complemento a base de um número **N** é o valor necessário para se obter **Bⁿ - N**, onde

n = quantidade de algarismos utilizados na operação;

N = valor do número.

Considera-se, então, que **Bⁿ** é o valor completo de um conjunto de números com **n** algarismos e, por isso, o complemento de cada um é o resultado da subtração de **N** por esse valor (é o que falta a **N** para completar o valor **Bⁿ**). A figura 7 mostra alguns exemplos de complemento a base. Todos os exemplos consideram números com cinco algarismos e, por isso, **n = 5**.

Na base 10:	N = 763₁₀	C10 de N=10⁵ – N = 100 000₁₀ – 763₁₀ = 99237₁₀
Na base 8:	N = 254₈	C8 de N= 85 – N = 100 000₈ – 254₈ = 77524₈
Na base 2:	N = 110₂	C2 de N= 25 – N = 100 000₂ – 110 = 11010₂

Figura 7 – Exemplos de valores representados em complemento a base

Na prática, a obtenção mais rápida do valor do complemento a base de um número é realizada através da seguinte operação, em duas etapas:

- Subtrair cada algarismo do maior algarismo da base considerada (9 na base 10, 1 na base 2, etc).
- Ao resultado encontrado somar 1 ao algarismo menos significativo (o mais à direita).

Os complementos dos números utilizados como exemplos na figura 7 podem ser calculados por esse novo método, assim:

$$\begin{array}{lll} \mathbf{N = 763_{10}} & \mathbf{C10 \text{ de } N = 99\,999 - 763 = 99\,236_{10} + 1 = 99237_{10}} \\ \mathbf{N = 254_8} & \mathbf{C8 \text{ de } N = 77\,777 - 254 = 77\,523_8 + 1 = 77254_8} \\ \mathbf{N = 110_2} & \mathbf{C2 \text{ de } N = 11\,111 - 110 = 11\,001_2 + 1 = 11010_2} \end{array}$$

A primeira etapa desse método consiste na operação de obtenção do complemento a base -1 , descrito no item Complemento a base menos 1.

Quando se trata de valores na base 2 (números binários), pode-se executar a primeira etapa da obtenção do complemento a 2 de um número através de um outro método ainda mais rápido que o anterior:

- Em vez de se subtrair cada algarismo do número do maior algarismo da base (no caso de base 2, trata-se de subtrair de 1), simplesmente inverte-se o valor de cada algarismo, isto é, se o algarismo é 0, passa a ser 1, e se for 1, passa a ser 0. A segunda etapa (somar 1 ao resultado) permanece a mesma.

A figura 8 mostra alguns exemplos desse método (todos os números estão representados na base 2).

Como o que interessa é o trabalho interno nos computadores e esses são elementos binários, detalhes da representação e de operações aritméticas em complemento a 2 são mais importantes que complemento a base em geral.

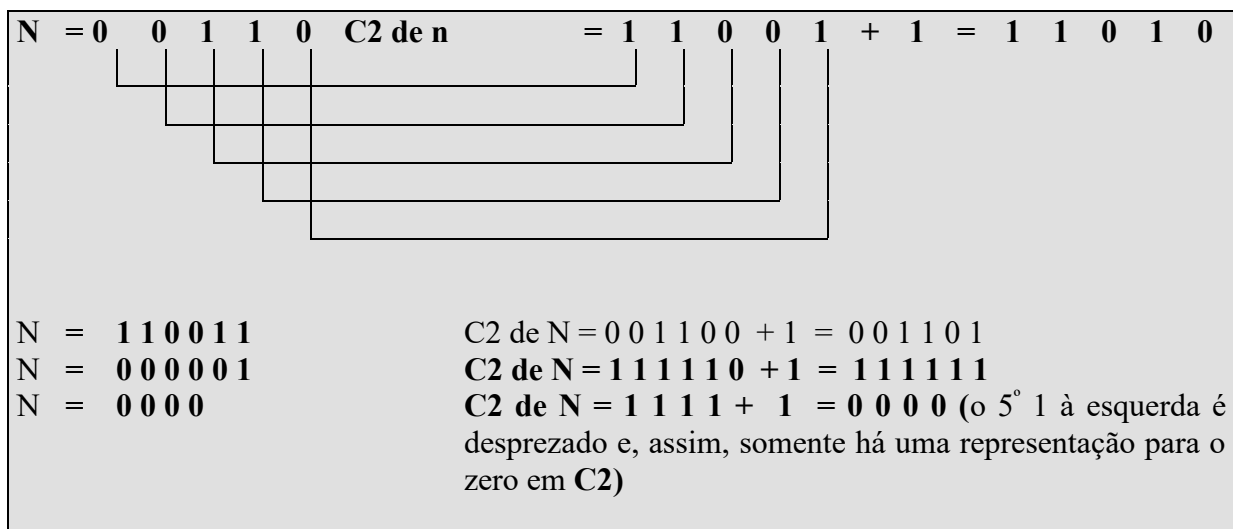


Figura 8 – Método rápido de obtenção de complemento de um número

Pode-se dizer, sem risco de erro, que a quase totalidade dos computadores modernos utilizam aritmética de complemento a 2 (quando se trata de representação em ponto fixo), devido a duas grandes vantagens daquele método sobre sinal e magnitude, e mesmo sobre complemento a 1 (como veremos a seguir):

- Possuir uma única representação para o zero;
- Necessitar de apenas um circuito somador para realizar, não só operações de soma, como também operações de subtração (mais barato).

Conforme já pudemos observar no último exemplo da figura 8, a obtenção do complemento a 2 do valor 0 redundava no mesmo valor 0 sem troca de sinal; não há, portanto, duas representações para o zero, como acontece na representação em sinal e magnitude e complemento a 1.

Seja, por exemplo, um computador com um registrador de trabalho com capacidade para armazenar números de 6 bits. Nesse caso, representa-se o valor zero como:

000000 o 1º 0 indica o sinal do número.

Para calcular o complemento a 2 desse número, vamos usar o método rápido (da figura 7):

000000 \rightarrow **111111 + 1 = 000000**

O 'vai 1' para a 7ª ordem (à esquerda) é desprezado, porque consideramos o limite de 6 bits para o registrador. O valor final é igual ao inicial – uma única representação para o 0.

Isso acarreta uma assimetria na quantidade de números que podem ser representados em complemento a 2, permitindo que se tenha a representação de

um número negativo a mais do que os números positivos. No caso exemplificado (com um registrador de 6 bits), teremos:

2^6 valores representáveis = 64 números binários.

Desses 64 números, 32 iniciam com 0 e 32 iniciam por 1. Os 32 números, cujo 1º bit à esquerda é 0, representam 31 valores positivos (de +1 a +31) e o zero (000000), enquanto os 32 números com 1º bit igual a 1 representam 32 valores negativos (-1 a -32).

Então, o número -32 não tem correspondente positivo (+32).

Pode-se generalizar para qualquer quantidade de bits nos registradores, definido a faixa de representação de números em complemento a 2.

$$- 2^{n-1} \text{ a } + (2^{n-1} - 1)$$

Utilizando, por exemplo, um registrador de 6 bits, teremos $n=6$ e $n-1=5$; a faixa de representação será:

$$\text{de } - 32 \text{ a } +31$$

(+32 não possui representação para essa quantidade de bits)

2.5.3.2 Aritmética com complemento

Na aritmética (e representação de números) com complemento, o sentido de negatividade do número não é dado por um símbolo que indica o sinal e é acrescentado ao número, mas que não é incluído na operação aritmética propriamente dita, como na representação em sinal e magnitude (símbolo + ou 0 e - ou 1). Em complemento a 2 (ou mesmo a 1), a negatividade é incorporada ao próprio número e, nesse caso, nas operações de soma e subtração, o número completo é usado (sem se dispensar o algarismo do sinal).

O ponto fundamental (diferença entre a aritmética de S&M e de C2) é que o complemento de um número N também representa a sua forma com o sinal inverso. Ou seja:

Complemento de $N = -N$ e complemento de $-N = N$

Em consequência, podemos estabelecer as seguintes conclusões:

1. $N1 + N2$ é efetuada como uma soma comum, algarismo por algarismo (inclusive o algarismo de sinal – para a aritmética binária). Os números

negativos devem estar previamente representados em complemento a base.

2. $N1 - N2 = N1 + \text{compl. a base de } N2$ ou $N1 + N2$

- O traço horizontal sobre **N2** significa que se trata da representação do complemento (a base (C_8)) de $N2$;
- A operação de soma se realiza de modo igual a o especificado no item 1.

Podemos demonstrar (e para isso utilizaremos como exemplos a aritmética decimal – mais bem entendida por nós) que operações aritméticas de soma e subtração podem ser efetuadas apenas empregando-se somas entre os números, desde, é claro, que esses estejam representados em complemento e que se utilize a aritmética de complemento.

Para simplificar, consideremos números decimais expressos com quatro algarismos para:

Exemplo 1

Somar **(+15)** e **(-12)**

Soma algébrica: $(+15) + (-12) = (+15) - (+12) = +3$

Em **C10**: $+15 = 0015$ e $-12 = 9988$

Somando em C10, efetua-se a operação algarismo por algarismo. Se ocorrer “vai 1” na soma dos dois algarismos mais significativos, ele será desprezado.

$$\begin{array}{r} \begin{array}{|l} \rightarrow 1 \\ \rightarrow \end{array} \begin{array}{r} 0015 \\ 9988 \\ \hline 0003 \end{array} \longrightarrow \text{Resultado: } = +3 \\ \text{“vai 1” desprezado} \end{array}$$

2.5.3.3 Aritmética em complemento a 2

Aritmética em complemento a 2 requer apenas um circuito para somar dois números e um circuito que realize a operação de complementação. O algoritmo básico refere-se, então, “a soma dos números, considerando-se que os negativos estejam representados em complemento a 2; ele acusa, também, se o resultado ultrapassar a quantidade de bits representáveis na UAL (e registradores), e que denominamos *overflow*,”

Algoritmo da soma em complemento a 2

- Somar os dois números, bit a bit, inclusive o bit de sinal;
- Desprezar o último “vai 1” (para fora do número), se houver;
- Se, simultaneamente, ocorrer “vai 1” para o bit de sinal e para fora de número, ou se ambos não existirem, o resultado está correto;

Se ocorrer apenas um dos dois “vai 1” (ou para o bit de sinal ou para fora), o resultado está incorreto. Ocorreu um *overflow*.

Vamos consolidar o entendimento dessa aritmética simples e correta tendo sempre em mente que:

- As operações de soma são normalmente realizadas como soma.
- As operações de subtração são realizadas como soma de complemento.
- Se o resultado encontrado é um valor positivo, então o valor decimal correspondente da magnitude é obtido por pura conversão de base **2** para base **10**.
- Se o resultado encontrado é um valor negativo, deve-se primeiro converter esse valor para representação de sinal e magnitude e, em seguida, converter a magnitude de base **2** para base **10**.

2.5.4 Ponto Flutuante - Padrão (IEEE 754)

Na representação de grandezas podemos ter uma faixa de variação dos números muito grande. Exemplo:

1. Massa do Elétron: 9×10^{-28} gramas
2. Massa do Sol: 2×10^{33} gramas
3. Faixa de variação: $> 10^{60}$

Exemplo de Representação

- [illegible]

Deve-se usar um sistema de representação de maneira que a faixa de variação dos números seja independentemente do número de dígitos significativos dos números representados.

► Solução: Notação Científica

Número x Base^{expoente}

Assim, existem várias formas de representação que “flutuam” a vírgula mudando o expoente:

- ▶ Exemplo:
 - $2,14 = 0,0214 \times 10^2 = 214 \times 10^{-2}$

Forma Normalizada: Único dígito diferente de zero antes da vírgula. Só existe uma forma de representar um número.

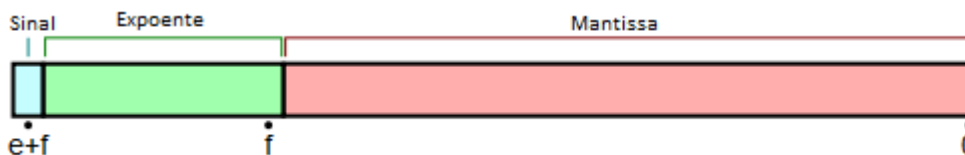
O ponto flutuante é o formato de representação digital de números reais usado nos computadores.

- ▶ Na representação normalizada binária há apenas um “1” antes da vírgula.
- ▶ Tudo é armazenado em base 2.

Exemplo:

- $1,01101 \times (10)101$
- Mantissa = 1,01101
- Expoente = 101

Caso a forma normalizada seja usada, o “1” antes da vírgula pode ficar implícito economizando um bit. É o chamado **bit escondido**. O número de bits para representar a mantissa e o expoente depende da norma.



Até a década de 1980, cada fabricante de computador tinha seu próprio formato de representação de ponto flutuante. O IEEE: *Institute of Electrical and Electronic Engineers*. Organização composta por engenheiros, cientistas e estudantes, que desenvolvem padrões para a indústria de computadores e eletro-eletrônicos. O padrão IEEE 754 foi inventado para padronizar essa forma de representação.

A primeira versão desse padrão data de 1985 e perdurou por 23 anos até ser substituída em agosto de 2008 por uma outra versão da IEEE 754.

- ▶ Padrão mais usado para computação de ponto flutuante
- ▶ Formatos e aritmética binária são preservados no IEEE 754-2008

São 4 os formatos de representação de valores de ponto flutuante:

- ▶ Precisão simples (32 bits)
- ▶ Precisão dupla (64 bits)

- ▶ Precisão simples estendida (≥ 43 bits, não é comumente usada)
- ▶ Precisão dupla estendida (≥ 79 bits, usualmente implementada com 80 bits)
- ▶ Apenas a precisão simples é requerida pelo padrão, as outras são opcionais.

Ocorre a polarização do Expoente

- ▶ O expoente é polarizado por: $(2^{e-1}) - 1$
e: Número de bits do expoente

Representação de um número com expoente 25 numa representação com 8 bits para expoente:

- ▶ $25 + (2^{8-1}) - 1 = 25 + 128 - 1 = 152$

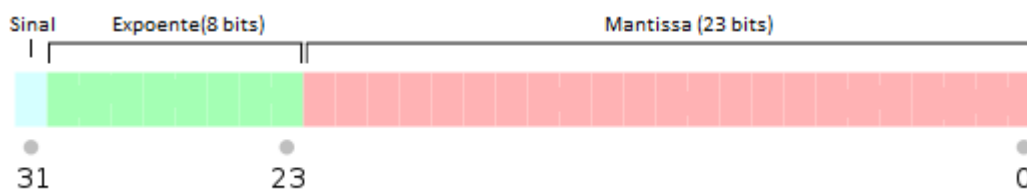
A polarização é realizada pois os expoentes têm que possuir valores com sinal para poder representar valores grandes e pequenos. Portanto, o expoente é polarizado antes de ser armazenado ajustando seu valor para colocá-lo dentro de uma faixa sem sinal, adequado para comparação.

O bit mais significativo da mantissa não é armazenado, porém ele pode ser determinado pelo valor do expoente polarizado.

▶ Casos

1. Se $0 < \text{expoente} < 2^e - 1$, então o bit mais significativo é 1 e o número é dito **normalizado**.
2. Expoente = 0 e Mantissa $\neq 0$, número **não normalizado**.
3. Expoente = 0 e Mantissa = 0, ± 0 depende do bit de sinal.
4. Expoente = $2^e - 1$ e Mantissa = 0, $\pm \infty$ depende do bit de sinal.
5. Expoente = $2^e - 1$ e Mantissa $\neq 0$, NaN.

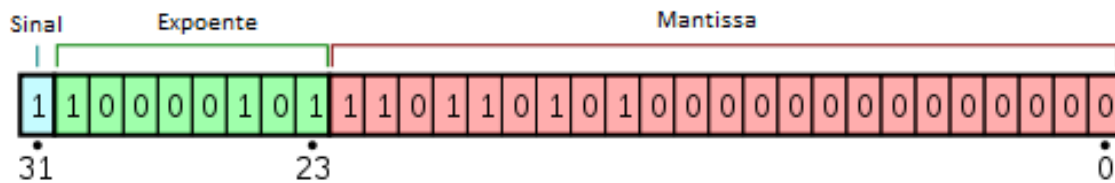
Precisão simples - Polarizado com 127



- ▶ Expoente vai de -126 a +127.
- ▶ -127 não pode pois significa número não normalizado ou zero.
- ▶ 128 seria polarizado para 255, não pode é NaN (não é um número) ou infinito

O valor do número é: $v = s \times 2^e \times m$

Exemplo: Como representar o número -118.625 no padrão IEEE 754?

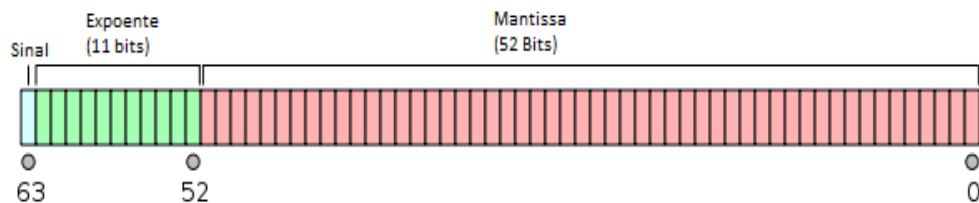


Para transformar de volta:

- ▶ Bit 1 no sinal indica número negativo.
- ▶ Expoente $10000101 = 133_{10}$
 - Portanto o valor antes da polarização era:
 - $x + (2^{8-1}) - 1 = 133$
 - $x + 127 = 133$
 - $x = 6$
 - Com o bit escondido temos a mantissa:
 - $1.110110101 \times 2^6 = 1110110.101 = 118.625$
 - Como o bit de sinal representa um número negativo temos:
 - -118.625

Precisão dupla

- Essencialmente a mesma coisa da precisão simples só que os campos são maiores. Polarizado com 1023



O **IEEE 754-2008** inclui praticamente todo o IEE 754-1985 e o IEE 854-1987 que generalizava o IEEE 754-1985 para cobrir aritmética decimal além da binária. Define:

- Formatos Aritméticos
- Formatos de Intercâmbio
- Algoritmos de Arredondamento
- Operações
- Manuseio de Exceções

As operações requeridas pela norma são:

- Operações Aritméticas (adição, subtração, multiplicação, divisão, raiz quadrada, etc.)
- Conversões (entre formatos)
- Dimensionamento e quantificação
- Cópia e manipulação do sinal (abs, negate, etc.)
- Classificação e teste para NaNs

- Testes e definição de sinalizadores
- Operações diversas

Uma nova cláusula dessa norma recomenda 50 operações incluindo log, potenciação, etc. Porém todas são opcionais.

Portanto, podemos utilizar o padrão dividindo-o nas etapas seguintes do Exemplo 1: Converter 25,5 em binário.

1ª Etapa: Transformar o número em algo parecido com $1,### \times 2^{###}$

Isso é alcançado através de divisões ou multiplicações. No caso do exemplo, divisões, pois o número é maior que 1.

Nesta etapa o sinal de negativo do número, caso exista, deve ser ignorado!

$$\begin{aligned} 25,5 / 2 &= 12,75 \\ 12,75 / 2 &= 6,375 \\ 6,375 / 2 &= 3,1875 \\ 3,1875 / 2 &= 1,59375 \end{aligned}$$

Ao final desta etapa, obtemos $R1 = 1,59375$ através de **4 divisões**.

Com isso, apenas encontramos outra forma de representar

25,5

que é

$$1,59375 \times 2^4$$

2ª Etapa: Calcular a mantissa baseado na parte fracionária de $R1$

Esse processo se dá sempre através de multiplicações sucessivas, até que o resultado seja 0, ou até um máximo de 23 multiplicações.

Como $R1 = 1,59375$, logo, sua parte fracionária é 0,59375

$0,59375 \times 2 = 1,1875$ (De uma etapa para a outra, leva-se apenas a parte fracionária!)

$$0,1875 \times 2 = 0,375$$

$$0,375 \times 2 = 0,75$$

$$0,75 \times 2 = 1,5$$

$$0,5 \times 2 = 1,0$$

$$0,0 \times 2 = 0,0 \quad (\text{Acabou!})$$

Mantissa calculada = **100110**

Como a quantidade de bits da mantissa ficou inferior à 23, completa-se o final com 0's até formar 23 dígitos.

Mantissa final = **10011000000000000000000**

3ª Etapa: Coletar as informações necessárias para criar a representação binária

- Sinal

Positivo = 0 (Se fosse negativo seria 1)

- Expoente

4 (Calculado na 1ª etapa)

Porém, devido ao padrão estabelecido pela IEEE para números de ponto flutuante, esse expoente deve ser acrescido de 127. Assim:

4 + 127 = 131 = 10000011

- Mantissa

100110000000000000000000 (Calculada na 2ª etapa)

4ª Etapa: Montar a representação binária

Basta juntar o sinal, o expoente e a mantissa coletados na 3ª etapa, para montar o valor de 32 bits que representa o valor inicial.

Assim, a representação final é:

01000001110011000000000000000000

Ou 01000001110011000000000000000000 (Sem a formatação de cores)

3. ÁLGEBRA DE BOOLE

3.1 Noções de Teoria dos Conjuntos

O conjunto é entendido, intuitivamente, como um grupo bem determinado de elementos, não importa de que natureza, todos eles com pelo menos uma característica comum.

Um conjunto é representado, geralmente, por uma letra maiúscula e os seus elementos, quando letras, são representados por minúsculas e entre duas chaves.

Por exemplo: Vamos considerar o conjunto dos números naturais e representá-lo pela letra N .

Então: $N = \{ 1, 2, 3, 4, 5, 6, \}$

Vamos agora formar um outro conjunto com alguns elementos do conjunto N e que possuam, por exemplo, a característica comum de serem números naturais pares. Representaremos este conjunto pela letra A .

Então: $A = \{ 2, 4, 6, 8, \}$

Dado um conjunto X qualquer. Dizemos que Y é um sub-conjunto de X se todos os elementos de Y também pertencem ao conjunto X .

Como você pode observar, no exemplo anterior, A é um sub-conjunto do conjunto N .

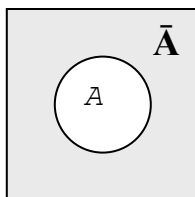
Dado um conjunto X qualquer e um sub-conjunto Y de X , dizemos que \bar{Y} é o complementar de Y em relação a X , se todos os elementos de \bar{Y} pertencem a X e não pertencem a Y .

Existem, basicamente, três operações lógicas na teoria dos conjuntos:

- Complemento, Negação ou Inversão;
- Intersecção ou Produto;
- União ou Adição.

3.1.1 Complemento, Negação ou Inversão

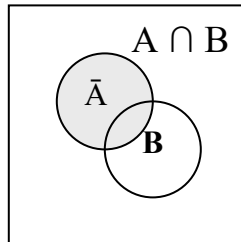
Dado um conjunto A de S , associamos a **negação** de A ao conjunto de elementos contidos em S e não contidos em A . Representamos essa negação por \bar{A} e lemos complemento de A , como já foi visto anteriormente.



Por exemplo: se **S** representa o número de alunos em uma sala de aula, o conjunto **A** pode ser, por exemplo, o conjunto de alunos com mais de 20 anos, e \bar{A} o conjunto de alunos com menos de 20 anos.

3.1.2 Intersecção ou Produto

Dados dois conjuntos **A** e **B**, chamamos intersecção de **A** e **B** ao conjunto formado pelos elementos comuns a **A** e **B**. Indicamos esse conjunto por $A \cap B$ ou **A, B**, ou **AB** e lemos **A inter B**.



3.1.3 União ou Adição

Dados dois conjuntos **A** e **B**, chamamos união de **A** e **B** ao conjunto formado pelos elementos que pertencem a **A** ou **B**. Indicamos esse conjunto por $A \cup B$ ou **A + B** e lemos **A união B**.

3.1.4 Propriedades

Através dos diagramas de *Venn* é possível demonstrar as propriedades relativas às operações fundamentais de intersecção e união.

Essas propriedades são:

- Comutativa
- Associativa
- Distributiva

3.1.4.1 Propriedade Comutativa

Se **A** e **B** pertencem ao conjunto **S**, então:

- a . 1) $A \cup B = B \cup A$ ou $A + B = B + A$
b . 2) $A \cap B = B \cap A$ ou $AB = BA$

3.1.4.2 Propriedade Associativa

Se A, B, C pertencem ao conjunto S, então:

$$\begin{aligned} \text{b. 1) } A \cup (B \cap C) &= (A \cup B) \cap C \quad \text{ou} \quad A + (B \cdot C) = (A + B) \cdot C \\ \text{b. 2) } A \cap (B \cup C) &= (A \cap B) \cup C \quad \text{ou} \quad A \cdot (B + C) = (A \cdot B) + C \end{aligned}$$

3.1.4.3 Propriedade Distributiva

Se A, B, C pertencem ao conjunto S, então:

$$\begin{aligned} \text{c. 1) } A \cup (B \cap C) &= (A \cup B) \cap (A \cup C) \quad \text{ou} \quad A + (B \cdot C) = (A + B) \cdot (A + C) \\ \text{c. 2) } A \cap (B \cup C) &= (A \cap B) \cup (A \cap C) \quad \text{ou} \quad A \cdot (B + C) = (A \cdot B) + (A \cdot C) \end{aligned}$$

3.2 Noções de Álgebra Booleana

Álgebra *Booleana* é uma área da matemática que trata de regras e elementos de *Lógica*. O nome *booleana* é uma retribuição da comunidade científica ao matemático inglês George Boole (1815-1864), que desenvolveu uma análise matemática sobre a Lógica. Em 1854, ele publicou o famoso livro “*An Investigation of the Laws of Thought on Which are founded the Mathematical Theories of Logic and Probabilities*” (Uma investigação das leis do pensamento nas quais estão alicerçadas teorias matemáticas de lógica e probabilidade), no qual ele propôs os princípios básicos dessa álgebra.

Talvez a álgebra booleana se tornasse apenas uma ferramenta de filosofia se não tivesse ocorrido o desenvolvimento tão acentuado da eletrônica neste século e a grande utilização da lógica digital nesse processo.

Em 1938, Claude Sannon, no M.I.T. (*Massachusetts Institute of Technology*), utilizou os conceitos desta álgebra para o projeto de circuitos de chaveamento (“*switching circuits*”) que usavam relés. Tais circuitos estavam ligados a centrais de comutação telefônica e mais tarde, a computadores digitais. Atualmente, esta álgebra é largamente empregada no projeto de circuitos digitais, para:

- **Análise** – é um método prático e econômico de descrever as funções de um circuito digital e, conseqüentemente, seu funcionamento: e
- **Projeto** – ao identificar a função a ser realizada por um circuito, a álgebra *booleana* pode ser aplicada para simplificar sua descrição e, assim, também sua implementação.

O projeto de elementos digitais está relacionando com a conversão de ideias em hardware real, e os elementos encontrados na álgebra *booleana* permitem que uma ideia, uma afirmação, possa ser expressa matematicamente. Permitem também que a expressão resultante da formulação matemática da ideia possa ser

simplificada e, finalmente, convertida no mundo real do hardware de portas lógica (“*gates*”) e outros elementos digitais.

Como na álgebra comum, a álgebra booleana trata de variáveis e operações a serem realizadas com estas. A diferença é que, no que se refere à álgebra booleana, as variáveis usadas são binárias, tendo apenas dois valores possíveis, VERDADE (equivalente ao bit **1**) e FALSO (equivalente ao bit **0**). Da mesma forma que o bit só possui dois valores, também um relé ou uma porta lógica pode estar em uma de duas possíveis posições: ou na posição ABERTO (=bit **1**) ou na posição FECHADO (=bit **0**).

Por exemplo, observamos a expressão de uma ideia:
*“A lâmpada acenderá se o sinal **A** estiver presente”.*

Esta afirmação implica a dependência de uma ação sobre outra. Se criássemos o indicador **L** para o fato de a lâmpada acender e o indicador **A** para o sinal **A**, poderíamos estabelecer uma equação simbólica para expressar a afirmação acima (ver figura 1a).

$$L = A \text{ (à direita)} \quad \text{e} \quad L \neq A \text{ (à esquerda)}$$

Poderíamos também modificar a expressão para indicar:

*“A lâmpada acenderá se a chave **A** E a chave **B** estiverem fechadas”.*

Neste caso, a expressão será diferente, aparecendo a conexão E (AND), conforme exemplo, na figura 1(b), de um circuito em série: **L = A AND B**

Ou ainda a expressão:

*“A lâmpada somente acenderá se o sinal **A** NÃO estiver presente”.*

E a expressão correspondente será:

$$L = \text{NOT } A$$

Além de utilizar variáveis que possuem apenas 2 valores, 0 e 1, a álgebra booleana define a existência de três operações fundamentais (as demais operações booleanas são combinações das três fundamentais):

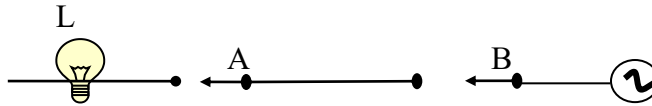
AND (representada pelo ponto da multiplicação aritmética)

OR (representada pelo sinal + da adição aritmética)

NO (representada por uma barra em cima da variável – terminologia ANSI – ou por apóstrofo à direita da variável)



(a) exemplo de afirmação expressa por uma equação lógica



(b) exemplo de afirmação com dupla ideia

Figura 1 – Exemplo de circuitos que implementam ideias.

Tendo em vista que livros, revistas e manuais, principalmente na área de computação, usam termos um tanto ou quanto diferentes para identificar as operações lógicas **AND**, **OR** e **NOT**, parece interessante esclarecer ao leitor, mencionando estes termos. Já vimos que a operação **OR** é representada pelo símbolo de + e que sua tabela verdade consiste nas linhas a seguir:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 1 \end{aligned}$$

Quem observa essas expressões, à primeira vista pode pensar que se trata de soma aritmética e, não fosse pela última operação, que os resultados são iguais mesmo aos de uma soma aritmética. É por isso que alguns autores chamam a operação lógica **OR** de soma lógica e outros criaram diferentes símbolos para não confundirmos com soma. Símbolos como U e V (usados também em teoria dos conjuntos).

Assim, usar-se-ia:

$$A \cup B \cup C \text{ em vez de } A + B + C$$

Mas o pessoal de computação (e os livros e manuais estão aí para confirmar isso) continua preferindo usar o símbolo +, talvez por uma deferência a mais com George Boole, já que foi ele quem propôs originalmente o símbolo.

O mesmo acontece com a operação **AND** e seu símbolo. Como o símbolo da referida operação (ponto ".") também representa uma operação de multiplicação aritmética e os resultados da tabela verdade são idênticos aos da multiplicação, chama-se também a operação lógica **AND** de multiplicação lógica.

A tabela 1 apresenta todas as regras básicas da álgebra booleana.

1. $X + 0 = X$	12. $X \cdot Y = Y \cdot X$
2. $X + 1 = 1$	13. $X + (Y + Z) = (X + Y) + Z$
3. $X + X = X$	14. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
4. $X + \bar{X} = 1$	15. $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$
5. $X \cdot 0 = 0$	16. $X + X \cdot Z = X$
6. $X \cdot 1 = X$	17. $X(X + Y) = X$
7. $X \cdot X = X$	18. $(X + Y) \cdot (X + Z) = X + Y \cdot Z$
8. $X \cdot \bar{X} = 0$	19. $X + \bar{X} \cdot Y = X + Y$
9. $\bar{\bar{X}} = X$	20. $X \cdot Y + Y \cdot Z + \bar{Y} \cdot Z = X \cdot Y + Z$
10. $X + Y = Y + X$	21. $\overline{(X + Y)} = \bar{X} \cdot \bar{Y}$
11. $X \oplus X = 0$	22. $\overline{(X \cdot Y)} = \bar{X} + \bar{Y}$

Tabela 1

4. CONCEITOS DA LÓGICA DIGITAL

4.1 Introdução

Um computador digital é uma máquina projetada para armazenar e manipular informações representadas apenas por algarismos ou dígitos e que só podem assumir dois valores distintos, 0 e 1, razão por que são chamados computadores digitais, sistemas digitais ou simplesmente máquinas digitais binárias. Como na prática não há máquinas digitais não binárias, como, por exemplo, máquinas digitais decimais, é mais usual simplificar-se o termo, usando apenas computador digital (a palavra binário fica implícita).

A informação binária (valores 0 ou 1) é representada em um sistema digital por quantidades físicas, sinais elétricos, os quais são gerados e mantidos internamente ou recebidos de elementos externos, em dois níveis de intensidade, cada um correspondente a um valor binário (há outras formas de armazenamento de bits internamente em um computador, como campo magnético e sinais óticos).

A figura 1 mostra um exemplo de valores elétricos de sinais binários, sendo escolhido um sinal de +3V para representar o bit 1 e +0,5V para representar o valor binário 0. Como se observa na figura, cada valor tem uma faixa de tolerância, tendo em vista que nenhum sinal é sempre absolutamente preciso em seu valor.

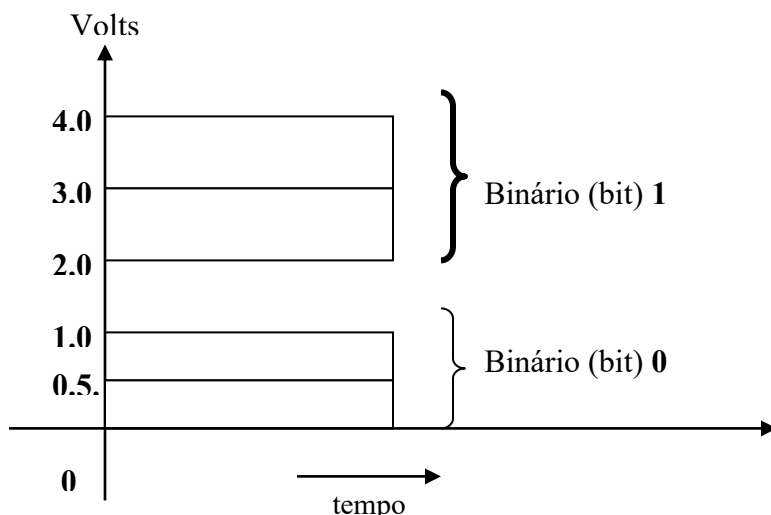


Figura 1 – exemplo de um sinal binário

Internamente, um computador é fabricado com circuitos eletrônicos que precisam armazenar os sinais binários e realizar certos tipos de operações com eles. Esses circuitos, chamados circuitos digitais, são formados de pequenos elementos capazes de manipular grandezas apenas binárias. Os pequenos elementos acima referidos são conhecidos como portas (“*gates*”) lógicas, por

permitirem ou não a passagem destes sinais, e os circuitos que contêm as portas lógicas são conhecidos como circuitos lógicos.

Uma porta ("gate") é, então, um elemento de hardware (é um circuito eletrônico) que recebe um ou mais sinais de entrada e produz um sinal de saída, cujo valor é dependente do tipo de regra lógica estabelecida para a construção do referido circuito.

Em resumo, um computador digital é fabricado, então, contendo uma infinidade de circuitos lógicos ou portas, convenientemente distribuídos e organizados, de modo que alguns servirão para armazenamento de valores, outros permitirão e controlarão o fluxo de sinais entre os componentes e outros, ainda, serão utilizados para realizar operações matemáticas.

O projeto de circuitos digitais e a análise de seu comportamento em um computador podem ser realizados através do emprego de conceitos e regras estabelecidas por uma disciplina conhecida como Álgebra de Chaveamentos ("*Switching* Álgebra"), que é um ramo da álgebra booleana ou álgebra moderna.

Neste capítulo vamos apresentar noções sobre portas lógicas e as operações que podem ser realizadas com estes elementos, bem como alguns aspectos de lógica digital que sejam considerados importantes.

4.2 Portas e Operações Lógicas

Uma porta lógica ("*gate*") é um circuito eletrônico, portanto uma peça de hardware, que se constitui no elemento básico e mais elementar de um sistema de computação. Grande parte do hardware do sistema é fabricado através da adequada combinação de milhões desses elementos, como a UCP, memórias principais e *cache*, interfaces de E/S e outros.

Há diversos tipos bem definidos de portas lógicas, cada uma delas sendo capaz de implementar uma operação ou função lógica específica. Uma operação lógica (de modo semelhante a uma operação algébrica) realizada sobre um ou mais valores lógicos produz um certo resultado (também um valor lógico), conforme a regra definida para a específica operação lógica (ver figura 2).

Assim como na álgebra comum (a que estudamos no 2º grau) é necessário definir símbolos matemáticos e gráficos para representar as operações lógicas (e os operadores lógicos). A figura 3 mostra os símbolos matemáticos e gráficos referentes às operações lógicas (porta) que iremos analisar neste item.

Como já foi citada, uma operação lógica produz um resultado que pode assumir somente dois valores, **0** ou **1**, os quais são relacionados na álgebra *booleana* às declarações **FALSO (F=0)** ou **VERDADEIRO (V=1)**. Se as variáveis de entrada só podem assumir os valores F (falso) ou 0 ou V (verdade) ou 1 e se o resultado também, então podemos definir previamente todos os possíveis valores de resultado de uma dada operação lógica, conforme a combinação possível de valores de entrada. Essas possibilidades são representadas de forma tabular e chama-se o conjunto de Tabela Verdade. Cada operação lógica possui sua própria tabela verdade, estabelecida de acordo com a regra que define a respectiva operação lógica.

Uma tabela verdade tem, então, tantas linhas de informação quantas são as possíveis combinações de valores de entrada, o que pode variar conforme a quantidade de diferentes valores de entrada que se tenha.

Assim, se tivermos apenas um valor de entrada, então a saída só pode assumir dois valores (já que a variável de entrada só pode assumir dois valores distintos, $F=0$ ou $V=1$) e, nesse caso, a tabela verdade teria duas linhas, uma para a entrada igual a **0** e outra para entrada igual a **1**. Se, por outro lado, fossem definidas duas entradas, então haveria quatro possíveis combinações dos valores de entrada (**00, 01, 10, 11**), pois $2^2 = 4$ e a tabela verdade possuiria quatro linhas e assim por diante.

De modo geral, a tabela verdade de uma dada operação lógica possui 2^n linhas ou combinações de valores de entrada, sendo n igual à quantidade de entrada.

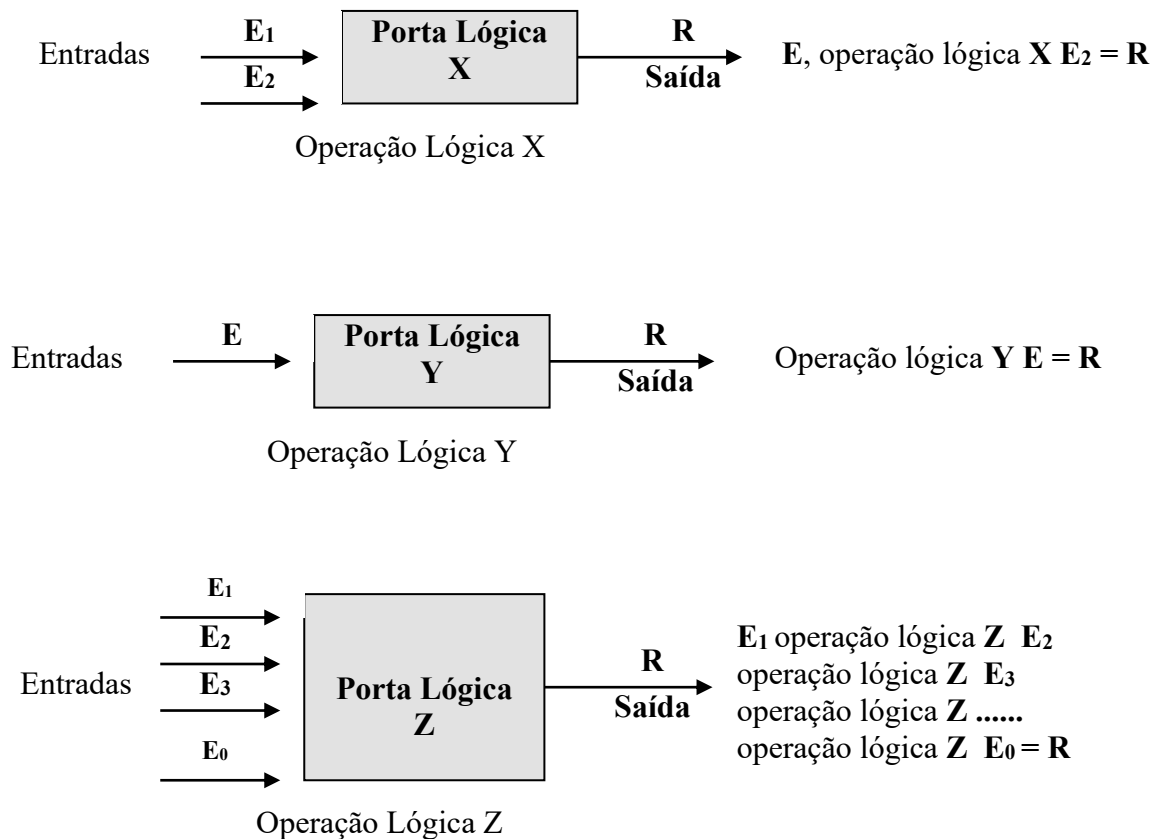


Figura 2 – Exemplo de configuração de operações lógicas.

$E, E_1, E_2, E_3 \dots E_n$	→	valores lógicos, 0 ou 1
Operações Lógicas X, Y, Z	→	operações lógicas ou portas lógicas
R	→	resultado de uma operação lógica. 0 ou 1

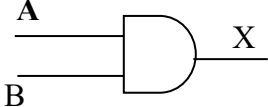
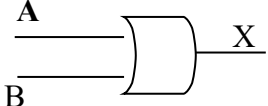
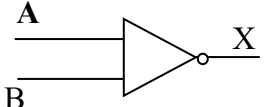
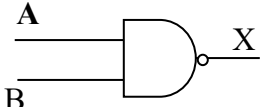
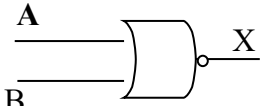
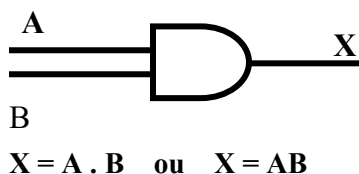
Porta Lógica	Símbolo Matemático	Símbolo Gráfico
AND	$\cdot \quad X = A \cdot B$	
OR	$+ \quad X = A + B$	
NOT	$- \quad X = \bar{A}$	
NAND	$X = \overline{A \cdot B}$	
NOR	$X = \overline{A + B}$	

Figura 3 – Símbolos gráficos e matemáticos de portas lógicas

4.2.1 Operação Lógica ou Porta AND (E)

A porta AND é definida como sendo o elemento (ou operação lógica) que produz um resultado verdade (=1) na saída, se e somente se todas as entradas forem verdade. Essa definição pode ser expressa pela tabela verdade e símbolos mostrados na figura 4.

Uma porta lógica AND pode ter várias utilidades na fabricação de um sistema digital. Entre elas, uma das mais importantes pode ser a de ativação de uma linha de dados para movimentar bits de um registrador (ou células) para outro.



Entrada	Saída
$A \cdot B$	$X = AB$
$0 \cdot 1$	0
$1 \cdot 0$	0
$1 \cdot 0$	0
$1 \cdot 1$	1

Figura 4 – Porta lógica E

4.2.2 Operação Lógica ou Porta OR (OU)

A porta OR é definida para produzir um resultado verdade (=1) na sua saída, se pelo menos uma das entradas for verdade. Esta definição pode ser expressa pela tabela verdade e símbolos mostrados na figura 5.

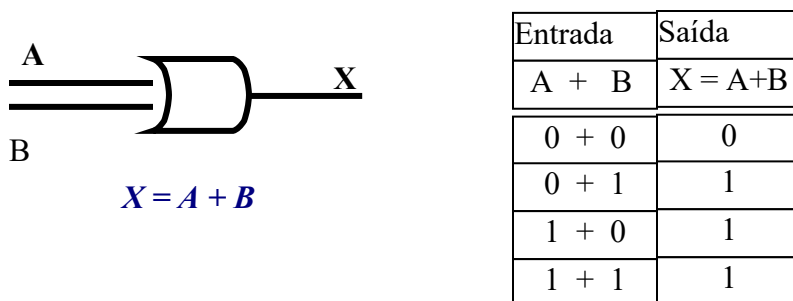


Figura 5 – Porta lógica **OR** (OU)

4.2.3 Operação Lógica NOT (Inversor)

A operação lógica NOT é também chamada de inversor ou função complemento. Ela inverte o valor de um sinal binário colocado em sua entrada, produzindo na saída o valor oposto. É um circuito lógico que requer apenas um valor na entrada (um outro circuito lógico - "buffer" – também requer apenas um valor de entradas, a figura 6 mostra um circuito inversor, bem como os símbolos utilizados e a respectiva tabela verdade.

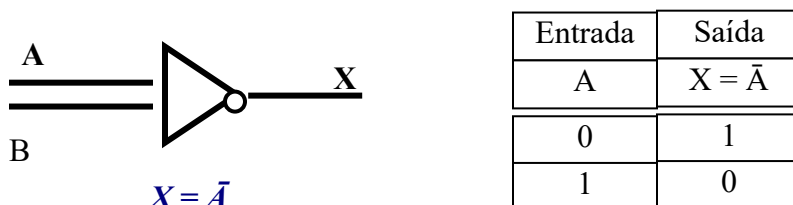


Figura 6 – Porta lógica **NOT** (Inversor ou Inverter)

É interessante observar que a conexão de dois circuitos inversores em série produz, na saída, um resultado de valor igual ao da entrada. Ou seja, um duplo complemento restaura o valor original.

4.2.4 Operação Lógica NAND – NOT AND

A operação lógica ou porta NAND é definida como o complemento da porta AND, isto é, a saída de um circuito lógico NAND (que é o mesmo resultado da

operação lógica NAND) é obtida ao se aplicar a regra da operação lógica AND e inverter o resultado. São, então, dois passos, conforme pode ser visto pelos símbolos mostrados na figura 7.

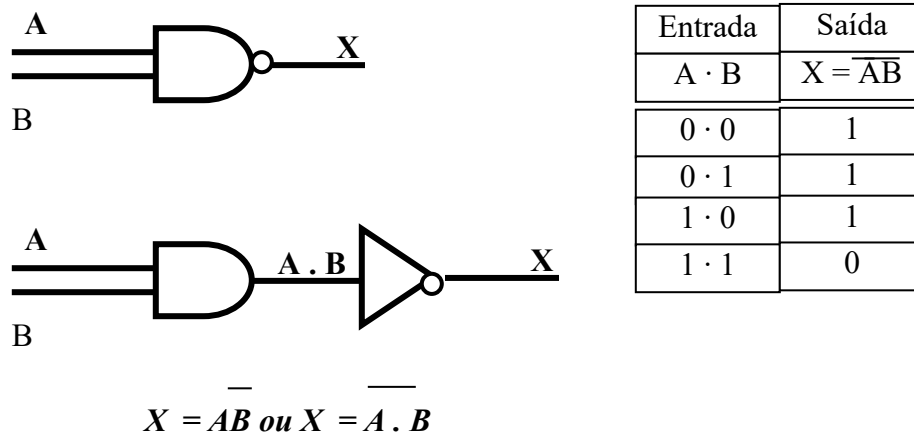


Figura 7 – Porta lógica NAND – NOT AND

A porta NAND produzirá uma saída falsa se e somente se todas as entradas forem verdade. Do contrário, a saída será verdade se, pelo menos, uma entrada for falsa.

4.2.5 Operação Lógica NOR – NOT OR

Assim como a porta NAND, a porta NOR é o complemento ou o inverso da porta OR. A saída de um circuito lógico NOR é obtida ao se efetuar a operação lógica OR sobre as entradas e inverter o resultado. Também esta operação lógica (NOR) é realizada em dois passos: primeiro a operação OR e, em seguida, a operação NOT com o resultado.

A figura 8 mostra os símbolos e a tabela verdade da operação lógica NOR.

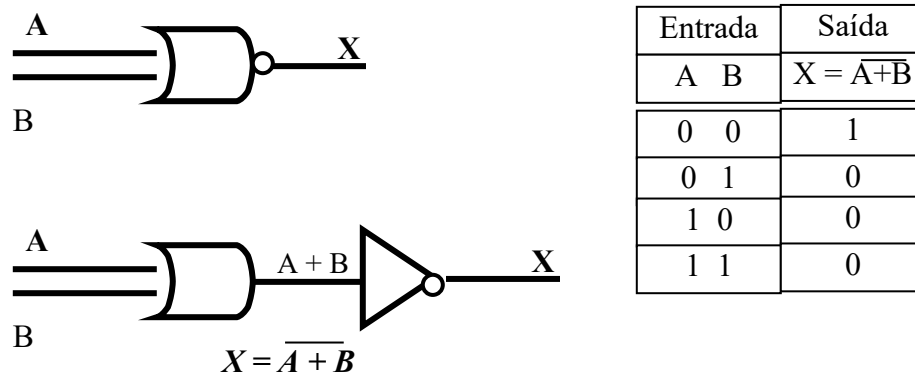


Figura 8 – Porta lógica NOR ou NOT OR

A porta NOR apresentará uma saída verdade se e somente se todas as entradas forem falsas. Caso contrário, a saída será falsa (se, pelo menos, uma das entradas for verdade).

4.2.6 Operação Lógica XOR – EXCLUSIVE OR

A operação lógica **XOR**, abreviação do termo **EXCLUSIVE OR**, pode ser considerada um caso particular da função **OR**, ou seja, sua definição: “a saída será verdade se exclusivamente uma ou outra entrada for verdade”. Isto só se aplica, é claro, se houver apenas 2 entradas e o termo exclusivamente é crucial. Não podem ambas as entradas ser verdade e é esta a diferença para os resultados da porta **OR**.

A figura 9 mostra os símbolos e a tabela verdade para a operação lógica **XOR**, e realmente podemos verificar que os resultados mostrados na tabela são iguais aos da operação **OR**, exceto quanto ao resultado de $1 \oplus 1$. Isto por causa do termo exclusivo. No caso da operação lógica **OR**, não há exclusividade de que uma ou outra entrada sejam verdadeiras, mas no que se refere à operação **XOR**, sim.

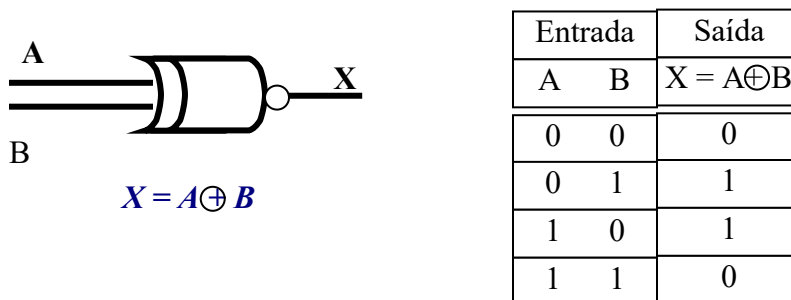


Figura 9 – Porta lógica XOR (Exclusive OR = OU exclusivo)

4.3 Minimização de funções booleanas

4.3.1 Implementação de funções booleanas

Qualquer função *booleana* pode ser implementada por um circuito eletrônico, na forma de uma rede de portas lógicas. Para qualquer função, existem diversas implementações alternativas. Considere a função *booleana* representada pela tabela verdade da Tabela 1. Podemos expressar essa função simplesmente relacionando as combinações de valores das variáveis A, B e C que tornam F igual a 1:

$$F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C}$$

Existem três possíveis combinações dos valores de entrada que fazem F igual a 1; se qualquer dessas combinações ocorrer, o resultado na saída será 1. Essa forma de expressar é denominada, por razões evidentes, uma soma de produtos). A figura 10 mostra implementação direta da função F, usando portas **AND**, **OR** e **NOT**.

Tabela 1 Função *booleana* com três variáveis

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

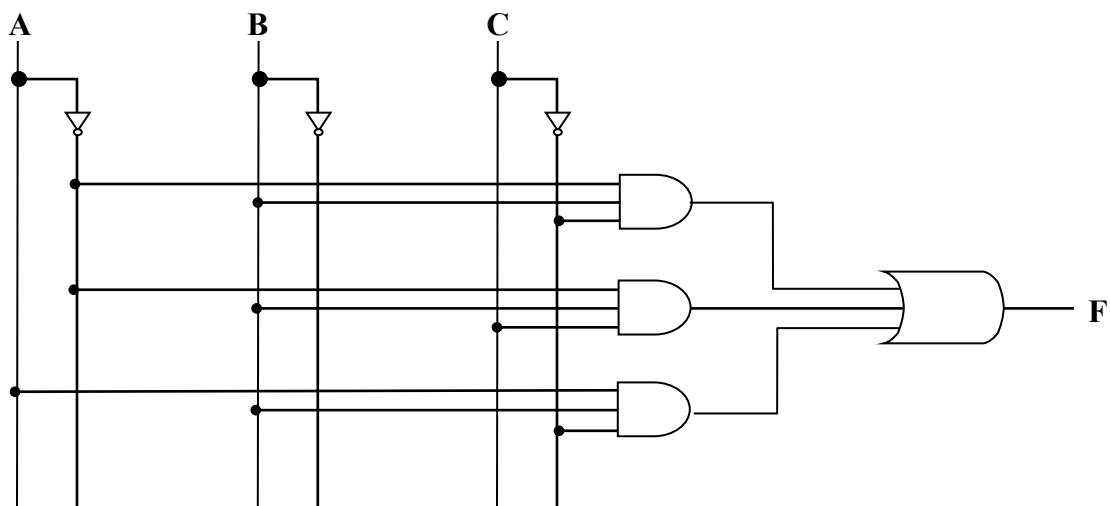


Figura 10 – Implementação em soma de produtos da Tabela A3

Consideremos a Tabela abaixo:

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

A função será verdade quando

→

→

→

→

→

$\overline{\overline{A}}\overline{\overline{B}}\overline{\overline{C}}$
 $\overline{\overline{A}}\overline{\overline{B}}\overline{\overline{C}}$

$\overline{\overline{A}}\overline{\overline{B}}\overline{\overline{C}}$
 $\overline{\overline{A}}\overline{\overline{B}}\overline{\overline{C}}$

$\overline{\overline{A}}\overline{\overline{B}}\overline{\overline{C}}$

Então:

$$X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

Simplificando algebricamente a expressão

$$\begin{aligned} X &= \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + ABC = \\ &= \bar{B}\bar{C}(A+\bar{A}) + AC(B+\bar{B}) + \bar{A}BC = \\ &= \bar{B}\bar{C} + AC + \bar{A}BC = \bar{B}\bar{C} + C(A + \bar{A}\bar{B}) = \\ &= \bar{B}\bar{C} + C(A+\bar{B}) = \bar{B}\bar{C} + AC + \bar{B}C = \\ &= AC + \bar{B}(\bar{C} + C) = AC + \bar{B} \end{aligned}$$

Uma simplificação como esta pode levar a resultados diferentes e que não levam a um circuito de configuração mínima, dependendo da escolha dos termos a serem agrupados. Simplifica-se a expressão para que a construção do circuito seja a mais simples possível.

4.4 Mapas de *Karnaugh*

O mapa de *Karnaugh* é uma forma de representação conveniente de uma função *booleana* com pequeno número de variáveis (até 4 ou 6), visando à simplificação da expressão que define essa função. O mapa consiste em uma matriz de posições, representando as possíveis combinações de valores de n variáveis binárias. A figura 11a mostra um mapa de quatro posições para uma função de duas variáveis. É conveniente, como veremos mais adiante, listar essas combinações na ordem **00**, **01**, **11**, **10**. Como as posições correspondentes às combinações são usadas para registrar informações, as combinações usualmente são escritas acima de cada posição. No caso de três variáveis, a representação é um arranjo de oito posições (figura 11b), como os valores de uma das variáveis à esquerda e os valores das duas outras variáveis acima de cada posição. Para quatro variáveis, são requeridas 16 posições, arranjadas tal como indicado na figura 11c.

O mapa de *Karnaugh* pode ser usado para representar qualquer função *booleana*, da seguinte forma. Cada posição corresponde a um único produto da expressão na forma de soma de produtos, onde o valor **1** corresponde à variável e o valor **0** corresponde à negação (**NOT**) dessa variável. Portanto, o produto AB

corresponde à quarta posição na figura 11a. Para cada um dos produtos da expressão que define a função, é colocado o valor 1 na posição correspondente.

Portanto, no exemplo com duas variáveis, o mapa corresponde à função definida pela expressão $AB + \bar{A}\bar{B}$. Dada a tabela verdade de uma função booleana, é fácil construir o mapa de *Karnaugh* correspondente: para cada combinação de valores das variáveis que produz resultado 1 na tabela verdade, preencha a posição correspondente com valor 1. A figura 11b mostra o mapa correspondente à função F especificada. Para converter uma expressão *booleana* para um mapa, é necessário primeiro escrever essa expressão no que se chama *forma canônica*: cada termo da expressão deve conter cada variável.

Os rótulos mostrados na figura 11d enfatizam o relacionamento entre as variáveis e as linhas e colunas do mapa. Nesse caso, as duas linhas indicadas pelo rótulo A são aquelas em que a variável A tem valor 1; as linhas que não têm o rótulo A são aquelas em que a variável A tem valor 0 (analogamente para B , C e D).

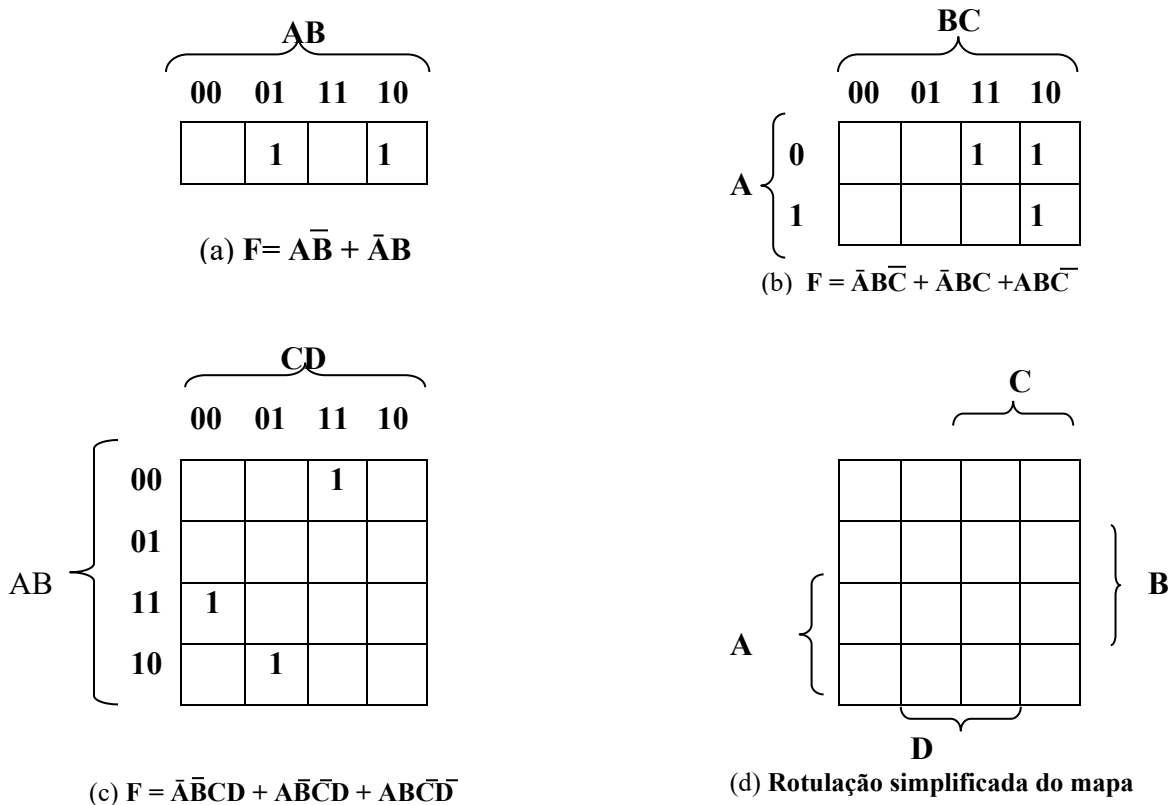


Figura 11 – Uso de mapas de *Karnaugh* para representar funções *booleanas*

Uma vez criado o mapa da função, podemos quase sempre escrever uma expressão algébrica equivalente mais simples, verificando o arranjo de 1s do mapa. O princípio é o seguinte: quaisquer duas posições adjacentes diferem em apenas uma das variáveis; se duas posições adjacentes têm ambas valor igual a 1, então os termos correspondentes do produto diferem em apenas uma variável.

Nesse caso, os dois termos podem ser combinados, eliminando essa variável. Por exemplo, na figura 12a, as duas posições adjacentes correspondem aos termos $\bar{A} B \bar{C} D$ e $\bar{A} B C D$. Portanto, a função pode ser expressa como:

$$\bar{A} B \bar{C} D + \bar{A} B C D = \bar{A} B D$$

Esse processo pode ser estendido de várias formas. Primeiro, o conceito de adjacência pode ser estendido para incluir um giro em torno das extremidades do mapa. Assim, a posição no topo de uma coluna é adjacente à posição mais embaixo da mesma coluna; a posição mais à esquerda de uma linha é adjacente à posição mais à direita da mesma linha. Essas situações são mostradas na figura 12b e 12c. Segundo, podemos agrupar não apenas duas, mas posições adjacentes; isto é, 4, 8 etc. Os três exemplos seguintes da figura 12 mostram agrupamentos de 4 posições. Note que, nesse caso, duas das variáveis podem ser eliminadas.

Os três últimos exemplos mostram agrupamentos de oito posições, que possibilitam eliminar três variáveis.

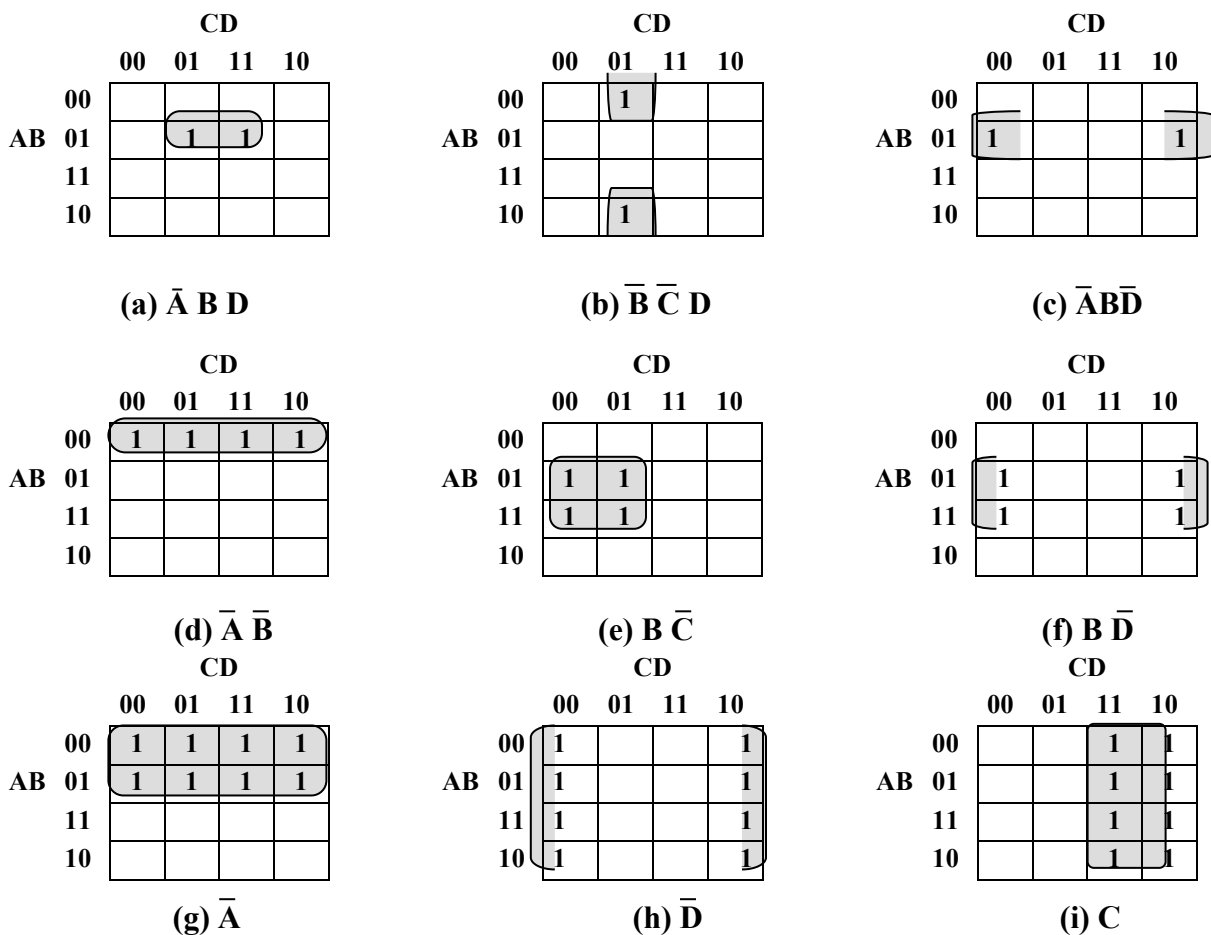


Figura 12 - Uso de mapas de *Karnaugh*

Podemos resumir as regras de simplificação do seguinte modo:

1. Dentre as posições marcadas (posições com valor 1), encontre aquelas que pertencem a um único bloco máximo de 1, 2, 4 ou 8 posições e contorne esses blocos.
2. Selecione blocos adicionais de posições marcadas, que sejam tão grandes quanto possível e em menor número possível, mas que incluam toda posição marcada, pelo menos uma vez. Pode não haver um resultado único em alguns casos. Por exemplo, se uma posição marcada combina exatamente com duas posições, e não existe uma quarta posição marcada para completar um bloco maior, então existem duas possibilidades de escolha de agrupamento. Quando se estiver marcando blocos, é permitido usar a mesma posição de valor 1 mais de uma vez.
3. continue marcando o contorno de posições marcadas simples, ou pares de posições marcadas adjacentes, ou grupos de quatro, oito etc. posições marcadas adjacentes, de modo que cada posição marcada pertença a pelo menos um bloco; então use o menor número possível desses blocos que inclua todas as posições marcadas.

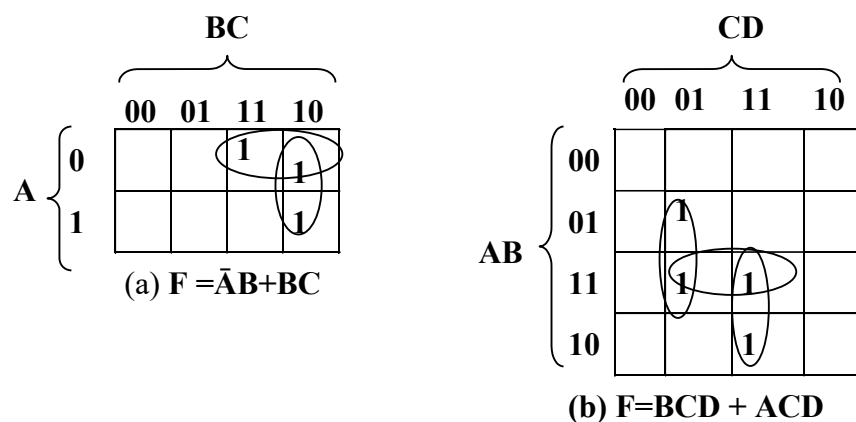


Figura 13 – Grupos sobrepostos

A figura 13a, baseada no valor de $F = \bar{A} B \bar{C} + \bar{A} B C + A B \bar{C}$ ilustra esse procedimento. Se restar alguma posição marcada isolada, depois do agrupamento em blocos, então cada uma delas será marcada como um bloco. Finalmente, antes de converter o mapa em uma expressão *booleana* simplificada, qualquer bloco de 1s que seja completamente sobreposto por outros blocos é eliminado. Isso é mostrado na figura 13b. nesse caso, o grupo horizontal é redundante, e pode ser ignorado na construção da expressão *booleana*.

Uma característica adicional dos mapas de *Karnaugh* deve ser mencionada. Em alguns casos, certas combinações de valores de variáveis nunca ocorrem e, portanto, a saída correspondente nunca ocorre. Esses casos são denominados “negligenciáveis” (*don't care*). Para cada um deles, é usada a letra “d” na posição correspondente no mapa. Ao fazer o agrupamento de posições marcadas e a simplificação, cada “d” pode ser tratado como um valor 1 ou 0, escolhendo-se o valor que resulta na expressão mais simples.

Mapas de Karnaugh acima de 4 variáveis

Como visto os mapas de Karnaugh possuem estruturas e aplicações bastante simples. Porém para número de variáveis acima de quatro, faz-se necessário a construção de múltiplos mapas de quatro variáveis (MK4) e o uso (ou abstração) de ilhas (ou vizinhanças) também entre MK4. A seguir, são apresentados exemplos de M.K. de cinco variáveis.

Mapas de Karnaugh para 5 variáveis

O mapa de Karnaugh para cinco variáveis ($S = f(A, B, C, D, E)$) é formado por trinta e duas células ($2^5 = 32$) dispostas como mostra a figura a seguir.

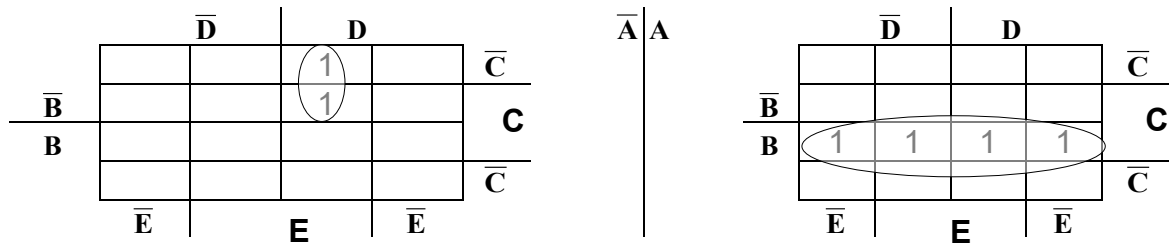
				\bar{D}	D							
				m_0	m_1	m_3	m_2	\bar{C}				
\bar{B}				m_4	m_5	m_7	m_6	C				
B				m_{12}	m_{13}	m_{15}	m_{14}					
				m_8	m_9	m_{11}	m_{10}	\bar{C}				
				\bar{E}			\bar{E}					
								E				

$\bar{A} \mid A$

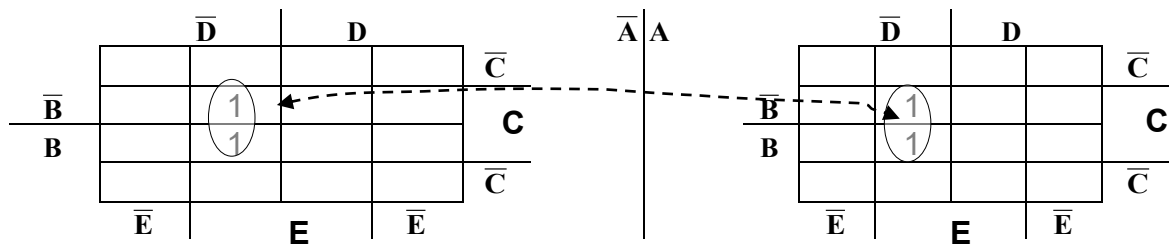
				\bar{D}	D							
				m_{16}	m_{17}	m_{19}	m_{18}	\bar{C}				
\bar{B}				m_{20}	m_{21}	m_{23}	m_{22}	C				
B				m_{28}	m_{29}	m_{31}	m_{30}					
				m_{24}	m_{25}	m_{27}	m_{26}	\bar{C}				
				\bar{E}			\bar{E}					
								E				

A	B	C	D	E	Minitermos
0	0	0	0	0	$m_0 = \bar{A} \bar{B} \bar{C} \bar{D} \bar{E}$
0	0	0	0	1	$m_1 = \bar{A} \bar{B} \bar{C} \bar{D} E$
0	0	0	1	0	$m_2 = \bar{A} \bar{B} \bar{C} D \bar{E}$
0	0	0	1	1	$m_3 = \bar{A} \bar{B} \bar{C} D E$
0	0	1	0	0	$m_4 = \bar{A} \bar{B} C \bar{D} \bar{E}$
0	0	1	0	1	$m_5 = \bar{A} \bar{B} C \bar{D} E$
0	0	1	1	0	$m_6 = \bar{A} \bar{B} C D \bar{E}$
0	0	1	1	1	$m_7 = \bar{A} \bar{B} C D E$
0	1	0	0	0	$m_8 = \bar{A} B \bar{C} \bar{D} \bar{E}$
0	1	0	0	1	$m_9 = \bar{A} B \bar{C} \bar{D} E$
0	1	0	1	0	$m_{10} = \bar{A} B \bar{C} D \bar{E}$
0	1	0	1	1	$m_{11} = \bar{A} B \bar{C} D E$
0	1	1	0	0	$m_{12} = \bar{A} B C \bar{D} \bar{E}$
0	1	1	0	1	$m_{13} = \bar{A} B C \bar{D} E$
0	1	1	1	0	$m_{14} = \bar{A} B C D \bar{E}$
0	1	1	1	1	$m_{15} = \bar{A} B C D E$
1	0	0	0	0	$m_{16} = A \bar{B} \bar{C} \bar{D} \bar{E}$
1	0	0	0	1	$m_{17} = A \bar{B} \bar{C} \bar{D} E$
1	0	0	1	0	$m_{18} = A \bar{B} \bar{C} D \bar{E}$
1	0	0	1	1	$m_{19} = A \bar{B} \bar{C} D E$
1	0	1	0	0	$m_{20} = A \bar{B} C \bar{D} \bar{E}$
1	0	1	0	1	$m_{21} = A \bar{B} C \bar{D} E$
1	0	1	1	0	$m_{22} = A \bar{B} C D \bar{E}$
1	0	1	1	1	$m_{23} = A \bar{B} C D E$
1	1	0	0	0	$m_{24} = A B \bar{C} \bar{D} \bar{E}$
1	1	0	0	1	$m_{25} = A B \bar{C} \bar{D} E$
1	1	0	1	0	$m_{26} = A B \bar{C} D \bar{E}$
1	1	0	1	1	$m_{27} = A B \bar{C} D E$
1	1	1	0	0	$m_{28} = A B C \bar{D} \bar{E}$
1	1	1	0	1	$m_{29} = A B C \bar{D} E$
1	1	1	1	0	$m_{30} = A B C D \bar{E}$
1	1	1	1	1	$m_{31} = A B C D E$

Para entendermos melhor o significado desse conceito, observe os exemplos apresentados a seguir para cinco variáveis:



$$S = \bar{A} \cdot \bar{B} \cdot D \cdot E + A \cdot B \cdot C$$



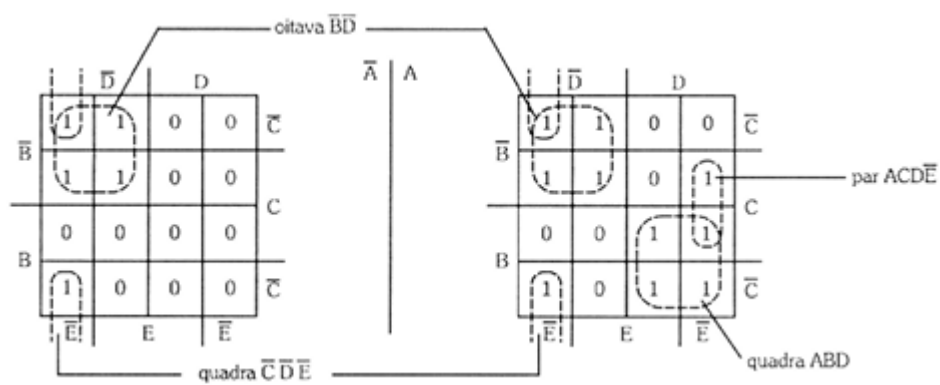
$$S = C \cdot \bar{D} \cdot E$$

Dada a tabela verdade encontrar a expressão simplificada.

A	B	C	D	E	S
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	1
1	1	1	1	1	1

Colocando os casos no diagrama, temos:

	\bar{D}		D			\bar{A} A		\bar{D}		D		
\bar{B}	1	1	0	0	\bar{C}			1	1	0	0	\bar{C}
	1	1	0	0				1	1	0	1	C
B	0	0	0	0	\bar{C}			0	0	1	1	C
	1	0	0	0	\bar{C}			1	0	1	1	\bar{C}
	\bar{E}	E	\bar{E}					\bar{E}	E	\bar{E}		



A expressão simplificada será: $S = \bar{B}\bar{D} + \bar{C}\bar{D}\bar{E} + ABD + ACD\bar{E}$.

Uma outra forma de resolução é apresentada na figura abaixo. Utiliza-se da última variável (E). O mapa final pode ser visualizado como sendo dois mapas de quatro variáveis sobrepostos. Um dos mapas, referente a E=0, corresponde à parte inferior da linha diagonal de divisão das células do mapa final. O outro mapa, referente a E=1, corresponde à parte superior da linha diagonal de divisão das células do mapa final. Cada mapa apresenta a sua leitura individual. Se a leitura em um dos mapas for igual (sobreposta) à leitura do outro mapa, estas duas leituras formam uma única leitura.

		CD			
		00	01	11	10
E \ AB	00				
	01				
	11				
	10				

		CD			
		00	01	11	10
E \ AB	00	1 0	1 0	1 0	1 0
	01	0 1	0 1	0 0	0 0
	11	0 1	0 1	1 1	0 0
	10	1 0	0 1	0 0	0 0

$$S = \overline{A}\overline{B}E + \overline{B}\overline{C}\overline{E} + ABCD + A\overline{C}D\overline{E} + \overline{B}\overline{C}DE$$

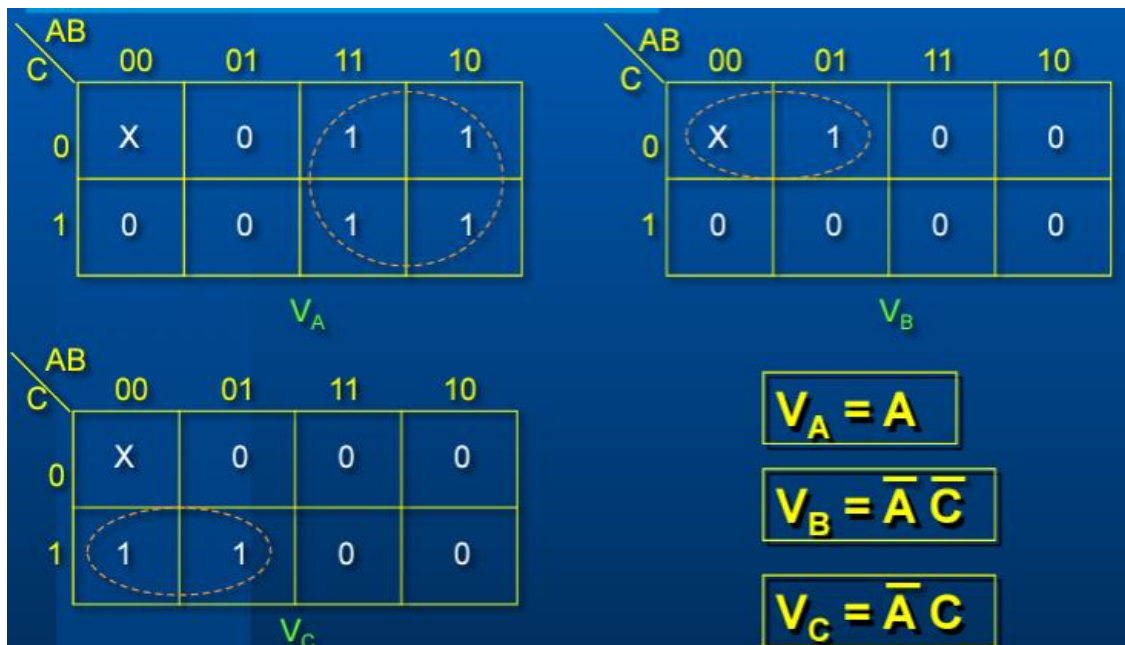
Condição Irrelevante

- Condições de entrada para as quais não existem níveis de saída especificados;
- Condições de entrada que nunca ocorrerão.

A	B	C	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	x
1	0	0	x
1	0	1	1
1	1	0	1
1	1	1	1

x } "don't
x } care"

Nesses casos: Utiliza-se "x" como "0" ou "1" convenientemente, de modo à tornar a expressão mais simples. A figura a seguir apresenta alguns exemplos.



Quadro Resumo: Diagramas (Mapas) de Karnaugh

2 variáveis

	\bar{B}	B
\bar{A}		
A		

3 variáveis

	\bar{B}	B
\bar{A}		
A		
	\bar{C}	C

4 variáveis

	\bar{C}	C	
\bar{A}			\bar{B}
A			B
	\bar{D}	D	\bar{B}

5 variáveis

	\bar{D}	D		\bar{A}	A		\bar{D}	D	
\bar{B}						\bar{B}			
B						B			
	\bar{E}	E	\bar{E}				\bar{E}	E	\bar{E}

Para mais de cinco variáveis, o processo de minimização utilizando Mapas de Karnaugh fica difícil de ser executado, pois, a montagem do mapa é trabalhosa e a visualização das adjacências é mais complicada. Para estas situações, utilizam-se outros métodos ou, então, outros dispositivos eletrônicos que são capazes de implementar circuitos lógicos sem a necessidade de minimização da expressão booleana correspondente.