

# EA - 772 CIRCUITOS LÓGICOS



**Prof. Dr. José W. M. Bassani**  
**Revisão - 2015**  
**DEB/FEEC/UNICAMP**

## Sumário

1. Introdução, motivação, objetivos ... ou pra que diabos serve o curso de circuitos lógicos .....	3
2. Evolução histórica das máquinas de calcular.....	5
3. Números.....	11
4. Aritmética Binária.....	29
5. Células Binárias.....	41
6. Álgebra Booleana.....	54
7. Blocos Lógicos.....	64
8. Tecnologias de circuitos integrados.....	68
9. Funções booleanas e circuitos digitais.....	73
10. Simplificação de funções booleanas.....	83
11. Circuitos Sequenciais.....	90
12. Códigos.....	97
13. Exemplos de Circuitos Digitais .....	100
14. Bibliografia.....	104

NOTA: Este texto não se trata de publicação original. Alguns pontos são apenas tradução de textos citados. São notas de aula e não uma apostila ou outra coisa do gênero. Os estudantes devem usar este material como referencia inicial para estudo. O autor não autoriza reprodução ou qualquer outra utilização que não seja como roteiro e material de estudo individual. O seu uso indevido é responsabilidade de quem o fizer.

## 1. Introdução, motivação, objetivos ... ou pra que diabos serve a disciplina de circuitos lógicos?

O curso tem como objetivo ensinar como projetar circuitos básicos que são utilizados no dia-a-dia do projeto e construção dos computadores e de outras máquinas. Outra meta é o ensino de técnicas que serão úteis no entendimento e no projeto de sistemas de transmissão de dados. De modo mais geral vamos estudar e projetar circuitos digitais, tais como os circuitos de transferência de dados, contadores, divisores, vários tipos de detetores, somadores, temporizadores e outros.

O desenvolvimento de circuitos lógicos é fundamentado por uma álgebra própria, a Álgebra Booleana. Com base nesta álgebra pode-se desenvolver expressões que correspondem a circuitos complexos. Uma especificação organizada (e.g. na forma de uma tabela), do processo que se deseja sintetizar é suficiente para possibilitar a construção de circuitos com base nas expressões algébricas. Utilizando regras e propriedades da operação algébrica pode-se realizar, formalmente, simplificações nas expressões que possibilitam a construção de circuitos incrivelmente mais simples do que os circuitos originais. Isto, por si só, gera a oportunidade de se ver a importância do desenvolvimento das novas teorias e métodos matemáticos para expandir o conhecimento nas outras áreas e quão importante é o aprendizado das regras e fundamentos básicos das teorias para o desenvolvimento futuro da Engenharia Elétrica.

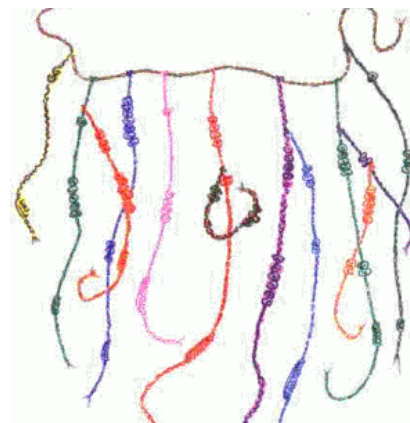
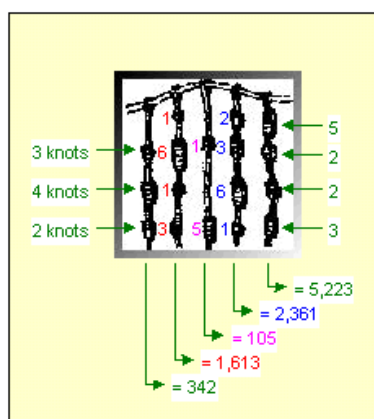
O estudante deverá tomar contato com técnicas envolvidas na síntese de circuitos lógicos e ter bom conhecimento de como manipular as expressões algébricas para obter os melhores circuitos (menor custo e melhor desempenho). Por outro lado, espera-se que o aluno se sinta estimulado a pensar sobre novos problemas no campo. Por exemplo, nos novos métodos de simplificação automática de circuitos, nos computadores moleculares e no fato de que ser engenheiro significa, contribuir,

com criatividade para que a ciência possa seguir seu caminho da auto-correção, buscando aproximar os modelos da verdade.

Alternativamente você pode se lembrar que é hora de cursar uma disciplina básica, obrigatória, importante para o entendimento de muitas outras (e.g. comunicações, processamento digital de sinais), e que é preciso estudar muito para ser um bom engenheiro.

## 2. Evolução histórica das máquinas de calcular

Há uma grande necessidade de se desenvolver métodos e máquinas para computar. Os engenheiros são, em princípio, os responsáveis pela construção destas máquinas. Portanto, começo com um apelo aos jovens estudantes, não tirem dos seres menos preparados, o prazer de poder usá-las de modo cada vez mais simples. Mas, de onde vem este interesse pela computação? É possível que isto esteja ligado a alguma necessidade ancestral que temos de contar. Já em 1871, Charles Darwin em sua obra “The decent of man” comenta que algumas espécies animais, possuem memória e imaginação (Boyer, 1991). Evolutivamente, saber se há um grupo de 2 ou de 4 predadores em ação pode ter sido importante para a sobrevivência. É muito estimulante pensar sobre, mas difícil de saber, quais os elementos mínimos necessários para que cérebros mais primitivos efetuem a atividade de contar. Talvez a capacidade de identificar padrões seja suficiente (dentro de certos limites), ou talvez seja preciso padrões mais elementares (e.g. um único símbolo) e memória. Um sistema bastante sofisticado de armazenamento e representação de quantidades (Figura 0) e contagem já podia ser encontrado na civilização Inca (National Geographic, Dec. 1973, 729-766). Identificar símbolos e contá-los é computar!



© 1999 Simon Fraser University  
HTML Written by Alice Storey

Figura 0. O Quipu. Sistema Inca de contagem e armazenamento de informação. À direita: Ref.: Como fazer seu próprio quipu; à esquerda: quipu Inca com ilustração de codificação para contagem (veja: [http://agutie.homestead.com/files/Quipu\\_B.htm](http://agutie.homestead.com/files/Quipu_B.htm))

O primeiro dispositivo de cálculo, contudo, parece ter sido o ábaco (do latim *Abacus*), introduzido pelos babilônios cerca de 3000 AC. A Figura 1 ilustra um ábaco que consiste de uma armação retangular com hastes e contas, dividida por uma barra transversal, separando conjuntos de 5 ou de 2 contas. Cada coluna de contas tem um peso, da direita para a esquerda, 1, 10, 100, etc. Na representação de um número até 5 são utilizadas as contas do grupo de 5 e quando este valor é ultrapassado, até dez, usa-se a combinação do movimento (na direção da divisão central) das contas agrupadas em duplas. O sistema permite que as operações aritméticas sejam efetuadas e que o resultado fique registrado. Os cálculos podem ser efetuados de modo bastante rápido por indivíduos experientes.

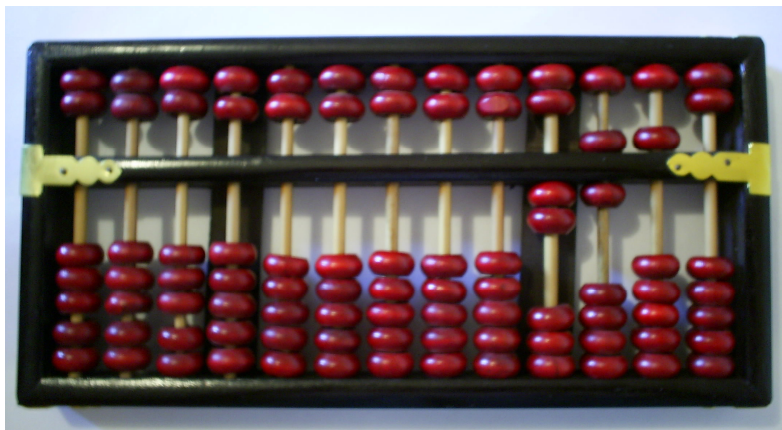


Figura 1. Foto de um ábaco, mostrando o número 2650. Versão chinesa do ábaco (origem: abacus do Latin que quer dizer *sandboard*. A palavra pode ter se originado do Grego *abakos* que quer dizer *calculating table* (*abax*).

Milênios se passaram até que o homem, que já conhecia o ábaco, criasse a primeira máquina de calcular. A era das máquinas mecânicas durou aproximadamente de 1623 a 1945, um período bastante longo. Já no século XVII os matemáticos usavam algumas máquinas capazes de efetuar adição, subtração, multiplicação e divisão. Dentre os nomes importantes estão Wilhelm Schickhard, Blaise Pascal e Gottfried Leibnitz. A primeira máquina programável foi projetada por Charles Babbage em 1823, nunca tendo sido realmente construída. A primeira máquina de calcular que chegou a

ser exibida publicamente foi a máquina construída por George e Edvard Scheutz (Pai e filho). Uma das primeiras utilizações de máquinas mecânicas para computação foi o equipamento que fazia uso de cartões perfurados projetado por Herman Hollerith em 1890. Mais tarde a sua companhia se juntou com uma empresa competidora para fundar nada menos que a IBM (International Business Machine).

Aproximadamente no período de 1937 a 1953 nasceram as máquinas eletromecânicas e eletrônicas, constituindo a primeira geração de computadores eletrônicos. As chaves eletrônicas, embora não muito confiáveis no início, podiam "abrir" e "fechar" cerca de 1.000 vezes mais rapidamente que os acionadores ou chaves mecânicas.

Uma contribuição importante veio de Alan Turing que projetou a chamada Colossus usada para quebrar códigos das forças alemãs, durante a II Guerra Mundial. A maior contribuição de Turing, no entanto, foi a sua chamada Máquina de Turing seguindo um formalismo matemático usado no estudo das chamadas funções computáveis.

O primeiro computador eletrônico de aplicação geral foi o ENIAC, projetado na universidade da Pensilvânia por Eckert e Mauchly entre 1943 e 1945, usado intensamente nos cálculos no projeto da bomba de hidrogênio. A partir desta máquina estes autores aliados a John von Neumann desenvolveram o primeiro computador eletrônico de programa armazenado. Esta máquina acabou por dar origem ao UNIVAC em 1952.

A segunda geração de computadores inaugura a fase dos circuitos de chaveamento baseados em transistores e diodos, operando na faixa de frações de microsegundos. Por volta de 1954 as máquinas TRADIC da Bell Lab e a do TX-0 Lincoln Lab no MIT são exemplos. Mudanças importantes ocorreram do ponto de vista da arquitetura e do processamento, em particular com a introdução de cálculos em ponto flutuante permitindo a operação com números reais. Esta era, que durou até por volta de 1962, trouxe a inovação da construção de muitas linguagens de programação tais como o FORTRAN (1956) para computação científica, o ALGOL (1959) como uma linguagem estruturada e o COBOL (1959) voltada ao processamento comercial. Nesta época surge uma família de computadores da IBM que em sua versão mais moderna

trouxe a novidade dos processadores de entrada e saída que otimizaram a troca de informações entre os dispositivos e a memória, que já nesta época podia ser de acesso aleatório. O conceito de supercomputadores já pôde ser introduzido e implementado. São máquinas que devem possuir capacidade de computação cerca de uma ordem de magnitude acima das máquinas disponíveis na época.

Na terceira geração (1963-1972) surgem máquinas com grande ganho de "potência computacional". Conta-se com a inovação dos chamados circuitos integrados (CI). As memórias de semicondutores passam a ser usadas no lugar das memórias de núcleo magnético da geração anterior. O grau de integração, nesta fase, era ainda pequeno com cerca de 10 dispositivos por CI evoluindo em seguida para cerca de 100 dispositivos por pastilha (*chip*). Muitas novas máquinas surgiram com vantagens de arquitetura e novas linguagens também foram desenvolvidas, como por exemplo, o CPL (*Combined Programming Language*) para ficar com apenas as características mais importantes da ALGOL. É nesta geração que começam a surgir os chamados computadores paralelos (e.g. SOLOMON da Westinghouse Co. e o ILLIAC IV da Burroughs em cooperação com o departamento de defesa e a universidade de Illinois).

Na quarta geração começa a chamada integração em larga escala (Large Scale Integration, LSI), com 1.000 dispositivos por CI e já a escala muito grande de integração (Very Large Scale Integration, VLSI) com 100.000 dispositivos por CI. Nesta última já se podia incluir processadores inteiros em um único CI e para alguns sistemas o computador inteiro (processador, memória principal e controlador de entrada e saída). Surgem computadores com memória muito grande como por exemplo o CRAY 2. Muitas variantes do processamento paralelo começam a ser implementadas. Começam a surgir os microcomputadores e estações de trabalho e o uso de grandes computadores operando em time-sharing.

Nesta geração surge a linguagem C e o sistema operacional UNIX que teve uma de suas versões escrita em C para o recém-desenvolvido DEC PDP-11 (Digital Equipment Co., USA). Nos anos 80 os Estados Unidos tomaram conhecimento (provavelmente por meio do chamado Lax report, relatório editado por Peter Lax; <http://www.pnl.gov/scales/archives.stm>) de que o país estava investindo muito pouco e dando poucas oportunidades para desenvolvimento de supercomputadores, por exemplo em



contraste com o Japão. Isto deu origem a inúmeros centros para estudo de computação de alto desempenho ([http://www.nsf.gov/news/news\\_summ.jsp?cntn\\_id=106875](http://www.nsf.gov/news/news_summ.jsp?cntn_id=106875)), um destes o famoso John von Neumann Center de Princeton.

Ainda nos anos 80 (1984 a 1990) se inicia a quinta geração de computadores que se caracterizou pela adoção definitiva da computação paralela. Nesta geração algumas máquinas dispunham de centenas de processadores que podiam ao mesmo tempo atuar em diferentes partes do mesmo programa. Já em 1990 era possível integrar milhões de dispositivos no mesmo CI e as memórias de semicondutores passaram a ser padrão em todos os computadores. Proliferam as redes de computadores e o uso das estações de trabalho. O sistema DEC VAX-780 era um computador de uso geral, baseado no sistema operacional UNIX com vários processadores cada qual "cuidando" do processo de usuários diferentes. Surge para competir o chamado Sequent Balance 8000 com 20 processadores, dividindo um único bloco de memória com um bloco especialmente dedicado a cada processador.

Surge em seguida o sistema apelidado de hypercubo da Intel, o iPSC-1 que usava um bloco de memória para cada processador, que estavam ligados em rede. Esta foi uma filosofia diferente para a arquitetura, a chamada arquitetura de memória distribuída. Esta versão evoluiu para máquinas com mais de uma centena de processadores e por fim um sistema com vários milhares de processadores bem simples foi colocado a disposição. Ainda havia nesta fase o domínio do processamento vetorial ao invés de processamento paralelo. As redes evoluíram bastante tanto as WANs (Wide area network) quanto as LANs (Local area network). Nestes casos com a idéia de compartilhar um grande computador central parte do processamento passou a ser feito usando recursos de vários modos: localmente, distribuido em servidores e ainda acessando recursos de supercomputadores todos ligados na rede. As estações de trabalho e servidores ficaram mais eficientes e baratos com a entrada da chamada tecnologia RISC (**R**educed **I**nstruction **S**et **C**omputer).

Inicia-se o que podemos chamar de sexta geração por volta de 1990. Cada vez mais fica difícil predizer o que significa uma nova geração, a não ser retrospectivamente. É possível pela evolução que se observa de hardware e de software que a computação paralela irá, definitivamente, predominar. No fundo, já no

início dos anos 90, havia a idéia de que talvez o sistema híbrido vetorial e paralelo devesse ser uma boa solução. Nesta época já se prometia sistemas híbridos capazes de operar na faixa dos teraflops (TFLOPS,  $10^{12}$  FLOPS – Floating point operations per second). As arquiteturas começam a se fundir: RISC, pipelining e paralela. Será que isto significa que os princípios básicos das operações computacionais estariam definitivamente determinados e agora só nos restaria combiná-los? O que vem acontecendo hoje? Busque esta informação você mesmo!

A grande mudança nesta geração parece ter sido nas interligações de computadores. As taxas de transmissão aumentaram ordens de grandeza e a abrangência das redes cresceu exponencialmente. A computação de alto desempenho e a pesquisa de sistemas muito mais eficientes que os convencionais também cresceu muito. A cada instante um novo centro de estudos avançados destas tecnologias aparece e recebe, nos países evoluídos, grandes somas de recursos para pesquisa.

E então, os investimentos a partir do famoso relatório Lax fizeram efeito, pelo menos nos Estados Unidos? Veja o site sobre os 500 supercomputadores atuais (<http://www.top500.org>) e conclua por você mesmo.

## 3. Números

### 3.1 Conceito

É aparentemente óbvia e ao mesmo tempo intrigante a necessidade que os seres humanos, e provavelmente outras espécies tem, de contar. Esta atividade está na base do desenvolvimento do que hoje chamamos de números. É provável que a persistência da espécie humana não tenha sido independente da sua capacidade de desenvolver a matemática e, nela, o conceito de números, que se desenvolveu gradual e lentamente por vários milênios (Boyer, 1991). Este conceito está na origem de pensamentos sofisticados da matemática atual como também das linguagens e dessa nossa compulsão por ordenar, classificar e "atribuir números", aparentemente para minimizar erros e prever o futuro ("adivinhar" onde e quando eventos importantes podem ocorrer). Isso nos confere maiores chances de sobrevivência e provavelmente algum prazer e/ou conforto.

Faz diferença ter que enfrentar com as mãos limpas ou sem armas de fogo, um, dois ou muitos animais famintos. Faz diferença pescar um ou muitos peixes e ter um, dois ou cinco filhos. É provável ter sido confortante saber que seu exército primitivo continha 10.000 homens a mais que o dos seus opositores. De necessidades básicas como estas deve ter surgido a necessidade de contar. A representação numérica e o reconhecimento de formas e padrões também evoluíram lentamente ao longo de milênios com experiências acumuladas por civilizações diferentes. Contar requer estratégias para acumular, agrupar, dar peso e registrar. Quando há muitos elementos é preciso, definitivamente, computar. Para lidar com os números, povos como os egípcios e os mesopotâmicos há cerca de 4.000 anos criaram símbolos e os organizaram em grupos atribuindo-lhes pesos relativos às posições ocupadas em sequências pré-estabelecidas. Estas atividades devem ter originado o conceito de sistemas de numeração (bases numéricas), provavelmente por associação com conjuntos incidentalmente disponíveis, como por exemplo, 5 dedos nas mãos e pés. Estes, deram origem a sistemas quinários (base 5), decimais (base 10), vigesimais (base 20). Outras bases foram também usadas primitivamente como a binária, ternária

e outras, em porcentagem bastante inferior, mas deixando efeitos que são vistos nos nossos dias, como por exemplo, contar de 2 em 2.

Foi necessário desenvolver a linguagem para expressar os números da forma que hoje os conhecemos, mas os números e a capacidade de contar também influenciaram as linguagens. A palavra *twelve* (12 em inglês) pode ter sido originada do uso da base 10 significando *two above* ou *two more* (dois acima ou a mais). Nós diferenciamos no português, um de muitos, ou seja, singular e plural. No grego distingue-se 1, 2, 3 e muitos, como pode ser a origem do nosso mono, bi, tri e poli. Outro exemplo é o uso da base para "falar o número" como há ainda hoje na língua francesa. Escreve-se para o número oitenta, *quatre-vingt* (quatro-vingte), evidenciando o uso ancestral da base 20.

Insisto na questão de quão básico é contar. Será que outras espécies tem esta capacidade? Em sua obra "The descent of man" (1871), Charles Darwin descreveu a capacidade de alguns "animais superiores" de memória e imaginação. Hoje, estas habilidades estão melhor estudadas e confirmadas. Quem nunca ouviu falar da inteligência dos corvos (corvo = *crow* em inglês)? Procure sobre isto quando estiver navegando pela internet (veja por exemplo: <http://www.pbs.org/lifeofbirds/brain/>). Estes animais conseguem saber quantos caçadores estão se aproximando (se 1 ou até 4, por exemplo). Se uma artimanha é feita para atraí-los, isto não dura muito. Duas vezes seguidas da mesma estratégia faz com que as aves se posicionem em árvores bem distantes e façam um enorme barulho, não mais caindo na armadilha. Há indústrias nos EUA dedicadas a desenvolver maneiras de ludibriar os corvos para que possam ser caçados. Já sei, vocês devem estar pensando: e por que alguém iria querer caçar corvos? Procure sobre isto no mesmo local indicado anteriormente e em outros e verá!

Não há dúvidas que os animais podem contar. Os mesmos corvos conseguem enterrar milhares de sementes e depois recuperar cerca de 90% delas. Algumas páginas poderiam ser escritas aqui sobre a capacidade de animais distinguirem padrões, contar e executar outras atividades como por exemplo usar ferramentas, mas vamos voltar a nossa disciplina. Nós estávamos falando sobre contar, reconhecendo padrões, agrupando, colocando em sequencia, dando pesos, descrevendo por meio de uma linguagem, computando ... números.

### 3.2 Sistemas de representação numérica

Para que uma máquina digital execute cálculos, primeiro é preciso representar os dados numéricos por meio de estados dos dispositivos físicos disponíveis na máquina. Estes por sua vez são, em geral, dispositivos bi-estáveis, que podem operar a velocidades altas, de modo confiável, sendo usados, portanto, para representar 2 estados. Os dispositivos usados para operar como bi-estáveis, principalmente nas máquinas de 2ª geração, eram colocados em pontos extremos de operação: os núcleos magnéticos saturados em uma direção ou outra e os transistores "cortados" ou saturados. Posteriormente os projetos tem feito os dispositivos operar longe dos limites, em faixas de operação, mas de modo que possam ser tratados como dispositivos estáveis em um ou outro estado. O sistema numérico que mais se adapta a isso é o binário.

Outra classe de máquinas utilizou a representação decimal codificado em binário (BCD, *Binary coded decimal*) e embora a sua prevalência não seja grande hoje em dia, o seu estudo ainda é importante para se mostrar a diferença de processamento, necessária ao se usar diferentes representações. As representações surgem por alguma razão. A representação BCD não é tão eficiente sob o ponto de vista da máquina, mas pode ser mais conveniente do ponto de vista dos usuários das máquinas. É bom lembrar que o sistema decimal é o mais adequado para computação manual, manutenção de registros, etc.

Na prática o usuário da máquina raramente escreve suas instruções de modo que sequer se lembre a real linguagem que a máquina "entende". Qualquer computador atual é equipado com programas de conversão que transformam os símbolos usados pelo usuário em códigos de máquina.

Hoje em dia a comunicação com as máquinas pode ser feita praticamente em inglês, usando sistema decimal e expressões matemáticas. A máquina se encarrega de fazer as conversões. Contudo, futuros engenheiros e engenheiras: Quem faz as máquinas fazerem tudo isto? Resposta: Vocês! É por isso que vocês precisam saber

tudo sobre sistemas de numeração, exatamente para que o usuário comum, cada vez mais nem sequer precise saber que isso existe. O conhecimento do sistema numérico interno das máquinas é importante sob o ponto de vista de desenvolvimento de programas e sistemas e de manutenção da máquina. Saber detalhes de construção e da representação numérica nos computadores, bem como detalhes de suas operações aritméticas básicas é o que na Engenharia "*separa os meninos dos homens*". Qual será o conhecimento necessário para os próximos desenvolvimentos? Qual é o futuro da interação homem-máquina e o que será que vocês precisarão saber para possibilitar que isto ocorra cada vez melhor? Qual é a sua opinião? Devemos procurar evoluir no sentido de construir máquinas que substituam o cérebro humano?

Como já foi dito, a escolha natural para construção de máquinas digitais genéricas é o sistema binário de numeração. Isto não quer dizer que outras técnicas de numeração não estejam em teste, em particular para máquinas dedicadas, como o uso de números primos de Mersenne (números da forma  $2^n - 1$ ) e outros. Contudo, no caso das máquinas de uso geral, a soma das vantagens do uso de sistemas não-binários ainda não superou a soma das desvantagens. É preciso desenvolver mais pesquisa sobre o assunto.

### 3.2.1 Notação posicional

Um número decimal qualquer pode ser considerado como um polinômio de potências de dez. Na verdade o número é uma sequência dos coeficientes do polinômio. Por exemplo, 423.12 corresponde ao polinômio:

$$4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 1 \times 10^{-1} + 2 \times 10^{-2}$$

Números decimais como este são ditos representados na base ou raiz 10 por que há 10 dígitos para representá-los (0, 1, 2 ..., 9) a partir dos quais todo o sistema numérico é representado. De modo semelhante podemos representar qualquer número em qualquer base  $b$ . Vamos escrever o número como  $(N)_b$ . Assim, um número qualquer pode ser escrito:

$$(N)_b = c_n b^n + c_{n-1} b^{n-1} + \dots + c_0 b^0 + c_{-1} b^{-1} + c_{-2} b^{-2} + \dots + c_{-m} b^{-m} \quad (1)$$

ou de modo compacto:

$$(N)_b = \sum_{i=-m}^n c_i b^i$$

Os coeficientes do polinômio acima ficam no intervalo  $0 \leq c_i \leq b-1$ . Para números decimais, o símbolo "." é denominado de ponto decimal; no caso geral denomina-se ponto da raiz ou ponto da base. A parte do número à direita do ponto é denominada de parte fracionária e a parte à esquerda de parte inteira.

### *Números inteiros e bases numéricas*

Números expressos na base 2 são chamados de números binários. Estes são frequentemente usados nos computadores já que utilizam apenas dois coeficientes. Os valores de 0 a 15 são representados na Tabela 2.1 em várias bases. Como não há coeficientes para valores maiores ou iguais a 10 são usadas letras de A a F. A base 8 é chamada de octal e a 16 de hexadecimal. Números escritos em octal ou hexadecimal são frequentemente usados para representar números binários de modo mais compacto. Números escritos nestas bases podem ser convertidos para binários pela conversão de cada dígito individual. Esta propriedade é válida por que 8 e 16 são potências de 2. Quando o número não for potência de 2 a conversão é mais complexa e será vista logo adiante.

Tabela 2.1 Números inteiros em várias bases

2	4	8	10	12	16
Binária	Quaternária	Octal	Decimal	Duodecimal	Hexadecimal
0000	00	00	00	00	0
0001	01	01	01	01	1
0010	02	02	02	02	2
0011	03	03	03	03	3
0100	10	04	04	04	4

0101	11	05	05	05	5
0110	12	06	06	06	6
0111	13	07	07	07	7
1000	20	10	08	08	8
1001	21	11	09	09	9
1010	22	12	10	0A	A
1011	23	13	11	0B	B
1100	30	14	12	10	C
1101	31	15	13	11	D
1110	32	16	14	12	E
1111	33	17	15	13	F

### *Conversão entre bases*

Para fazer uso dos sistemas numéricos não-decimais é preciso ser capaz de converter um número expresso em uma base em outro número expresso em outra base. Uma maneira de fazer isto é usar o polinômio (Eq. 1) que representa o número. Por exemplo, o número binário  $(1011.101)_2$  tem como correspondente decimal:

$$1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$$

ou  $11 + 5/8 = 11.625$

Esta técnica é uma maneira geral de converter da base  $b_1$  para outra base  $b_2$ . Este método será chamado de método polinomial. O método consiste em:

1. Expressar o número  $(N)_{b_1}$  como um polinômio, com os números na base  $b_1$  usados como coeficientes do polinômio;
2. Avaliar o polinômio, usando aritmética na base  $b_2$ .

É claro que este método é muito usado quando se deseja converter para a base 10, por que, neste caso, utiliza-se aritmética decimal. Aproveitamos para dizer que a lista de passos acima constitui um algoritmo. Um algoritmo é um conjunto de passos ou uma lista de instruções, especificando uma sequência de operações, que irá dar a resposta para qualquer problema de um determinado tipo. As características importantes de um algoritmo são: (a) ser completo e não depender de intuição ou habilidade especial da pessoa que o executa; (b) sempre funcionar.



Exemplos:

$(26)_8$  será  $(22)_{10}$

1)  $2 \times 8_1 + 6 \times 8_0$

2)  $2 \times 8 + 6 \times 1 = (16)_{10} + (6)_{10} = (22)_{10}$

$(26)_8 = (42)_5$

1)  $2 \times 8^1 + 6 \times 8^0$

Lembrete: contas na base 5!

2)  $2_5 \times (13)_5 + (10)_5 + 1_5 = (31)_5 + (11)_5 = (42)_5$

Como apresentado no lembrete, as operações aritméticas deverão ser efetuadas na base 5. Isto será revisto mais adiante quando estudarmos as operações aritméticas binárias. Contudo, não é sempre conveniente usar aritmética na base  $b_2$  para fazer a conversão de  $b_1$  para  $b_2$ . Um algoritmo para fazer esta conversão usando aritmética na base  $b_1$  será discutido a seguir. Esta discussão será especificamente para a base  $b_1 = 10$ , mas pode ser estendida ao caso geral. Este será denominado de método iterativo, desde que envolve repetidas multiplicações e divisões. Na conversão de  $(N)_{10}$  para  $(N)_b$  a fração e a parte inteira são convertidas separadamente. Primeiro considere a parte inteira, ou seja, a porção à esquerda do ponto decimal. O procedimento geral é dividir  $(N)_{10}$  por  $b$ , resultando  $(N)_{10} / b$  e um resto. O resto, chamado de  $c_0$ , é o dígito menos significativo (mais à direita) de  $(N)_b$ . O próximo dígito menos significativo  $c_1$  será o resto da divisão de  $(N)_{10} / b$  por  $b$ , e os dígitos sucessivos são obtidos continuando este processo. Uma forma conveniente de conduzir esta conversão está apresentada nos exemplos a seguir.

a)  $(23)_{10} = (10111)_2$

Divisor	Dividendo/Q	Resto	Sequência
2	23	1	- sig. $c_0$
2	11	1	$c_1$
2	5	1	$c_2$
2	2	0	$c_3$

<b>2</b>	<b>1</b>	<b>1</b>	<b>+ sig. c<sub>4</sub></b>
	<b>0</b>		

Exercício: a) Faça a divisão apresentando os números da forma que você aprendeu dividir na escola; b) mostre claramente onde fica o dígito menos significativo.

b)  $(23)_{10} = (27)_8$

A seta indica a direção do dígito mais significativo para o dígito menos significativo.

Divisor	Div/Quoc.	Resto
8	23	
8	2	7
	0	2



c)  $(410)_{10} = (3120)_5$

Divisor	Div/Quoc.	Resto
5	410	
5	82	0
5	16	2
5	3	1
	0	3



Agora considere a porção do número à direita do ponto decimal (i.é. a parte fracionária). O procedimento para converter é multiplicar a fração de  $(N)_{10}$  por  $b$ . Se o produto resultante for menor que 1, então o dígito mais significativo (mais à esquerda) da fração é zero. Se o produto for maior que 1, dígito mais significativo da fração é a parte inteira do produto resultante. O próximo dígito mais significativo é formado multiplicando a parte fracionária deste produto por  $b$  e tomando a parte inteira. Os outros dígitos são obtidos pela repetição deste processo. Veja a seguir uma maneira conveniente de fazer esta conversão.

a)  $(0.625)_{10} = (0.5)_8$

$$0.625 \times 8 = 5.000 \quad | \quad 0.5$$

b)  $(0.23)_{10} = (0.001110...)_{2}$

$0.23 \times 2 = 0.46$	$0.0$
$0.46 \times 2 = 0.92$	$0.00$
$0.92 \times 2 = 1.84$	$0.001$
$0.84 \times 2 = 1.68$	$0.0011$
$0.68 \times 2 = 1.36$	$0.00111$
$0.36 \times 2 = 0.72$	$0.001110 \dots$

c)  $(27.68)_{10} =$

D	Div/Q	R
2	27	
2	13	1
2	6	1
2	3	0
	1	1
	0	1

$(11011.101011 \dots)_2$

$0.68 \times 2 = 1.36$	$0.1$
$0.36 \times 2 = 0.72$	$0.10$
$0.72 \times 2 = 1.44$	$0.101$
$0.44 \times 2 = 0.88$	$0.1010$
$0.88 \times 2 = 1.76$	$0.10101$
$0.76 \times 2 = 1.52$	$0.101011$

$= (33.53)_8$

8	27	R
8	3	3
	0	3

$0.68 \times 8 = 5.44$	$0.5$
$0.44 \times 8 = 3.52$	$0.53$

Neste exemplo podemos ver a relação interessante entre as bases 2 (binária) e 8 (octal). Se agruparmos os dígitos binários (também chamados de bits que é uma contração de ***B**inary **D**igit*) em grupos de 3 e expressarmos como números decimais teremos os correspondentes dígitos octais. Veja novamente ( a seta indica a direção de agrupamento dos dígitos):

$$\begin{array}{ccccccc} \leftarrow & & & & & & \rightarrow \\ (110 & 101 & . & 001 & 110 & 100)_2 \\ (6 & 5 & & 1 & 6 & 4)_8 \end{array}$$

Esta conversão é tão simples que muitas vezes números octais são usados para representar os números binários por questões de compactação. Lembre-se de ajustar o número de bits para múltiplos de 3, caso contrário, em alguns casos a precisão adequada não seria cumprida. Erros serão introduzidos na fração. Para avaliar o número octal, dentro de cada bloco de 3 usamos a notação posicional na base 2 com aritmética decimal (dado que os dígitos octais seriam os mesmos em decimal). Por exemplo (no caso de número inteiro):

$$(1\ 101\ 011)_2 = (1\ 5\ 3)_8$$

onde 011 seria o dígito 3 por que seria, na notação posicional,  $0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 0 + 2 + 1 = 3$ . Se o número de dígitos for ajustado para múltiplos de 3 nas partes inteira e fracionária a direção da conversão poderia ser efetuada em qualquer direção.

Quando uma fração é convertida, a conversão pode não terminar por que pode ser que não seja possível representar a fração exata em um número finito de dígitos. Por exemplo, considere a conversão  $(0.1)_3$  para uma fração na base 10. O resultado será  $(0.333 \dots)_{10}$  que pode ser escrito com  $(0.\underline{3})_{10}$  para representar que 3 seria repetido indefinidamente.

---

Exercício: Faça a conversão de  $(0.1)_3$  para a base 10.

---

É fácil concluir que pela combinação dos dois métodos de conversão é possível converter entre duas bases arbitrárias quaisquer usando somente aritmética em uma terceira base. Por exemplo  $(16)_7$  para a base 3, convertendo antes para a base 10:

$$(16)_7 = 1 \times 7^1 + 6 \times 7^0 = 7 + 6 = (13)_{10}$$

Agora converta  $(13)_{10}$  para a base 3.

3	13	R
3	4	1
3	1	1
3	0	1

$$(16)_7 = (13)_{10} = (111)_3$$

### 3.2.2 Representação de números fracionários e negativos

A representação de frações segue naturalmente da notação posicional. Um símbolo denominado ponto da raiz ou da base é usado em uma sequência que representa uma fração ou um número misto. O ponto da raiz separa os dígitos da sequência para os quais os expoentes (potências) são maiores ou iguais a zero daqueles que são negativos.

A sequência  $c_1 c_2 \dots c_n \bullet c_{n+1} c_{n+2} \dots c_{n+m}$  representa a soma

$$\sum_{i=1}^{n+m} c_i b^{n-i}$$

onde  $c_i$  são dígitos, " $\bullet$ " é o ponto da raiz e  $b$  é a raiz ou base numérica. Na base 10 o ponto é chamado de ponto decimal e na base 2, ponto binário.

Vamos agora considerar o problema da representação de números negativos em uma máquina digital. A informação se um número é positivo ou negativo pode ser

$$(1 - 2c_0) \cdot \sum_{i=1}^n c_i b^{n-i}$$

armazenada em um único dígito binário. Vamos colocar o símbolo  $c_0$  na posição mais à esquerda da sequência e assumir que se  $c_0 = 0$  o número é positivo e se  $c_0 = 1$  o número é negativo. Denominamos  $c_0$  de bit de sinal. Agora, o valor da sequência pode ser sintetizado por:

Deste modo,  $c_0$  indica se o número é positivo ou negativo e a sequência  $c_1, c_2, \dots, c_n$  representa a magnitude do número. A sequência  $01010_2$  representa  $+1010_2 = 10_{10}$  e a sequência  $11010_2 = -10_{10}$ . Do mesmo modo a sequência  $19424_{10}$  representa  $-9424_{10}$  e  $09424_{10}$  representa  $+9424_{10}$ .

Dois outros sistemas de representação são frequentemente usados. São eles o chamado complemento da base (*true complement system*) e o complemento da base diminuída. Ambos são usados para facilitar a adição e subtração.


O complemento da base de um número  $c = c_1c_2 \dots c_n$  é definido como o número

$$b^n - c$$

onde  $b$  é a base do sistema. O complemento de base de  $c = c_1c_2 \dots c_n$  pode ser formado simplesmente pela subtração de cada  $c_i$  de  $b-1$ , formando assim o complemento de  $c_i$  ( $c'_i$ ) e então adicionando 1 ao número resultante  $c' = c'_1c'_2 \dots c'_n$ .

Assim o complemento da base do octal  $2413_8$  é definido como  $8^4 - 2413_8 = 5365_8$  e pode ser formado como segue:

$$(8-1) = \text{base} - 1$$


$$(7-2) \times 8^3 + (7-4) \times 8^2 + (7-1) \times 8^1 + (7-3) \times 8^0 + 1 = 5364 + 1 = 5365_8$$

O complemento da base de  $7414_{10}$  é  $2586_{10}$  e do número  $1010_2 = 0110_2$ .

Por definição o complemento da base diminuída (lembre-se é diminuída de 1) de  $c = c_1c_2 \dots c_n$  é o número:

$$b^n - c - 1$$

onde  $b$  é a base. O complemento é formado pela subtração de cada dígito que forma o número  $c_i$  de  $b-1$ . Por exemplo  $2417_{10}$  torna-se  $7582_{10}$  e  $1010_2$  torna-se  $0101_2$ .

O complemento da base no sistema decimal é denominado de complemento de 10 e no binário complemento de 2. De modo similar os complementos de base diminuída nestes dois sistemas seriam respectivamente complemento de 9 e complemento de 1.

Estas representações numéricas são usadas para apenas para os números negativos, deste modo o complemento de 10 de  $+2417_{10}$  é o número negativo  $-7583_{10}$  e o complemento de 9 seria  $-7582_{10}$ . A grande virtude destes complementos é que são fáceis de serem formados e, se utilizados, as adições e subtrações podem ser facilitadas com apenas algumas regrinhas, mas lembre-se: existirão regras para se efetuar as operações mas as operações são simplificadas de modo que mesmo uma máquina simples poderia efetuá-las (o preço a pagar são as regrinhas!).

Vamos assumir que números negativos estejam armazenados na forma de complemento de 9 e que todos os números serão representados em sequências de comprimento 5,  $c_0c_1c_2c_3c_4$ , onde  $c_0$  (dígito de sinal) indica se o número é positivo ou negativo e os dígitos seguintes representam a magnitude do número. Neste caso, a sequência  $00019_{10}$  representa o número  $+19_{10}$  e a sequência  $10114_{10}$  representa o

número  $-9885_{10}$ . Podemos adicionar os números simplesmente adicionando as sequencias:

$$\begin{array}{r} 19 \\ -9885 \\ \hline -9866 \end{array} = \begin{array}{r} 00019 \\ 10114 \\ \hline 10133 \end{array}$$

É preciso cuidado para ver se a soma ou diferença não é maior ou menor que os números que podem ser representados. Não podemos, por exemplo, representar a soma de 9484 com 5966 neste sistema, nem a soma de -5966 e -5976 (verifique!). Outra regra é necessário cumprir:  $c_0$  é interpretado como binário e deve, assim, ser adicionado como tal, qualquer que seja a base do número. Se um vai um (*carry*) é gerado do bit do sinal no sistema de base diminuída, então 1 é adicionado à posição  $c_n$  da soma. Exemplo:

$$\begin{array}{r} -27 \\ -54 \\ \hline -81 \end{array} \quad \begin{array}{r} 19972 \\ 19945 \\ \hline 1 \ 19917 \\ \quad \swarrow +1 \\ \hline 19918 \end{array} \quad \text{(retorno de vai um)}$$

A maioria das máquinas representam números negativos em complemento de 2 ou de 1 e representam números na forma fracionária. Vamos agora assumir complemento de 1 e números com 5 dígitos, sendo  $c_0$  o bit de sinal:  $c_0c_1c_2c_3c_4$ . Os dígitos de  $c_1$  a  $c_4$  representam a fração binária com o ponto binário à esquerda de  $c_1$ . Assim,  $-0.0101_2$  é representada por  $11010_2$  e  $+0.0101_2$  por  $00101_2$ . A soma destes números resulta zero ou  $11111_2$  que é o zero negativo, neste sistema. Há ainda outra representação para o zero, ou seja,  $00000_2$  chamada de zero positivo.

Em complemento de 2 o número  $+0.001_2$  é representado por  $0001_2$ . O número  $-0.001_2$  é representado por  $1111_2$ . Agora, no complemento de 2, qualquer vai um (*carry*) que ocorra da posição  $c_0$  durante a adição é desprezado como segue:

$$\begin{array}{r} -10 \\ 11110 \end{array}$$



---

<u>+10</u>	<u>00010</u>
00	100000

(drop!)

Assim 00000 é zero e há apenas 1 zero neste sistema de representação. Podemos definir o valor de uma sequência  $c = c_0c_1c_2 \dots c_n$  na representação fracionária em complemento de 2 como:

$$\delta_2(c) = -c_0 + \sum_{i=1}^n c_i 2^{-i}$$

Este sistema particular de representação de frações binárias irá representar números no intervalo:

$$-1 \leq \delta_2 \leq 1 - 2^{-n}$$

O sistema de representação numérica binária de frações em complemento de 1 fornece para o número  $c = c_0c_1c_2 \dots c_n$  a seguinte definição geral:

$$\delta_2(c) = c_0(2^{-n} - 1) + \sum_{i=1}^n c_i 2^{-i}$$

De modo similar os complementos da base e base diminuída podem ser usados em outras bases. Podemos definir o valor de uma sequência  $c = c_0c_1c_2 \dots c_n$  na base  $b$  como fração em complemento da base por

$$\delta_b(c) = -c_0 + \sum_{i=1}^n c_i b^{-i}$$

e na base  $b-1$

$$\delta_{b-1}(c) = c_0(b^{-n} - 1) + \sum_{i=1}^n c_i b^{-i}$$

No complemento da base as frações podem, então, ser representadas no intervalo

$$-1 \leq \delta_b \leq 1 - b^{-n}$$

e na base diminuída

$$-1 + b^{-n} \leq \delta_{b-1} \leq 1 - b^{-n}$$

Como o único sistema, fora o binário, que é usado de algum modo é o BCD, vamos descrever algumas de suas características. Sistemas em outras bases tem características semelhantes.

O número negativo na representação em complemento de 9 fracionário pode ser facilmente obtido. O bit de sinal muda de valor e os outros dígitos tem seus valores

subtraídos de 9. Assim o negativo de 04454 é 15545 e o negativo de 12426 é 07573. Adição e subtração podem ser feitas neste sistema usando somente adição. Dois pontos devem ser considerados: a formação do complemento de 9 ( no BCD o 9 representa a base diminuída) e o *end-around-carry* (retorno de vai um) que as vezes deve ser adicionado ao dígito  $c_n$  da soma.

Vamos primeiro considerar a formação do complemento de 9. Se dígitos decimais são representados em uma máquina usando BCD, a construção é simplificada se o complemento de 9 de um certo código representando um dígito decimal puder ser facilmente formado. O modo mais direto para formar o complemento de 9 seria simplesmente complementar cada dígito binário no código (grupo de bits para representar 1 decimal). Mas note que se isto for feito no código 8421 o resultado não será correto. Por exemplo, se complementarmos bit a bit o conjunto  $1000_2$ , que corresponde ao decimal  $8_{10}$ , obtemos  $0111$  que representa  $7_{10}$  e não  $1_{10}$ . No código 2421 iria funcionar:  $1111 = 9_{10}$  teria como complemento  $0000 = 0_{10}$ ;  $0001 = 1_{10}$  complementado seria  $1110 = 8_{10}$  e neste caso basta complementar cada dígito binário para obter de qualquer número o seu complemento de 9.

Outro código que permite a formação do complemento de 9 facilmente é o excesso de 3. Este foi usado nas primeiras máquinas da Harvard e é formado pela adição de  $3_{10} = 0011_2$  aos números obtidos no código 8421. Assim o código excesso de 3 representa  $3_{10} = 0110_2$  e  $4_{10} = 0111_2$ ,  $5_{10} = 1000_2$ , etc. Veja o que ocorre com o complemento de 9. O número  $0111_2 = 4_{10}$  complementado fica  $1000_2 = 5_{10}$  e  $0100_2 = 1_{10}$  complementado fica  $1011_2 = 8_{10}$ , etc. O código excesso de 3 não é um código pesado havendo dificuldades quando os cálculos envolvem números positivos e negativos. (*Veja pag. 315, Bartee, Lebow & Reed*).

No caso do 8421 o complemento de 9 pode ser formado, mas usando a subtração ou soma em complemento de 1 ou 2, respeitando eventual geração de retorno de vai um. Por exemplo: Dado o número  $1000_2 = 8_{10}$  o seu complemento de 9 deve ser  $1_{10}$ , ou seja,

$$\begin{array}{r} 1001 = 9_{10} \\ \underline{1000} \end{array}$$

$$0001 = 1_{10}$$

Usando o complemento de 1 para efetuar a subtração, ficaria:

$$\begin{array}{r} + \quad 1001 \\ \quad 0111 \quad (\text{complemento de 1 do número acima}) \\ \hline 10000 \\ \swarrow \quad 1 \quad (\text{retorno de "vai um" por causa da operação em complemento de 1}) \\ \hline 0001 \end{array}$$

Neste sistema o complemento de 10 é mais complexo operacionalmente por que para cada grupo de dígitos é preciso achar o complemento de 1 e adicionar 1. Isto complica o modo de sintetizar circuitos para executar automaticamente as operações. Neste caso, se um vai um ocorre no sinal ele é descartado.

## 4. Aritmética binária

Os computadores modernos usam o sistema binário para representar os números e consequentemente usam a aritmética binária nas operações. Vamos ver aqui as técnicas para realizar as operações aritméticas elementares na base 2. Para executar aritmética na base 10 é necessário saber a tabuada. Nas outras bases isto também ocorre. Há tabelas que ajudam a memorizar. Um ponto importante é que a adição é no fundo a operação fundamental. A partir dela todas as outras podem ser efetuadas. Desde que se saiba operar com números negativos a subtração pode ser transformada em adição e a partir daí obter as outras operações, tais como, multiplicação, divisão e exponenciação.

### 4.1 Operações básicas

#### Adição Binária

S	Vai um ( <i>carry</i> , C)
0+0 = 0	0
0+1 = 1	0
1+0 = 1	0
1+1 = 0	1

Os números devem ser alinhados pelo ponto da base e cada coluna somada da direita para a esquerda, levando-se em conta a possibilidade de geração de "vai-um". Neste caso o valor do "vai-um" é adicionado como uma linha a mais na soma. De modo geral cada coluna é somada na base 10 e dividida pela base. O resto entra como a soma para aquela coluna e o quociente vai como "vai-um" para a próxima coluna.

Exemplo:

$$\begin{array}{r}
 \text{C} = \quad 11\ 11 \\
 \quad 1001.011 \\
 \quad 1101.101 \\
 \hline
 \text{Soma} = \quad 1\ 0111.000_2
 \end{array}
 \qquad
 \begin{array}{r}
 \quad 9.375_{10} \\
 \quad 13.625_{10} \\
 \hline
 \quad 23.000_{10}
 \end{array}$$

## Subtração binária

A tabela para subtração binária está apresentada a seguir:

	Empréstimo D (Borrow, B)
0-0 = 0	0
0-1 = 1	1
1-0 = 0	0
1-1 = 0	0

A subtração é feita colocando-se o minuendo sobre o subtraendo alinhados pelo ponto binário. Se um *borrow* ocorre e o bit mais à esquerda do minuendo é 1 ele é mudado para 0 e o processo de subtração continua da direita para a esquerda.

Transbordo:	1
Trocas:	0
Minuendo:	<del>1</del> 0
Subtraendo:	01
<hr/>	
Diferença:	01

Se, por outro lado, ocorre transbordo e o bit mais à esquerda, do minuendo, é zero, este zero é trocado por 1, assim como todos os outros bits sucessivos que forem zero. O primeiro bit 1 à esquerda será trocado por zero e a subtração continua.

Transbordo:	1	
Trocas:	011	
Minuendo:	<del>1</del> 000	24
Subtraendo:	— 10001	17
<hr/>		<hr/>
Diferença:	00111	7

## 4.2 Aritmética binária usando complementos

É possível evitar este processo de subtração, usando a representação de complemento para os números negativos. Vamos discutir este assunto para números

binários fracionários e posteriormente faremos a generalização. O complemento de 2 ( ${}^2B$ ) de uma fração binária B é definido como se segue:

$${}^2B = (2-B)_{10} = (10-B)_2$$

Assim,  ${}^2(0.1101) = 10.0000 - 0.1101 = 1.0011$ . Um método simples para se fazer esta operação é notando que **10.0000 = 1.1111 + 0.0001**. Deste modo,

$$\begin{aligned} 10.0000 - 0.1101 &= (1.1111 + 0.0001) - 0.1101 \\ \text{ou} \\ 1.1111 - 0.1101 + 0.0001 \end{aligned}$$

Esta subtração é particularmente fácil já que basta inverter todos os bits e obter o resultado:

$$\begin{aligned} 1.1111 - 0.1101 &= 1.0010 \text{ e daí basta adicionar o que falta, ou seja, } 0.0001, \text{ ficando} \\ 1.0010 + 0.0001 &= 1.0011. \end{aligned}$$

Isto significa que para se obter o complemento de 2 de uma fração basta inverter todos os bits e adicionar a fração 0.00 ... 01. A aplicação do complemento de 2 está na possibilidade de se obter a diferença pela soma do complemento. Assim,

$$A-B = A + {}^2B = (A + 10 - B)_2 = (10 + (A-B))_2$$

Se  $A-B \geq 0$ , então  $(10 + A-B)_2$  será 10 + fração positiva (A-B). É possível obter A-B desprezando (*dropping*) o 1 mais à esquerda de  $A + {}^2B$ . Por exemplo:

A =	0.1110	A =	0.1110
B =	0.1101	B =	1.0011
Diferença	0.0001		<del>1</del> 0.0001

Se  $A-B < 0$ , então  $A + {}^2B = (10 - \text{mod}(A-B))_2$  é exatamente igual a  ${}^2|A-B|$ , o complemento de 2 de  $|A-B|$ .  
 Por exemplo:

A =	0.1101	A =	+ 0.1101	
B =	0.1110	B =	1.0010	
Diferença	0.0001			$1.1111 = {}^2(0.0001) = 1.1111$

Resumindo:

Se  $A-B \geq 0$  o resultado é  $A + {}^2B$  com dropping do 1 à esquerda;

Se  $A-B < 0$  o resultado é  ${}^2|A-B| = {}^2(B-A)$

O complemento de 1 é também bastante utilizado. A definição, neste caso, é a seguinte:

$${}^1B = (10 - 0.00 \dots 1 - B)_2$$

onde a localização do 1 na fração  $0.00 \dots 1$  corresponde ao dígito menos significativo de B. Uma vez que  $(10 - 0.00 \dots 1)_2$  é igual a  $01.11 \dots 1$ , é possível fazer  ${}^1B$  invertendo os dígitos de B e adicionando 1, antes do ponto da raiz. Assim,  ${}^1(0.1101) = (1.0010)_2$ .

Se somarmos  $A + {}^1B$  o resultado será  $(A-B + 10 - 0.00 \dots 1)_2$ . Se  $(A-B) > 0$ , podemos converter para A-B, removendo o  $(10)_2$  e adicionando 1 ao dígito menos significativo de  $A + {}^1B$ . Este procedimento é denominado retorno de vai-um (*end-around carry*). Por exemplo:

$\begin{array}{r} A = 0.1110 \\ B = 0.1101 \\ \hline 0.0001 \end{array}$	$\begin{array}{r} A = 0.111. \\ {}^1B = 1.0010 \\ \hline 10.0000 = A + {}^1B \\ \swarrow \\ 0.0001 \text{ (End-around carry)} \\ \hline A-B = 0.0001 \end{array}$
--	---

Se  $A-B < 0$ , então  $A + {}^1B$  será o complemento de 1 do módulo de A-B, ou seja,  ${}^1|A-B|$ . Por exemplo:



$$\begin{array}{r}
 A = 0.1101 \\
 B = 0.1110 \\
 \hline
 0.0001
 \end{array}
 \quad
 \begin{array}{r}
 A = 0.1101 \\
 {}^1B = 1.0001 \\
 \hline
 1.1110
 \end{array}
 = {}^1(0.0001) = 1.1110$$

$\nearrow$   
 ${}^1(0.1110 - 0.1101) =$

Mostrar que:  
 ${}^2B = {}^2(B \cdot 2^2) / 2^2$

### Generalizando o complemento de frações:

Agora, o complemento da base de uma fração F é definida por

$${}^bF = (10 - F)_b$$

O complemento da base diminuída é:

$${}^{b-1}F = (10 - F - 0.00 \dots 1)_b$$

Estes procedimentos são usados para formar os complementos e eventualmente executar as operações de subtração.

### Generalizando o complemento de números mistos:

Neste caso estamos apresentando ao lado o conceito de módulo e mostrando com isto o significado do complemento com base no módulo do número.

$${}^bN = (100 \dots 0. - N)_b \quad (M-N) + 1 \quad M = 2^n - 1 \text{ (definição de módulo)}$$

$${}^{b-1}N = (100 \dots 0. - N - 0.00 \dots 1)_b \quad M-N$$

onde 100 ... 0 contém 2 dígitos a mais que qualquer inteiro encontrado na subtração.

Por exemplo: Se  $N = 11.01$  então

$$\begin{aligned}
 {}^2(N) &= 1000.00 - 11.01 \\
 &= 111.11 - 11.01 + 0.01 \\
 &= 100.10 + 0.01 \\
 &= 100.11
 \end{aligned}$$

$$\begin{array}{r}
 M = 11.10 \\
 N = 11.01 \\
 \hline
 0.01
 \end{array}
 \quad
 \begin{array}{r}
 M = 11.10 \\
 {}^2N = 100.11 \\
 \hline
 1000.01
 \end{array}$$


 Descarta (Drop!)

### Operação deslocamento

Para efetuar multiplicação e divisão é comum a necessidade de passos intermediários de deslocamento do número para a direita ou esquerda. Deslocar um número na base  $b$  para a esquerda  $k$  vezes é equivalente a multiplicá-lo por  $b^k$  e para a direita equivale a multiplicá-lo por  $b^{-k}$ . Portanto, deslocar um número inteiro binário para a esquerda equivale a multiplicá-lo por 2 e para a direita a dividi-lo por 2.

$$(N)_b = \sum_{i=-m}^n c_i b^i = (c_n c_{n-1} \cdots c_1 c_0 \cdots c_{-1} c_{-2} \cdots c_{-m})_b$$

Deslocando  $k$  posições para a direita ou para a esquerda resulta:

$$\sum_{i=-m}^n c_i b^{i+k} = b^k \cdot \sum_{i=-m}^n c_i b^i = b^k \cdot (N)_b$$

Manipulação semelhante mostra o que ocorre com o deslocamento para a direita. Deslocar um número binário  $k$  posições para a esquerda equivale a multiplicá-lo por  $2^k$ . Por exemplo:

$$(110.101)_2 = (6.635)_{10}$$

$$(1.10101)_2 = 2^{-2}(6.625) = (6.625/4)_{10} = (1.65625)_{10}$$

$$(11010.1)_2 = 2^2(6.625)_{10} = (4 \times 6.625)_{10} = (26.5)_{10}$$

ou seja

$$\sum_{i=-m}^n c_i b^{i-k} = b^{-k} \cdot \sum_{i=-m}^n c_i b^i = \frac{(N)_b}{b^k}$$

### 4.3 Multiplicação binária

A tabela para multiplicação binária é a seguinte;

0x0 = 0  
 0x1 = 0  
 1x0 = 0  
 1x1 = 1

Exemplo: multiplicar 110.10 por 10.1

Multiplicando	110.10
Multiplicador	10.1
	11010
	00000
	11010
	-----
	10000.010

Para cada dígito do multiplicador que é igual a 1, um produto parcial é formado consistindo do multiplicando deslocado de modo que seu dígito menos significativo fique alinhado com o dígito 1 do multiplicador. Na verdade este produto nulo pode ser omitido. O produto final é obtido pela soma dos produtos parciais. O ponto binário é colocado no produto, usando a mesma regra da multiplicação decimal.

A técnica mais simples para multiplicação de números negativos é usar o processo já descrito para multiplicar as magnitudes. O sinal do produto é determinado separadamente, e o produto é feito negativo se multiplicando e multiplicador tiverem sinais diferentes. É possível efetuar multiplicação direta de números negativos representados em complemento. Isto é feito usando um esquema de recodificação chamado de algoritmo de Booth que também acelera a multiplicação.

### 4.4 Divisão binária

A divisão é a operação aritmética mais complexa. Divisões longas em decimal requerem operações de tentativa e erro. Por exemplo, ao dividir 362 por 46 é preciso

primeiro saber que 46 é maior que 36 e então chutar quantas vezes 46 cabe em 362. Se um primeiro chute for 8 ao efetuar  $8 \times 46 = 368$  vemos que este número é maior que 362 e daí o próximo chute seria 7. Este processo de tentativa e erro é mais simples em binário por que há menos número de opções nesta base. Para implementar a divisão binária em um computador, é necessário especificar um algoritmo de divisão. Dois algoritmos diferentes, chamados de restauração (*restoring*) e sem restauração (*non-restoring*) do dividendo são usados.

A divisão com restauração do dividendo é processada da seguinte forma: no primeiro passo, o divisor é subtraído do dividendo com seus dígitos mais à esquerda, alinhados. Se o resultado é positivo, um "1" é colocado como um dígito do quociente correspondendo ao dígito mais à direita do dividendo do qual um dígito do divisor foi subtraído. O próximo dígito mais à direita do dividendo é juntado ao resultado, que então passa a ser o próximo dividendo parcial e o processo se repete.

Se o resultado da subtração do divisor do dividendo for negativo, um "0" é colocado no quociente e o divisor adicionado ao resultado negativo para restaurar o dividendo original. O divisor é então deslocado uma posição para a direita e uma subtração é efetuada novamente. O processo da divisão por restauração é ilustrado no exemplo a seguir.

Divisor = 1111  
 Dividendo = 1100

$$\begin{array}{r}
 \begin{array}{cccccc}
 q_0 & q_{-1} & \dots & & q_{-5} \\
 x & x & \dots & & x
 \end{array} & \longleftarrow \text{quociente} \\
 1111 \overline{) 1100.0000} & \longleftarrow \text{número de dígitos após o ponto} \\
 \underline{1111} & \\
 - 0011 & 
 \end{array}$$

Se o dividendo é menor que o divisor, a primeira subtração já terá resultado negativo. Outro ponto é que a subtração mostrada é na verdade o divisor menos o dividendo e o sinal negativo indica que o módulo foi obtido pela subtração do menor do maior e o sinal é negativo por que o divisor é maior. Neste caso, ao  $q_0$  deve ser atribuído o valor 0 (zero).

$$\begin{array}{r}
 \begin{array}{cccccc}
 q_0 & q_{-1} & \dots & & q_{-5} \\
 0. & x & \dots & & x
 \end{array} & \longleftarrow \text{quociente} \\
 1111 \overline{) 1100.0000} & \longleftarrow \text{número de dígitos após o ponto} \\
 \underline{1111} & \\
 - 0011 & \\
 + 1111 & \\
 \hline
 1100 & 
 \end{array}$$

$$\begin{array}{r}
 11000 \\
 - 01111 \\
 \hline
 10010
 \end{array}$$

Como o resultado foi negativo o divisor foi adicionado ao resultado negativo para restaurar o dividendo e o divisor deslocado uma posição para a direita para que nova subtração seja efetuada. Agora o divisor ficou menor e com isto o resultado é positivo. Assim, ao  $q_{-1}$  será atribuído o valor 1 e o próximo número do dividendo (neste caso não há mais e portanto zero!) é "baixado" para ser parte do dividendo do qual o divisor será novamente subtraído.

$$\begin{array}{r}
 \begin{array}{cccccc}
 q_0 & q_{-1} & \dots & & q_{-5} \\
 0. & 1 & \dots & & x
 \end{array} & \longleftarrow \text{quociente} \\
 1111 \overline{) 1100. 0 \text{ } 0 \text{ } 0 \text{ } 0} & \longleftarrow \text{número de dígitos após o ponto} \\
 \underline{1111} & \\
 - \quad 0011 & \\
 + \quad 1111 & \\
 \hline
 1100 & \\
 11000 & \\
 - 01111 & \\
 \hline
 10010
 \end{array}$$

No próximo passo então o divisor será novamente subtraído deste novo dividendo (veja o zero baixado em negrito)

$$\begin{array}{r}
 \begin{array}{cccccc}
 q_0 & q_{-1} & q_{-2} & \dots & & q_{-5} \\
 0. & 1 & 1 & \dots & & x
 \end{array} & \longleftarrow \text{quociente} \\
 1111 \overline{) 1100. 0 \text{ } 0 \text{ } 0 \text{ } 0} & \longleftarrow \text{número de dígitos após o ponto} \\
 \underline{1111} & \\
 - \quad 0011 & \\
 + \quad 1111 & \\
 \hline
 1100 & \\
 11000 & \\
 - 01111 & \\
 \hline
 10010 & \\
 \underline{1111} & \\
 000110 &
 \end{array}$$

Nesta iteração o resultado foi positivo,  $q_2$  foi feito igual a 1, novo zero baixado e nova subtração deverá ser feita.

$$\begin{array}{r}
 \begin{array}{ccccccccc}
 q_0 & q_{-1} & q_{-2} & q_{-3} & \dots & & q_{-5} & & \\
 0. & 1 & 1 & 0 & \dots & & x & & \leftarrow \text{quociente}
 \end{array} \\
 1111 \overline{) 1100.0000} & \leftarrow \text{número de dígitos} \\
 \underline{1111} & \text{após o ponto} \\
 - 0011 & \\
 + 1111 & \\
 \hline
 1100 & \\
 11000 & \\
 - 01111 & \\
 \hline
 10010 & \\
 \underline{1111} & \\
 000110 & \\
 \underline{1111} & \\
 - 1001 & \\
 + 1111 & \\
 \hline
 01100 & \\
 - 1111 &
 \end{array}$$

Nesta iteração o resultado foi novamente negativo. Lembre-se que o valor mostrado é na verdade  $1111 - 0110$  já que  $0110$  é menor que  $1111$  e o sinal marcado como negativo. Assim,  $q_{-3}$  ficou igual a 0. Como anteriormente o dividendo foi restaurado e portanto  $1111$  (divisor) adicionado ao resultado (veja que esta adição é a subtração de  $1001$  de  $1111$ ), resultado  $0110$  que era o dividendo anterior (foi portanto, restaurado). O divisor é então deslocado para a direita para se efetuar a nova subtração. --- Notem a diferença de baixar o zero (colocado em negrito) e deslocar o divisor!). Veja também que o divisor foi deslocado com relação ao dividendo que era  $0110$ . A nova subtração será então  $1100 - 1111$ . Novamente o resultado será negativo.

$$\begin{array}{r}
 \begin{array}{ccccccc}
 q_0 & q_{-1} & q_{-2} & q_{-3} & q_{-4} \dots & q_{-5} & \\
 0. & 1 & 1 & 0 & 0 \dots & x & \leftarrow \text{quociente}
 \end{array} \\
 \\
 \begin{array}{r}
 1111 \overline{) 1100.0000} \quad \leftarrow \text{número de dígitos após o ponto} \\
 \underline{1111} \\
 q_0=0 \quad - \quad 0011 \\
 + \quad 1111 \\
 \hline
 1100 \\
 11000 \\
 - \quad 01111 \\
 \hline
 q_1=1 \quad 10010 \\
 \underline{1111} \\
 q_{-2}=1 \quad 000110 \\
 \underline{1111} \\
 q_{-3}=0 \quad - \quad 1001 \\
 + \quad 1111 \\
 \hline
 01100 \\
 - \quad 1111 \\
 \hline
 q_{-4}=0 \quad - \quad 0011 \\
 \underline{1111} \\
 11000 \\
 \underline{1111} \\
 q_{-5}=1 \quad (resto) \quad 1001
 \end{array}
 \end{array}$$

Da iteração anterior nova restauração foi necessária e a  $q_{-4}$  foi atribuído o valor 0. A última iteração resultou em resultado positivo (1001) levando  $q_{-5}$  para 1 e como o número de dígitos após o ponto já estava determinado e seria igual a 5 a divisão termina deixando como resto o ultimo resultado da subtração.

Na divisão sem restauração a etapa de adicionar o dividendo parcial negativo ao divisor é omitida e o divisor deslocado é somado ao dividendo parcial negativo. Este passo de somar o dividendo parcial negativo ao divisor deslocado substitui os dois passos de adicionar o divisor e então subtrair o divisor deslocado. Isto pode ser justificado por: Se X representa um dividendo parcial negativo e Y o divisor, então  $1/2Y$  representa o divisor deslocado uma posição para a direita. Adicionando o divisor e então subtraindo o divisor subtraído produz

$$X+Y-1/2Y = X+1/2Y$$

enquanto que adicionando o divisor deslocado produz o mesmo resultado,

$$X+1/2Y$$

Os passos para divisão de 1100 por 1111 estão apresentados a seguir:

Divisor: 1111

Dividendo: 1100

$q_0 \ q_{-1} \ \dots \ q_{-5}$	$0. \ 1 \ \dots \ x$	← quociente
1111	1100. 0 0 0 0	← número de dígitos após o ponto
	1111	
-	00110	$q_{-0}=0$
+	1111	desloca e soma
+	10010	$q_{-1}=1$
-	1111	desloca e subtrai
+	00110	$q_{-2}=1$
-	1111	desloca e subtrai
-	10010	$q_{-3}=0$
	+ 1111	desloca e soma
	- 000110	$q_{-4}=0$
	+ 1111	desloca e soma
	1001	$q_{-5}=1$ (resto)

#### 4.5 Operações em outras bases

Apenas para que não passe em branco, vamos exercitar algumas operações em outras bases numéricas. O exercício pode ser útil para estimular você a pensar sobre o assunto a não ficar preso apenas ao que é o convencional.



## 5. Células Binárias

Máquina digital é um tipo de máquina na qual os dados estão representados de modo discreto. Isto é feito pelo uso de dispositivos que exibem um número finito de estados e estes são colocados em correspondência com os valores das variáveis a serem representadas. Os dispositivos físicos usados para este propósito são quase que exclusivamente de dois estados. São dispositivos binários que são denominados de células binárias. Os tipos de células binárias em uso são muito diferentes e variados quanto ao comportamento. Mesmo quando um fenômeno físico associado a um certo dispositivo é basicamente contínuo, utiliza-se o modo binário de operação por causa da simplicidade e confiabilidade.

Como um dispositivo binário pode representar somente um dígito binário, é evidente que para representar valores de variáveis contínuas com qualquer precisão ou para representar inteiros de qualquer magnitude, os dispositivos bi-estáveis devem ser manipulados como conjuntos. Na prática os dados armazenados em uma máquina digital são divididos em conjuntos de caracteres ordenados chamados de palavras. Em uma máquina uma palavra é, então, um conjunto ordenado de caracteres com um significado. O termo palavra se refere a informação armazenada e não aos dispositivos físicos nos quais a informação é armazenada. Nos referiremos ao conjunto ordenado de dispositivos físicos usados para armazenar uma certa porção de dados como registradores.

### Célula binária de armazenamento

Vamos inicialmente imaginar a célula binária do ponto de vista ideal e depois sob o ponto de vista de dispositivos reais. A Figura 4.1 ilustra uma função com os estados assumidos por uma célula binária, no tempo.

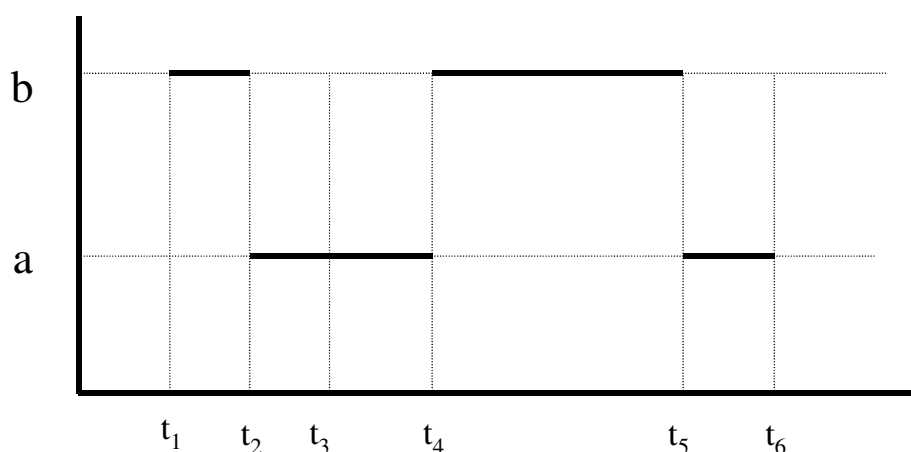


Figura 5.1 Valor do estado *versus* tempo em uma célula binária ideal. Os valores a e b são valores exatos assumidos pela variável binária. Os pontos  $t_1$ , ...,  $t_6$  são instantes no tempo.

Primeiro vamos descrever a célula como um dispositivo primário com as seguintes propriedades:

1. Possui dois estados possíveis que iremos chamar de A e B;
2. Qualquer mudança de estado (de A para B ou de B para A) ocorre instantaneamente, isto é, em instantes discretos de tempo  $t_1, t_2, \dots, t_j, \dots$
3. Uma medida de alguma propriedade física da célula tal como magnetização, tensão ou corrente em algum ponto, sempre resulta em um número  $a$  se a célula está no estado A ou  $b$  se a célula está no estado B. Esta definição de medição implica que o dispositivo de medição não perturbe a célula, ou equivalentemente, não extraia ou drene energia da célula. Como, pela propriedade 2 a célula muda de estado instantaneamente, uma medição não tem nenhum significado nos instantes de transição  $t_1, t_2, \dots, t_j, \dots$

Se fizermos um experimento no qual medimos o estado físico da célula binária em todos os instantes  $t \neq t_j$ , e plotarmos os resultados destas medições como função do tempo, nós obtemos uma função binária como apresentada na Figura 5.1. Assumimos, sem perda de generalidade que  $a < b$ .

Uma célula binária é então um dispositivo físico que pode armazenar uma função binária (que assume dois valores) do tempo. A determinação do estado da célula em um tempo  $t$  é equivalente a "avaliar" a função no instante  $t$  e ter como saída um bit de informação (mais tarde veremos que estamos falando da informação no conceito definido por Shannon). A célula binária é o elemento fundamental de armazenamento de um processador de dados digital. De modo correspondente a função binária é a função matemática fundamental para representação de variáveis físicas nas máquinas. O fato do elemento fundamental de armazenamento ser binário e a função matemática bi-estável ("two-valued") é devido a maior facilidade de construção dos elementos físicos.

Existe uma correspondência natural um-para-um entre a célula binária e a função que ela armazena. Vamos, as vezes, considerar a célula binária como uma função tipo "two-valued function". Segue que em um instante  $t$  a célula  $x$  armazena a função com valor  $a$  e então pode ser tratada como uma função  $x$  com valor  $x(t) = a$ .

#### Exemplos de células binárias de armazenamento

Na realidade nenhum dispositivo físico pode apresentar as propriedades 2 e 3, i.e., chavear instantaneamente e fornecer um valor de tensão tão exato. Estas condições podem ser atenuadas de modo que dispositivos físicos não-perfeitos podem ainda guardar funções bi-estáveis.

Um dispositivo bi-estável não apresentará resultados precisos da leitura de seus estados mas, preferencialmente ele irá apresentar dois intervalos estreitos de valores, como apresentado na Figura 5.2.

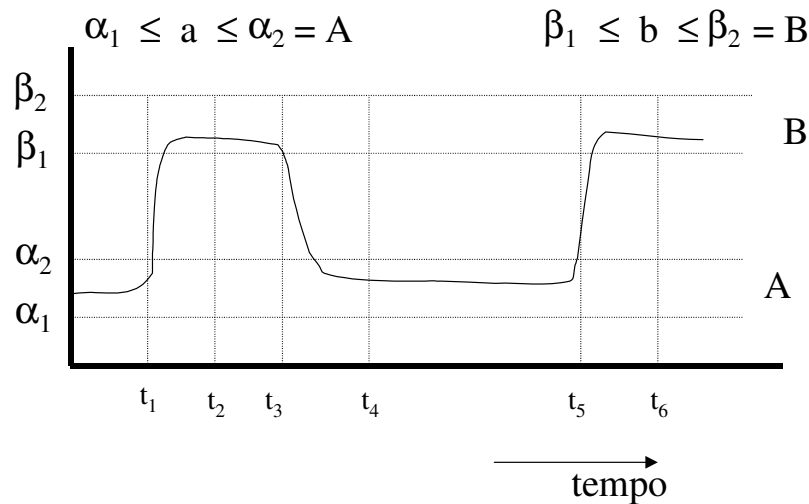


Figura 5.2 Célula binária real. A e B são os dois estados possíveis que ficam definidos pelas faixas alfa e beta. Os valores assumidos pela variável binária pode ser a ou b, definidos dentro das faixas.

O espalhamento no valor, ou seja a faixa ao invés de valor numérico, decorre do fato de que ambos, a célula e o mecanismo de medição estão afastados da condição ideal 3. Qualquer dispositivo real de medição absorve alguma energia da célula e isto por sua vez afeta o valor que está sendo medido. Se a faixa de valores for suficientemente pequena de modo que as faixas de variação dos dois valores não se interceptem, a propriedade 3 é efetivamente satisfeita. Nós denominamos a medição que não afeta o valor armazenado na célula de não-destrutiva.

De modo similar, qualquer dispositivo real requer um tempo finito para mudar de estado. Se o intervalo de tempo durante o qual a medição é feita não intercepta o intervalo para mudança de estado, então a propriedade 2 pode ser aceita. Veja o esquema da Figura 5.3.

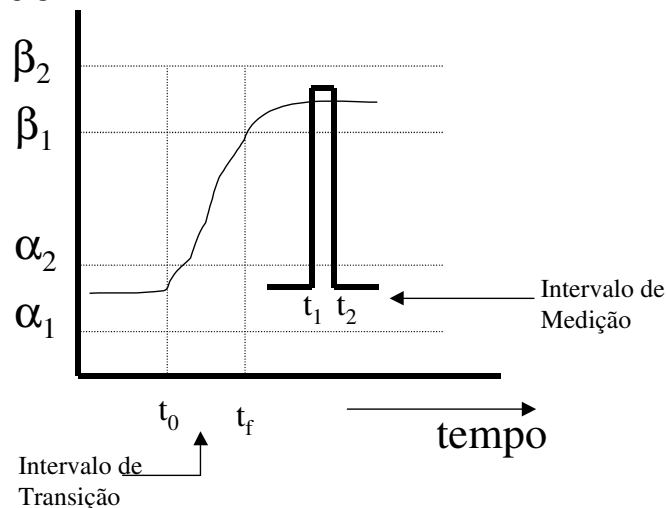


Figura 5.3 Relação entre tempo de medição e tempo de transição. Os intervalos estão apontados pelas setas.

O intervalo  $t_1 - t_2$  pode ser pré-fixado junto com um intervalo de medição de modo a não haver intersecção com o intervalo de chaveamento  $t_0 - t_f$ .

Talvez a célula binária de armazenamento mais comum seja o chamado Flip-Flop (assunto que iremos tratar mais tarde em detalhe) estático, como apresentado na Figura 5.4. Este é um circuito, geralmente consistindo de dois circuitos ativos como transistores (você verá este assunto em detalhe em outras disciplinas, mas vamos falar um pouco sobre os transistores mais adiante, também) e vários elementos passivos, com a propriedade de entrar em regime permanente, ou de se estabilizar com apenas um transistor ativo, ou seja, conduzindo corrente, de cada vez. Na Figura 5.4 o estado A ocorre quando T1 está conduzindo corrente e estado B quando T2 está conduzindo. A medição da tensão de saída no terminal X dará um valor próximo de um valor a determinado pelos parâmetros do circuito no estado A e próximo de um valor b no estado B. Esta medição terá significado ou valor desde que seja feita fora do tempo de transição de um estado para outro.

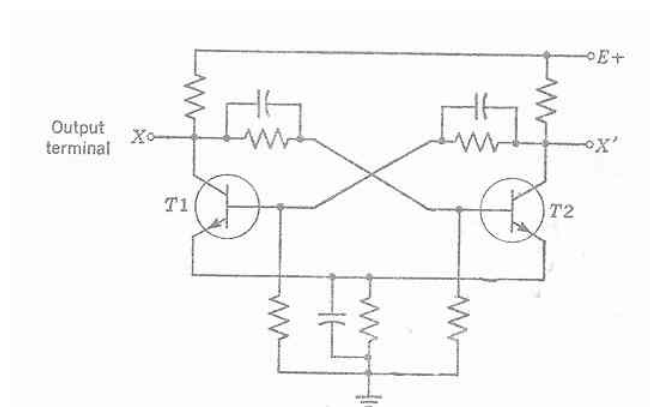


Figura 5.4 Flip-flop estático transistorizado. Bartee, Lebow & Reed, 1962

A Figura 5.5a ilustra um outro exemplo de célula binária, chamada de Flip-Flop dinâmico. Esta célula consiste de um elemento de atraso e um amplificador conectado em "loop". Os dois estados são representados pela presença ou ausência de um pulso no "loop". Suponha que o atraso seja de  $T$  segundos e um pulso, se presente, seja de  $\tau$  segundos a cada  $T$  segundos. A medição do estado consiste em observar a presença ou ausência do pulso durante  $\tau$  segundos (Figura 5.5 b).

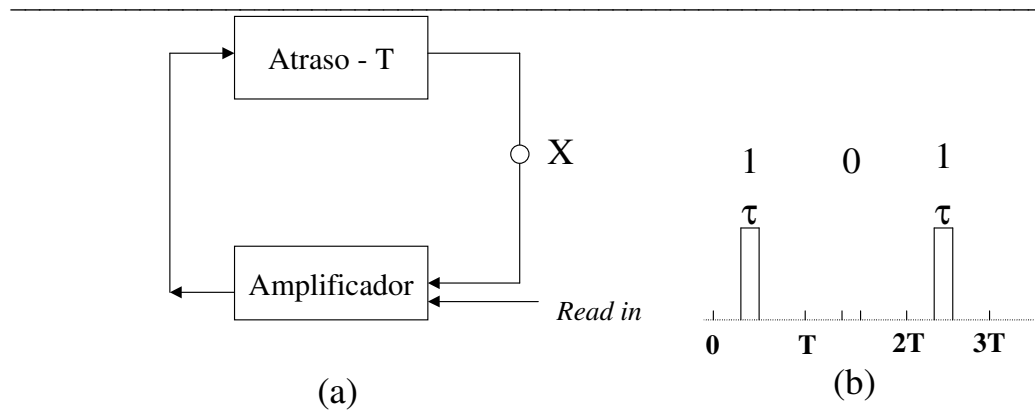


Figura 5.5. Flip-Flop dinâmico. Veja que os instantes de medição estão indicados por  $\tau$  e o valor lido representado na parte superior da parte b por 1 ou 0.

Um tipo diferente de célula binária é o núcleo magnético (Figura 5.6). O núcleo está no estado A quando as linhas de fluxo magnético estão no sentido anti-horário e no estado B quando se apresentam no sentido horário.

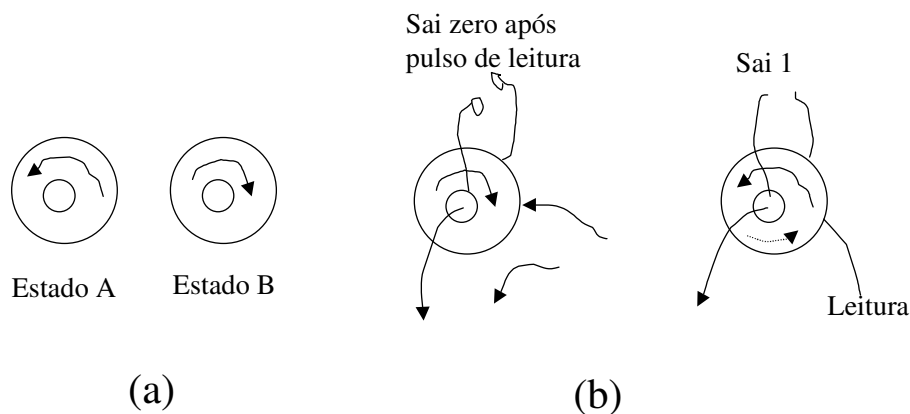


Figura 5.6. Célula binária a base de núcleo magnético (a). Estado A corresponde a magnetização no sentido anti-horário e B no sentido horário. A passagem de corrente (fio com seta) magnetiza o núcleo no sentido indicado. Se, como no caso mais à direita em (b), a passagem de corrente muda o sentido da magnetização anterior, o fio de leitura verá um pulso correspondendo a saída igual a 1 e destrói o estado anterior.

Veja que neste caso a leitura é destrutiva porque se a direção da corrente de leitura fizer aparecer um pulso na saída ("1") significa que o estado foi mudado para outro. Assim, após a leitura, se o resultado for "1" deve-se "passar corrente" no sentido inverso para restaurar o estado inicial.

Um outro tipo de célula binária seria o que está apresentado na Figura 5.7. Os estados A e B são obtidos pela abertura e fechamento da chave de modo que se estiver aberta a saída é zero volts ( $\cong a$ ) e quando fechada  $c = E^+$  (b).

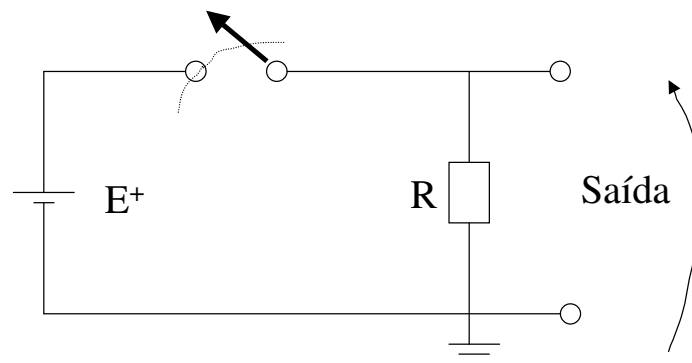


Figura 5.7. Célula binária baseada em uma chave.  $E^+$ , tensão da fonte de alimentação, R, resistor sobre o qual a saída é medida.

### O Registrador

Como mostrado até agora a célula binária é o elemento básico de armazenamento de uma máquina digital de processamento de dados. Uma extensão simples do conceito de célula binária é o conceito de registrador. Nós definimos um registrador como um conjunto ordenado de células binárias. As propriedades do registrador são derivadas imediatamente das propriedades da célula binária. Considere um registrador C de n células, composto pelas células  $c_1 c_2 \dots c_n$ . Cada célula  $C_i$  (1, 2, ..., n) armazena uma função bi-estável do tempo  $C_i(t)$ . Assim, em qualquer ponto  $t_1$ , cada  $C_i(t)$  tem um valor a ou b. Correspondentemente, C armazena n funções binárias do tempo  $c_1(t)$ ,  $c_2(t)$ , ...,  $c_n(t)$ , ou em outras palavras uma função vetorial do tempo,  $c(t)$ , com componentes  $c_i(t)$ ,  $i = 1, 2, \dots, n$ . Como no caso da célula, frequentemente nos referimos ao registrador como uma função binária vetorial do tempo. Assim, em um instante fixo t dizemos que o valor do registrador C é o valor da função que ele armazena, ou seja, a n-upla  $[c_1(t), c_2(t), \dots, c_n(t)]$ ,  $C_i(t) = a$  ou  $b$ .

Para exemplificar, considere o registrador de duas células  $C = (c_1 c_2)$ , armazenando os dois componentes da função vetorial do tempo  $c(t) = [c_1(t), c_2(t)]$ . Em um instante  $t_1$ , C tem o valor  $C(t_1) = [c_1(t_1), c_2(t_1)]$  com  $C_i(t) = a$  ou  $b$ . A Figura 5. 8 ilustra os quatro possíveis valores de  $c_1(t)$ , as coordenadas (a, a), (a, b), (b, a) e (b, b).

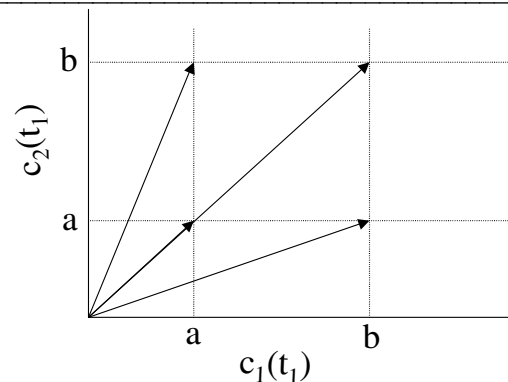


Figura 5.8. Faixa de valores de um registrador de duas células em um ponto específico do tempo. As setas são vetores com módulo igual ao valor da combinação correspondente dos valores de a e b.

Como apresentado, os registradores são dispositivos físicos que armazenam representações dos valores numéricos nas máquinas digitais. Nós demonstramos que o registrador armazena n-componentes de uma função vetorial binária. Para completar a figura devemos definir um meio pelo qual os valores de um registro estão associados a um conjunto de valores numéricos. A associação mais simples e mais natural é entre  $2^n$  diferentes valores que um registrador de n células pode assumir e os números inteiros de 0 a  $2^n - 1$ . Se associamos os inteiros 0 e 1 com os números a e b, respectivamente, e imaginando o ponto binário localizado à direita do registrador, então os valores do registrador são n-uplas de 0's e 1's que correspondem aos primeiros  $2^n$  inteiros não-negativos do sistema binário de numeração. Nós chamamos isso de representação inteira do registrador. Assim, por exemplo, se o registrador de 4 células ( $c_1, c_2, c_3, c_4$ ) tem o valor bbab, dizemos que na representação inteira o registrador armazena o inteiro binário 1101.

O valor decimal equivalente  $[\delta_I(c)]$  de um inteiro binário armazenado em um registrador c é obtido formando-se o polinômio:

$$\delta_I(c) = \sum_{i=1}^n c_i 2^{n-i}$$

com  $c_i = 0, 1$ ;  $i = 1, \dots, n$  e onde os valores  $2^{n-i}$  e a soma  $\delta_I(c)$  são expressos como números inteiros decimais. Assim, no exemplo acima, o decimal equivalente ao valor armazenado no registrador 1101 é 13. O sistema numérico octal é também frequentemente usado. Se as n células do registrador c são agrupadas em K sub-registradores de 3 células:

$$\begin{array}{llll} R^1 = & 0 & 0 & c_1 \quad n = 3k-2 \\ & 0 & c_1 & c_2 \quad n = 3k-1 \\ & c_1 & c_2 & c_3 \quad n = 3k \\ & \dots & & \end{array}$$

$$R^{k-1} = c_{n-5} \ c_{n-4} \ c_{n-3}$$

$$R^k = c_{n-2} \ c_{n-1} \ c_n$$

então o octal equivalente da representação inteira de  $c$  é uma  $k$ -upla

$$\delta_I(R^1), \delta_I(R^2), \dots, \delta_I(R^k)$$

Assim o octal equivalente de 1101 é 15.

Veja como:  $K = 2$  registradores de 3 células para um  $n = 4$

$$\begin{array}{ll} R^1 = c_{-1}c_0c_1 & \delta_I(R^1) = 1 \\ R^2 = c_2c_3c_4 & \delta_I(R^2) = 5 \end{array}$$

---

Exercício: Construa uma tabela com os valores do registrador (combinações aaaa, aaab, etc), valores binários, valores octais e valores decimais para a representação inteira de um registrador de 4 bits.

---

De agora em diante os valores de uma célula de um registrador,  $a$  ou  $b$ , serão sempre considerados representar os inteiros 0 e 1. Além disso, se nada for especificado o valor contido em um registrador será considerado inteiro. Consideraremos como sinônimos a representação inteira do número, em qualquer base, contida no registrador e o valor do registro. Vamos dizer que o valor do registrador  $X$  é  $(10)_{10}$ . Ou ainda, o registrador de 4 células com valor  $bbba$  terá o valor binário 1110 ou o valor octal 16, ou o valor decimal 14.

Outra representação de um registrador é assumido o ponto binário localizado do lado esquerdo do registrador. Assim, um registrador de 4 células 1110 terá a representação fracionária .1110. O valor decimal equivalente, neste caso, como já visto anteriormente será dado pelo polinômio:

$$\delta_F(c) = \sum_{i=1}^n c_i 2^{-i} = 2^{-n} \delta_I(c)$$

Similarmente, o equivalente octal é obtido agrupando as células de 3 em 3 começando agora da esquerda para a direita do registrador. Assim, a fração binária .1110 é equivalente à fração decimal  $7/8 = .875$  e a fração octal .70.



Ambas as representações inteira e fracionária são denominadas representações binárias uma vez que  $2^n$  valores do registrador de  $n$  células são feitos correspondentes a  $2^n$  números.

De modo diferente são as representações denominadas decimal-codificado em binário, ou BCD, como apresentado anteriormente. Nestas, grupos de células são associados a dígitos decimais. Por exemplo, suponha que o registrador  $C$  contém  $n = 4k$  células onde  $k$  é um inteiro. Divida  $C$  em  $k$  sub-registradores

$$R^1 = c_1, c_2, c_3, c_4$$

...

$$R^k = c_{n-3}, c_{n-2}, c_{n-1}, c_n$$

Tomando o caso em que  $K = 2$  e  $n = 8$

$$R^1 = c_1, c_2, c_3, c_4$$

$$R^2 = c_5, c_6, c_7, c_8$$

Restringimos cada registrador a assumir os valores 0, 1, 2, ..., 9 e atribuímos um único valor decimal a cada sub-registrador. Assim, a representação BCD do registrador  $C$  é uma  $k$ -upla

$$\delta_i(R^1), \delta_i(R^2), \dots, \delta_i(R^k)$$

com valor decimal

$$\delta_i(R^1)10^{k-1} + \delta_i(R^2)10^{k-2} + \dots + \delta_i(R^k)10^{k-k}$$

Nesta representação os  $10^k$  inteiros decimais são associados com  $10^k$  dos  $2^{4k}$  possíveis valores do registrador.

Exemplo:

$$K = 2 \text{ e } n = 8$$

$$C = 1000\ 0011 \quad R^1 = 1000 \quad \text{e} \quad R^2 = 0011$$

$$\text{BCD: } \delta_i(C) = \delta_i(1000)10^1 + \delta_i(0011)10^0 = 8 \times 10 + 3 \times 1 = 80 + 3 = 83$$

Claramente, muitas outras representações binárias são possíveis. O ponto binário (ou decimal), por exemplo, pode estar localizado em qualquer lugar com relação às células do registrador. Há também representações para números negativos.

Representações não-numéricas são também possíveis. Seja C um registrador de 5 células com  $2^5 = 32$  possíveis valores. As 26 letras do alfabeto podem ser representadas por C atribuindo a cada letra um dos 32 valores possíveis do registrador, como na Tabela 3.2. Em geral qualquer lista de **K** coisas pode ser representada por um registrador de  **$\log_2 K$**  células ou mais por uma associação de cada item com um único valor do registrador.

Tabela 5.2 Associação entre letras e valores de C

Valor do registrador	Representação do alfabeto
00000	A
00001	B
00010	C
00011	D
00100	E
...	...
11000	Y
11001	Z

### Generalização do conceito de registrador

Chegamos a uma definição de registrador como o dispositivo básico de um computador digital para o armazenamento de valores de variáveis e mostramos como ambos dados numéricos e não-numéricos podem ser armazenados nos registradores pelo estabelecimento de correspondência entre os números ou dados e os vários valores de um registrador. Esta definição, é ainda independente da implementação física do registrador.

Qualquer parte de um registrador (i.é., um sub-registrador) é um registrador. Ao considerar as representações inteiras octal e decimal assumimos que o registrador é um conjunto de sub-registradores no qual cada sub-registrador contém um valor octal ou decimal. Em qualquer registrador de n-células, cada uma das n células é um registrador de uma célula.

Há

$$\binom{n}{2} = n(n-1)/2$$

registradores de 2 células diferentes que mantem a ordem,  $\binom{n}{3}$  registradores de 3 células, etc. Há

$$\sum_{k=1}^n \binom{n}{k} = 2^n - 1$$

registradores diferentes incluindo o próprio registrador de  $n$  células como o último termo da soma. Nem todos estes sub-registradores possuem significados. Faz sentido considerar um sub-conjunto das células de um registrador como um sub-registrador separado somente se em determinado instante o seu conjunto é tratado independentemente do resto do registrador. Como um exemplo simples, suponha que um registrador  $C$  de 10 células as vezes sirva para armazenar binários inteiros e algumas vezes 2 letras do alfabeto. Os sub-registradores que teriam significado seriam as duas metades de  $C$  (registradores de 5 células) e o registrador  $C$  propriamente dito.

*Nota: A notação apresentada acima representa o coeficiente binomial que é definido por:*

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

O sub-registrador é apenas um caso especial da classe geral de registradores de funções. Resumidamente, um registrador, por definição é um dispositivo físico idealizado que armazena uma função vetorial do tempo, com cada componente podendo armazenar um dentre dois valores. O próprio registrador pode ser considerado uma função. Assim chegamos a uma definição matemática de um registrador de  $n$ -células como uma função cujo domínio é o eixo do tempo e valores no vetor  $n$ -dimensional  $V_n$ , no qual cada componente pode assumir um dos dois valores. Com base nesta definição matemática de registrador, podemos definir uma função ou transformação de um registrador como segue:

$$D = f(C)$$

de um registrador  $C$  é uma função com  $m$  componentes onde cada componente

$D_i$  ( $i = 1, 2, \dots, m$ ) de  $D$  é uma função binária de, em geral, dos componentes  $C_j$  ( $j = 1, 2, \dots, n$ ) do registrador  $C$ . Uma função de um registrador é uma função vetorial  $m$ -dimensional com algumas propriedades matemáticas do registrador. Uma função de um registrador não possui as propriedades físicas do registrador, já que não existe um conjunto de células de armazenamento, mas sim o valor da função no tempo  $t$  que é dependente do valor do registro físico no instante  $t$ . Um registrador é então análogo a uma variável independente e uma função de um registrador à variável dependente. Deste modo que para diferenciar o registrador da função nós referenciamos o registrador como **registrador independente** e função como **registrador dependente**. Agora vemos que um sub-registrador é apenas um caso especial de registrador dependente. O mapeamento de  $V_n$  em  $V_m$  é a projeção dos vetores de  $V_n$  no sub-espaço  $m$ -dimensional  $V_m$ . Neste caso especial o registrador dependente existe como um conjunto de células físicas e pode também ser considerado como um registrador independente. Um exemplo menos trivial é o de registrador dependente:

$$D = (D_1, D_2, \dots, D_n)$$

que é função do registrador independente

$$C = (C_1, C_2, \dots, C_n)$$

onde cada componente  $D_i$  é determinado por:

$$D_i = \begin{cases} 0 & \text{quando } C_i = 1 \\ 1 & \text{quando } C_i = 0 \end{cases}$$

Como veremos mais tarde o registrador dependente  $D$  é chamado de complemento de  $C$  e é uma função bastante comum. Outro registrador dependente comum é o registrador de  $n$ -células  $D$  definido por:

$$D_i = C_{i+1} \quad i = 1, 2, \dots, n-1 \\ D_n = 0$$

Esta função é evidentemente equivalente a deslocar cada célula de  $C$  uma posição para a esquerda e introduzir 0 na célula  $C_n$ . De modo análogo podemos introduzir funções de dois registradores. Seja  $C$  e  $D$  registradores independentes com componentes dados por:

$$C = (C_1, C_2, \dots, C_n) \\ D = (D_1, D_2, \dots, D_r)$$

Uma função de  $C$  e  $D$  é um registrador de  $m$  células  $F(C, D)$  no qual cada componente de  $F_i$  ( $i = 1, 2, \dots, m$ ) é uma função binária de todos os  $C_j$  e  $D_k$ .

Uma máquina digital é essencialmente uma coleção de registradores dependentes e independentes. Nas máquinas diferentes, o número de registradores independentes e o número e natureza dos registradores dependentes pode variar muito de acordo com o propósito da máquina. A menos deste fato, **qualquer máquina pode ser descrita considerando seus registradores dependente e independentes, e suas interações.** Para tratar da construção de máquinas sob este ponto de vista é preciso tratar a questão das funções de registradores muito mais detalhadamente. Isto não será feito neste curso. O objetivo aqui é deixar clara a possibilidade de que a construção das máquinas digitais pode ser feita totalmente baseada em operações matemáticas com registradores. Fica para a imaginação do aluno fazer exercícios sobre isto e possibilitar que suas “fantasias” sobre as possibilidades imaginadas se tornem realidade no futuro.

---

Exercício:

Um computador possui um total de 10 instruções ou operações que pode executar. Defina uma representação binária para as 10 instruções usando o mínimo número de células.

Defina uma representação binária para o computador do exercício anterior de modo que cada valor binário contenha dois e somente 2 dígitos 1. Qual é o mínimo número de células requerido?

Suponha que as operações do computador sejam O1, O2, ... O10. Defina uma representação binária para as operações tal que os valores para as sucessivas operações difiram em um e somente um dígito. Qual é o mínimo número de células requeridas para este que é chamado código de Gray? (veremos mais tarde)

---

## 6. A álgebra booleana

Em 1854 George Boole publicou um trabalho intitulado "*An Investigation of the Laws of Thought*", no qual ele descreveu o desenvolvimento de uma notação simbólica que podia ser usada em certos problemas de lógica formal e um cálculo para manipulação das expressões simbólicas. Subsequentemente vários outros matemáticos estenderam este trabalho criando uma nova área da matemática chamada de lógica simbólica da qual a álgebra de Boole era parte integrante. Apenas em 1938 é que Claude Shannon descreveu o uso da álgebra de Boole para o projeto de circuitos de chaveamento. Desde então a álgebra booleana passou a ser ferramenta fundamental no desenvolvimento de circuitos e máquinas digitais.

Uma álgebra booleana é definida sobre um conjunto com dois elementos, tipicamente falso e verdadeiro, alto e baixo, um e zero. Nós nos referimos a álgebra booleana binária, quando os elementos deste conjunto são "0" e "1". Neste sentido esta classe de álgebra pode ser definida como uma n-upla  $\{V, +, \bullet\}$  onde  $V$  é um conjunto e os símbolos  $+$  e  $\bullet$  são operações binárias sobre os elementos de  $V$ .

### Operações básicas:

(As duas primeiras operações definem a chamada álgebra de chaveamento que consiste de um conjunto  $V=\{0,1\}$  e das duas operações AND e OR como apresentadas abaixo.

**AND** (E), símbolo:  $\bullet, \wedge$

- 1)  $0 \bullet 0 = 0$
- 2)  $0 \bullet 1 = 0$
- 3)  $1 \bullet 0 = 0$
- 4)  $1 \bullet 1 = 1$

**OR** (OU), símbolo:  $+, \vee$

- 5)  $0 + 0 = 0$
- 6)  $0 + 1 = 1$
- 7)  $1 + 0 = 1$
- 9)  $1 + 1 = 1$

Veja mais adiante o uso desta operação

---

**XOR** (OU exclusivo), símbolo:  $\oplus$

- 10)  $0 \oplus 0 = 0$
- 11)  $0 \oplus 1 = 1$
- 12)  $1 \oplus 0 = 1$
- 13)  $1 \oplus 1 = 0$

*Postulado: proposição que se deve admitir antes de um raciocínio e que não será posta em dúvida.* Para esta álgebra são válidos os seguintes postulados:

Postulados:

(Os símbolos + e  $\bullet$  são operadores que denominamos OU (OR) e E (AND). Os elementos do conjunto V são literais a serem usados em expressões como variáveis booleanas)

**P1: se a e b são elementos de V, então**

P1.1  $a+b=b+a$

P1.2  $a\bullet b=b\bullet a$

**P2: Se a, b, c são elementos de V, então**

P2.1  $a+(b\bullet c)=(a+b)\bullet(a+c)$  A operação OR é distributiva sobre a operação AND

P2.2  $a\bullet(b+c)=(a\bullet b)+(a\bullet c)$  A operação AND se distribui sobre a OR

**P3: Para os dois elementos do conjunto  $V=\{0,1\}$ , há identidade tal que:**

P3.1  $0+a=a+0=a$

P3.2  $1\bullet a=a\bullet 1=a$

**P4: Para um elemento a de V existe a' denominado de complemento de a em V com as seguintes propriedades:**

P4.1  $a+a'=1$

P4.2  $a\bullet a'=0$

Vamos, a seguir, estudar os teoremas mais conhecidos da álgebra de Boole, procurando apresentar algumas demonstrações com o objetivo de introduzir o aluno a técnicas usadas para provar os teoremas:

Teoremas básicos : *proposições que podem ser demonstradas.*

**T1: Comutatividade:**  $a+b = b+a$

Se as operações básicas forem usadas pode-se verificar que as tabelas abaixo são simétricas com relação a diagonal.

OR	0	1
0	0	1
1	1	1

AND	0	1
0	0	0
1	0	1

Verifique as operações básicas com cuidado e confirme.

**T2: Distributividade:**  $a \bullet (b+c) = a \bullet b + a \bullet c$  ou  $a(b+c) = ab+ac$

Pode-se mostrar por indução perfeita. Neste caso deve-se considerar todas as possibilidades de valores para as variáveis:

a b c	a(b+c)	ab+ac
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	0	0
1 0 0	0	0
1 0 1	1	1
1 1 0	1	1
1 1 1	1	1

**T3: Elemento identidade:**  $a+0=a$

Deduz-se das operações básicas que  $0+0=0$  e  $0+1=1$  portanto 0 é o elemento identidade aditivo. O elemento identidade multiplicativo ( $a \bullet 1=a$ ) seria então verificado por  $0 \bullet 1=0$  e  $1 \bullet 1=1$  ou seja 1 é o elemento identidade multiplicativo.

**T4: Existência de complemento:**  $a + a' = 1$

Vamos assumir que 0 é o complemento de 1 e verificar que o postulado P4 se verifica:

a	a'	a+a'	a•a'
1	0	1	0
0	1	1	0

ou seja dado a existe a' que satisfaça P4.

**T5: Princípio da Dualidade:**

Toda identidade algébrica da álgebra booleana permanece válida se:

- 1) todos os operadores  $\bullet$  e  $+$  forem intercambiados;
- 2) todos os elementos 1 e 0 também forem intercambiados;

Exemplo:

$$a+b+c \bullet d=1 \rightarrow a \bullet b \bullet c+d=0$$

Para todo postulado da álgebra de Boole existe o seu dual que é obtido trocando-se 0s por 1s e as operações  $+$  por  $\bullet$ .

**T6: Unicidade do complemento:**

Prova (ref. 6). Vamos admitir que  $a \in V$  e que x e y sejam complementos de a. Podemos efetuar as seguintes operações:

$x$	$= x \bullet 1$	P3.2	identidade
	$= x \bullet (a+y)$	Por hipótese	y é o complemento de a
	$= x \bullet a + x \bullet y$	P2.2	distributividade
	$= a \bullet x + x \bullet y$	P1.2	comutatividade



$$\begin{aligned} &= 0 + x \bullet y \\ &= x \bullet y \end{aligned}$$

Por hipótese  
P3.1

x é o complemento de a  
identidade

Trocando x por y e vice-versa e repetindo todos os passos para **y** obtemos:

$$\begin{aligned} \mathbf{y} &= y \bullet x \\ &= x \bullet y \end{aligned}$$

P1.2

e portanto,  $x = y$ .

Após demonstrada a unicidade do complemento podemos considerar o símbolo ' (NOT) como uma operação de complementação.

### T7: Elemento nulo:

Para qualquer elemento a de V :

$$a + 1 = 1 \text{ e } a \bullet 0 = 0$$

$$\begin{aligned} a + 1 &= 1 \bullet (a + 1) && \text{P3.2} \\ &= (a + a') \bullet (a + 1) && \text{P4.1} \\ &= a + (a' \bullet 1) && \text{P2.1} \\ &= a + a' && \text{P3.2} \\ &= 1 && \text{P4.1} \end{aligned}$$

$$\begin{aligned} a \bullet 0 &= 0 + (a \bullet 0) && \text{P3.1} \\ &= (a \bullet a') + (a \bullet 0) && \text{P4.2} \\ &= a \bullet (a' + 0) && \text{P2.2} \\ &= a \bullet a' && \text{P3.1} \\ &= 0 && \text{P4.2} \end{aligned}$$

(Este último pode ser provado por meio de dualidade do anterior)

### T8: O complemento de 0 é 1 e o complemento de 1 é 0

Sabemos que  $a + a' = 1$  (P4.1) e que o complemento é único (T6). Assim podemos dizer que se  $a' = 1$  então a será 0 e vice-versa. Logo o complemento de 1 é 0 e vice-versa.

### T9: Lei da Idempotência:

Para todo a pertencente a V

$$a + a = a$$

$$a \bullet a = a$$

$$\begin{aligned} a + a &= (a + a) \bullet 1 && \text{P3.2} \\ &= (a + a) \bullet (a + a') && \text{P4.2} \\ &= a + (a \bullet a') && \text{P2.1} \\ &= a + 0 && \text{P4.2} \end{aligned}$$

---


$$= a \quad \text{P3.1}$$

$a \bullet a = a$  prova-se pelo teorema da dualidade (T5).

#### **T10: Lei da Involução:**

Para todo  $a$  pertencente a  $V$  podemos dizer que

$$(a')' = a$$

Pode-se dizer que  $(a')'$  e o próprio  $a$  são ambos complementos de  $a'$  e que pela unicidade do complemento  $(a')'$  e  $a$  tem que ser iguais.

#### **T11: Lei da Absorção**

Para  $a$  e  $b$  pertencentes a  $V$ ,

$$a + a \bullet b = a$$

$$a \bullet (a + b) = a$$

$$\begin{aligned} a + ab &= a \bullet 1 + ab && \text{P3.2} \\ &= a(1 + b) && \text{P2.2} \\ &= a(b + 1) && \text{P1.1} \\ &= a \bullet 1 && \text{T7} \\ &= a && \text{P3.2} \end{aligned}$$

$$a \bullet (a + b) = a \quad \text{Princípio da dualidade T5}$$

**T12:** Para todo par de elementos  $a$  e  $b$  de  $V$ ,

$$a + a'b = a + b$$

$$a(a' + b) = ab$$

$$\begin{aligned} a + a'b &= (a + a')(a + b) && \text{P2.1} \\ &= 1 \bullet (a + b) && \text{P4.1} \\ &= a + b && \text{P3.2} \end{aligned}$$

$$a(a' + b) = ab \quad \text{Princípio da dualidade (T5)}$$

#### **T13: Associatividade das operações $+$ e $\bullet$**

Para  $a$  e  $b$  pertencentes a  $V$ ,

$$a + (b + c) = (a + b) + c$$

$$a(bc) = (ab)c$$

Para demonstrar este teorema faremos uso de propriedades que só podem ser retiradas dos fundamentos da matemática associados a álgebra booleana. Neste caso a terceira operação descrita acima ( $\oplus$ ) é usada para definir um Anel Booleano que inclui propriedades especiais. Não é objetivo deste curso detalhar estes conceitos,

portanto vamos extrair as propriedades da matemática fundamental e usá-las como parte de operações válidas para a álgebra booleana. Deste modo as propriedades que vamos denominar de R1, ... Rn (*Ring*) apenas para lembrar a existência de fundamentos básicos da teoria dos anéis farão parte do conjunto de opções para demonstração de vários teoremas, sem nenhum rigor matemático no estabelecimento de cada uma das propriedades adicionais (Veja referência 4 para se esclarecer sobre o assunto).

Reverendo a operação ( $\oplus$ ) denominada de OU exclusivo (XOR, de *exclusive OR*)

### **XOR**

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

$$a \oplus 0 = a \quad R1$$

$$a \oplus 1 = a' \quad R2$$

$$a \oplus a = 0 \quad R3$$

$$a \oplus a' = 1 \quad R4$$

$$a \oplus b \oplus ab = a + b \quad R5$$

$$a \oplus b = a \cdot b' + a' \cdot b \quad R6$$

### **Provando T13,**

$$\begin{aligned} a+(b+c) &= \\ &= a \oplus (b \oplus c \oplus bc) \oplus a(b \oplus c \oplus bc) \quad R5 \\ &= a \oplus b \oplus c \oplus bc \oplus ab \oplus ac \oplus abc \\ &= (a \oplus b \oplus ab) \oplus c \oplus (a \oplus b \oplus ab)c = (a+b)+c \end{aligned} \quad \begin{array}{l} \text{distributiva} \\ \text{comutativa e R5} \end{array}$$

$$a(bc) = (ab)c \quad \text{Princípio da dualidade (T5)}$$

### **T14: Distributividade do produto sobre a soma:**

$$\begin{aligned} a(b+c) &= ab + ac \\ &= ab \oplus ac \oplus abac \\ &= ab \oplus ac \oplus abc \\ &= a(b \oplus c \oplus bc) \\ &= a(b+c) \end{aligned}$$

### **T15: Teorema de De Morgan:**

$$(a \cdot b)' = a' + b' \quad (a+b)' = a' \cdot b'$$

$$\begin{aligned}
 (a+b)' &= a' \cdot b' \\
 &= (a \oplus 1)(b \oplus 1) && \text{R2} \\
 &= a \cdot b \oplus a \oplus b \oplus 1 && \text{distributiva} \\
 &= (a+b) \oplus 1 = (a+b)' && \text{R5}
 \end{aligned}$$

$$\begin{aligned}
 (a \cdot b)' &= a' + b' \\
 &= a' \oplus b' \oplus a' b' && \text{R5} \\
 &= a \oplus 1 \oplus b \oplus 1 \oplus (a \oplus 1)(b \oplus 1) && \text{R2} \\
 &= a \oplus 1 \oplus b \oplus 1 \oplus ab \oplus a \oplus b \oplus 1 && \text{distributiva} \\
 &= a \oplus a \oplus b \oplus b \oplus 1 \oplus 1 \oplus ab \oplus 1 && \text{R3} \\
 &= 0 \oplus 0 \oplus 0 \oplus ab \oplus 1 && \text{R3} \\
 &= 0 \oplus ab \oplus 1 && \text{R1} \\
 &= (a \cdot b) \oplus 1 && \text{R2} \\
 &= (a \cdot b)'
 \end{aligned}$$

Tabela 1. Esta tabela contém um resumo de diversas propriedades da álgebra booleana, mostrando a identidade e seu dual. Lembre-se que o dual pode ser obtido trocando-se zeros por uns e intercambiando as operações + e  $\cdot$  em toda a expressão.

Propriedade	Identidade	Identidade Dual
Identidade	$a + 0 = a$	$a \cdot 1 = a$
Elemento nulo	$a + 1 = 1$	$a \cdot 0 = 0$
Idempotência	$a + a = a$	$a \cdot a = a$
Complemento	$a + a' = 1$	$a \cdot a' = 0$
Involução	$(a')' = a$	
Comutativa	$a + b = b + a$	$a \cdot b = b \cdot a$
Associativa	$(a+b)+c = a+(b+c)$	$(ab)c = a(bc)$

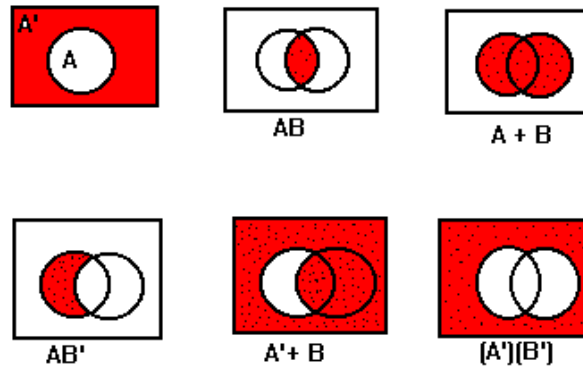
---

Distributiva	$a(b+c) = ab + ac$	$(a+b)(a+c) = a+bc$
Cobertura	$a + ab = a$	$a(a+b) = a$
Combinação	$ab + ab' = a$	$(a+b)(a+b') = a$
Consenso	$ab + a'c + bd =$ $bc + bd$	$(a+b)(a'+c)(b+d) =$ $(b+c)(b+d)$
De Morgan	$(a \cdot b)' = a' + b'$	$(a+b)' = a' \cdot b'$

( Retirado de ref.03, Tab.1.1, pag. 6)

(Nem todas as propriedades aqui listadas foram demonstradas ou discutidas. Mais tarde serão utilizadas e então apresentadas para melhor entendimento. Talvez fosse bom que o aluno se antecipasse em verificar algo sobre elas na bibliografia e ficasse preparado. Não se sabe o dia de amanhã, certo?)

Analogia entre operações lógicas e a teoria de conjuntos:



Outras funções podem ser também representadas. Experimente!

## Exercícios

Nos dois primeiros casos abaixo utilizamos a técnica de demonstração que consiste no desenvolvimento da expressão procurando aplicar as propriedades mais básicas.

Demonstrar:

$$\begin{aligned}
 (a+b) \cdot (a+c) &= a + (b \cdot c) \\
 (a+b) \cdot (a+c) &= a \cdot a + a \cdot c + b \cdot a + b \cdot c \\
 &= a + a \cdot (b+c) + b \cdot c \\
 &= a \cdot (1+b+c) + b \cdot c \\
 &= a \cdot (1) + b \cdot c \\
 &= a + b \cdot c
 \end{aligned}$$

Demonstrar:

$$\begin{aligned}
 (a \oplus b) &= ab' + a'b \\
 &= ab' \oplus a'b \oplus ab'a'b \\
 &= ab' \oplus a'b \\
 &= a(b \oplus 1) \oplus (a \oplus 1)b \\
 &= ab \oplus a \oplus ab \oplus b \\
 &= ab \oplus ab \oplus a \oplus b \\
 &= 0 \oplus a \oplus b \\
 &= a \oplus b
 \end{aligned}$$

Podemos, contudo demonstrar teoremas *por verificação* que é o que faremos para os casos a seguir.

Demonstrar:

$$a + a \cdot b = a$$

a	b	$a \cdot b$	$a + a \cdot b$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Neste caso montamos uma tabela com todas as combinações das variáveis que estão dos dois lados da identidade. Observa-se que a função booleana  **$a + a \cdot b$**  é igual a  **$a$**  qualquer que seja o valor de  **$b$** .

Demonstrar:

$$(a+b)' = a' \cdot b'$$

a	b	(a+b)	$(a+b)'$	a'	b'	$a' \cdot b'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Demonstrar:

$$(a \oplus b)' = a' \oplus b = a \oplus b' = a' \cdot b' + a \cdot b$$

$$a + a = a \oplus a \oplus a \cdot a = (a \oplus a) \oplus a = 0 \oplus a = a$$

$$b + a \cdot b = a' + b$$

$$a \cdot (a' + b') = a \cdot b'$$

$$(a+b) \cdot (a+c) = a + (b \cdot c)$$

$$a \cdot b + a \cdot b' = a$$

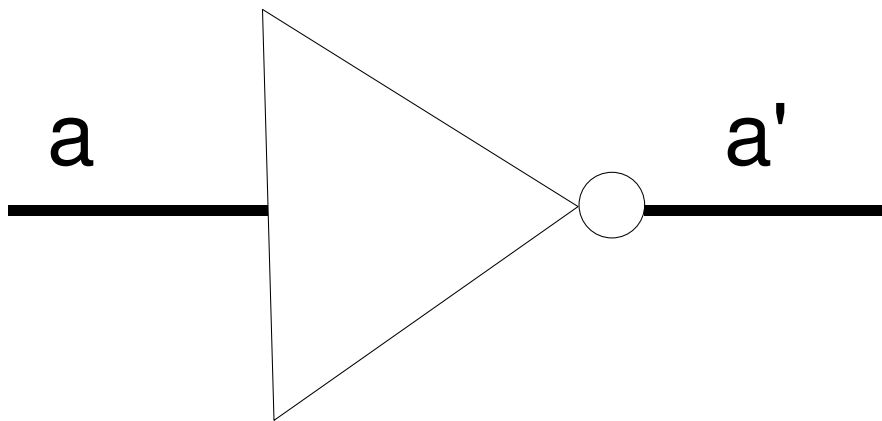
### Exercício:

Demonstrar todos os teoremas dados em classe. Use o bom senso! Aqueles mais simples, demonstre apenas por verificação. Todos que possibilitarem um certo trabalho algébrico demonstre pelo desenvolvimento das expressões. Justifique todos os passos com os postulados ou teoremas previamente demonstrados. Você precisa saber todos de cor!

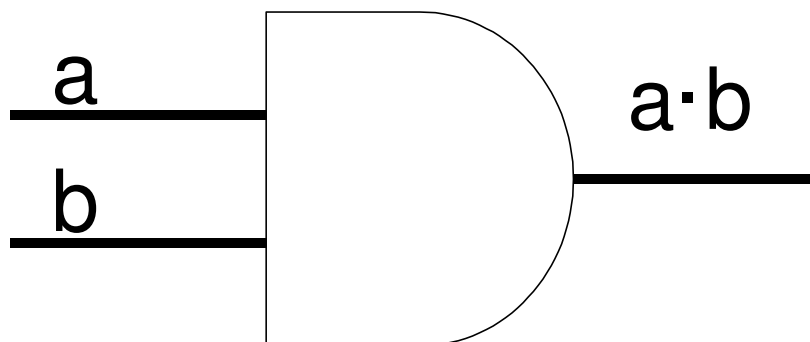
## 7. Blocos lógicos

Aqui serão descritos, de acordo com a operação algébrica correspondente, os principais símbolos que caracterizam os chamados blocos lógicos. Com estes blocos (daí o nome) pode-se construir uma infinidade de combinações que irão representar e sintetizar circuitos altamente complexos.

Função: **NOT** (inversor)

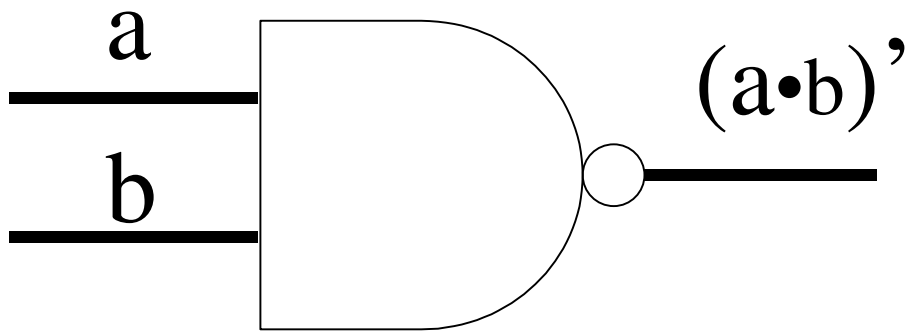


Função: **AND** (E)



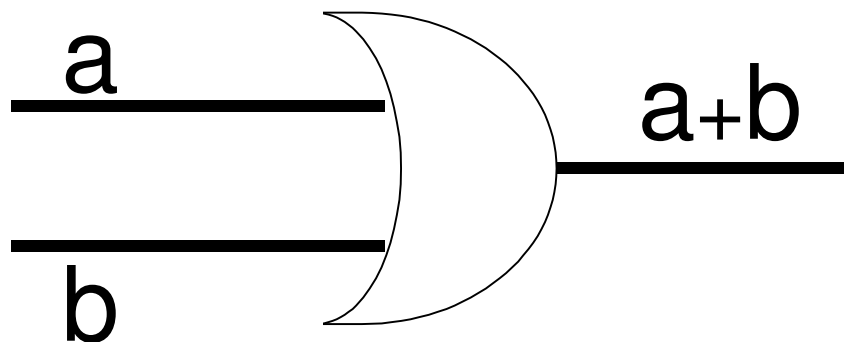


Função: **NAND** (NOT E)

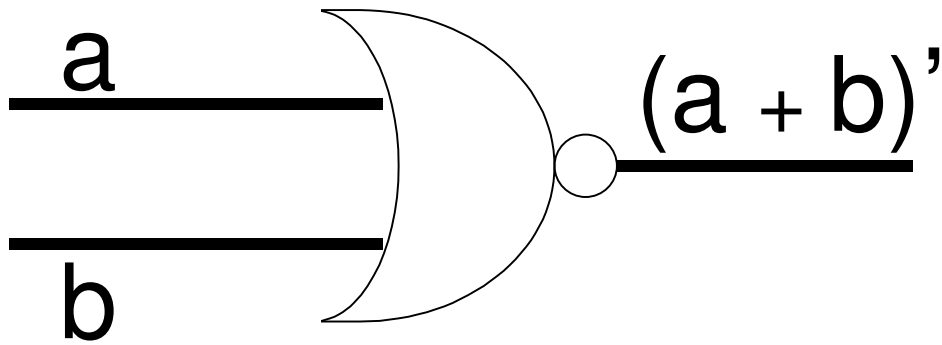


Esta operação é comutativa, i.é.,  $L = (ABC)' = (BAC)'$ ,  
mas não é associativa.

Função: **OR** (OU)

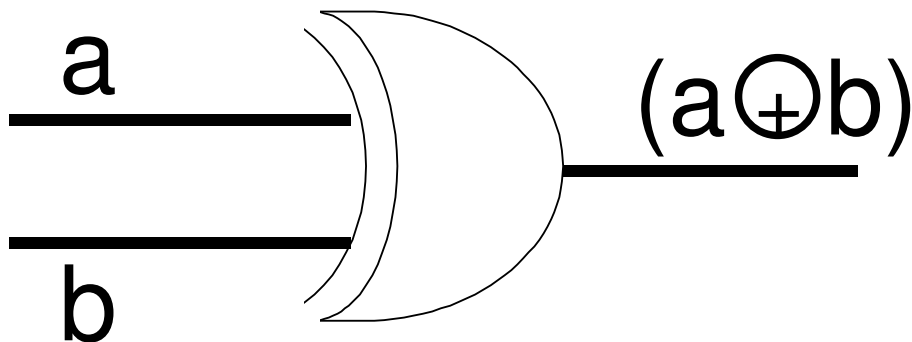


Função: **NOR**



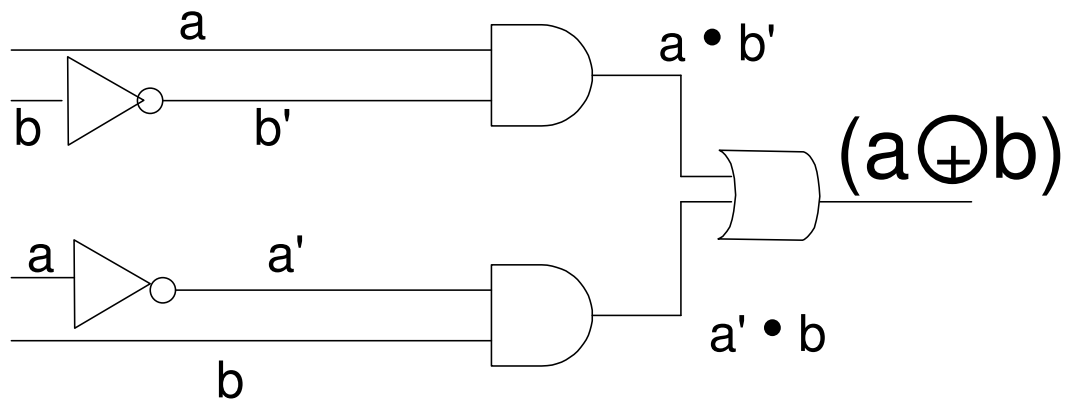
Esta operação é comutativa, i.é.,  $L = (A+B+C)' = (B+A+C)'$ ,  
mas não é associativa.

Função: **XOR**



Esta operação é **comutativa e associativa!**

A operação  $A \oplus B$  pode ser representada por meio de blocos lógicos já conhecidos conforme apresentado abaixo:



Vamos trabalhar um pouco com os blocos lógicos. A idéia é construir circuitos gerados pela conexão, aleatória ou não, de blocos e determinar a função final obtida. Neste ponto, podemos intuitivamente identificar os conceitos de *simplificação*, *dualidade*, *falhas* e *custo*.

### Exercício:

Um determinado circuito com 3 entradas binárias A,B e C pode ser projetado usando-se 2 blocos AND, de modo que a saída seja simplesmente o produto lógico ABC. Como sempre você não dispõe de todos os recursos necessários. Daí entra o papel do engenheiro! Como você faria se não dispusesse de circuitos (blocos) AND, mas sim de blocos OR e NOT ? Justifique. E se você dispusesse de 2 blocos OR, um bloco NAND e dois blocos NOT ? Justifique. (Obs.: considere os blocos NAND, AND e OR de duas entradas).

## 8. Tecnologia

### **Conceitos apresentados de modo simplificado leia refs. adicionais !**

Os componentes eletrônicos mais importantes atualmente são os chamados semicondutores. Os semicondutores podem atuar como condutores ou isolantes dependendo das condições. Um dos materiais semicondutores mais populares é o silício, um dos ingredientes encontrados na areia. O átomo de silício tem 4 elétrons na sua camada mais externa e assim pode se ligar **covalentemente** a até 4 outros átomos compartilhando seus elétrons. Um *cluster* de tais átomos forma um cristal. Há bastante silício na natureza. Cerca de 28% da crosta terrestre é formada por silício, que, contudo, não está na sua forma pura, como sempre...

Quando silício é misturado, ou "temperado" com outros elementos, tais como fósforo ou boro, a mistura tem propriedades elétricas muito úteis. O tempero é denominado de *doping*. Os elétrons das camadas mais externas dos átomos de silício podem ser compartilhados com os de átomos de boro ou fósforo. O átomo de boro tem apenas 3 elétrons na sua camada mais externa. Já o átomo de fósforo tem 5 elétrons na camada mais externa. Silício com elétrons de fósforo extras forma um cristal do tipo **N** (negativo). Quando a combinação é feita com um átomo deficiente em elétrons, como o boro, o tipo é **P** (positivo).

#### Cristal tipo P

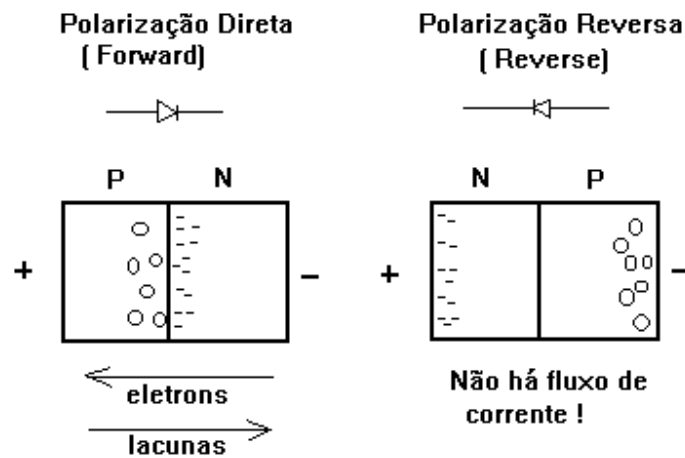
Um átomo de boro em um *cluster* de átomos de silício deixa uma "vaga", chamada de "lacuna" (buraco), para um elétron. E' possível que elétrons de átomos vizinhos "caiam" na tal lacuna. Deste modo a lacuna teria se movido para outro local através do silício.

#### Cristal tipo N

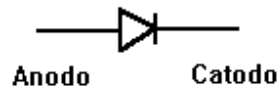
Um átomo de fósforo, em um *cluster* de átomos de silício, doa um elétron extra. Este elétron pode mover-se através do cristal. Silício tipo N pode, portanto, conduzir corrente elétrica. E também o tipo P! As lacunas "conduzem" a corrente.

#### O diodo

Ambos os tipos de silício (P e N) conduzem corrente elétrica. A resistência é determinada pelo número de buracos ou de eletrons extras. Deste modo, ambos podem funcionar como resistores, conduzindo eletricidade em qualquer direção. Contudo, pela formação de silício tipo P em uma pastilha de silício tipo N, os elétrons irão fluir em apenas uma direção. Este é o princípio básico da construção e funcionamento do diodo. A interface P-N é chamada de Junção PN.

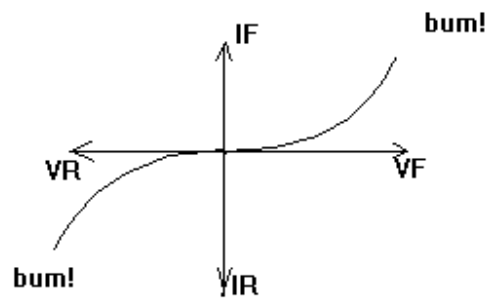


O símbolo do diodo é apresentado abaixo:



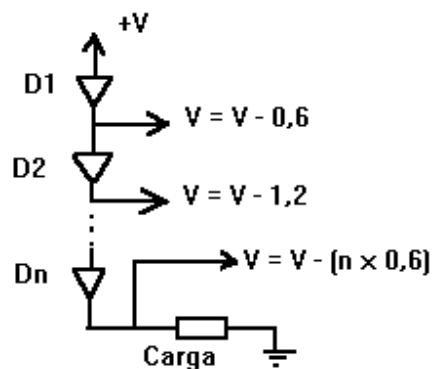
A condução se dará pelo diodo quando o anodo for mais positivo que o catodo. Além disso deve-se observar o seguinte:

- O diodo não irá conduzir até que a tensão direta seja maior que um certo limiar. Para diodos de silício esta tensão é de 0,6 V.
- Se a corrente direta for excessiva o componente pode fundir. Quando isto acontece o diodo pode conduzir em ambas as direções. O calor torna-se excessivo e pode vaporizar o componente.



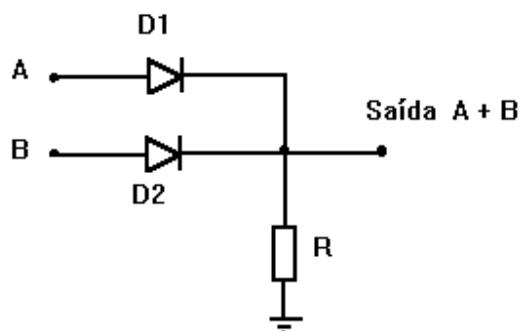
d) Existem vários tipos de diodos. Diodos para pequenos sinais (e.g. 1N914), retificadores de potencia (alta corrente), diodos Zener (e.g. 1N4733), LEDs e fotodiodos.

O circuito abaixo irá reduzir a tensão a partir da tensão da fonte de alimentação, de cerca de 0,6 V por diodo e constitui um exemplo de aplicação:



Sabendo, agora, como funcionam os diodos, vamos sintetizar circuitos lógicos com estes componentes e observar seu funcionamento.

OR

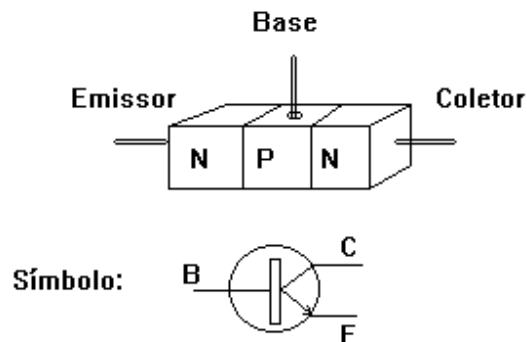


## O Transistor

É um dispositivo que pode, com uma corrente pequena, controlar uma corrente muito maior. Pode funcionar como **amplificador** ou **como chave**. Há duas grandes classes de transistores: Bipolares e de efeito de campo (FET - field effect transistor).

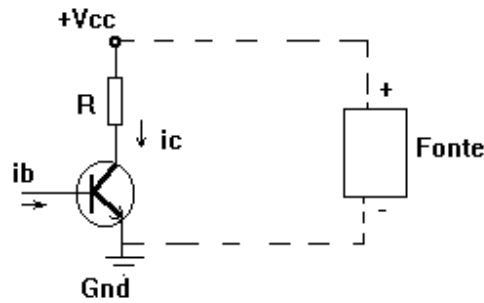
### *Transistor Bipolar*

Do ponto de vista de construção estes transistores são formados por combinações convenientes de junções N e P. Veja a figura abaixo:



Quando a base de um transistor NPN é ligada ao terra (0 V), não há fluxo de corrente do emissor para o coletor (diz-se que o transistor está "cortado"). Se uma polarização direta suficiente para vencer a barreira de potencial de 0,6 V é aplicada a base do transistor, uma corrente irá fluir **do emissor para o coletor**. Se esta corrente for próxima de  $V_{cc}/R$  dizemos que o transistor está saturado. Operando com correntes de base que levem o componente sempre de cortado para saturado e vice-versa, dizemos que o transistor está funcionando como **chave**. É neste modo de operação que os transistores são usados para síntese de circuitos digitais e portanto apenas este caso será estudado neste curso.

Para que o transistor opere é necessário que seja polarizado. Isto significa aplicar uma tensão no coletor, positiva com relação a do emissor. Veja a figura a seguir:



Se o transistor estiver polarizado, a corrente no coletor dependerá da corrente na base de modo que, para pequenas correntes de base,

$$I_c = \beta \cdot I_b$$

onde  $\beta$  é o ganho de corrente do transistor (pode variar de algumas dezenas até algumas centenas). Se  $I_b$  for suficientemente grande, o transistor atingirá a **saturação**. Este ponto ocorre quando a corrente no coletor tiver crescido tanto (você sabe que  $I_c = \beta \cdot I_b$ ) a ponto da tensão no coletor cair para zero (queda no resistor é igual a  $V_{cc}$ ). Ocorre saturação quando

$$I_{csat} \geq V_{cc} / R,$$

o que leva a

$$I_{bsat} \geq V_{cc} / (\beta R)$$

O transistor vai operar em dois modos:

cortado:

$$I_b = 0, I_c = 0 \text{ e } V_c = V_{cc}$$

Saturado:

$$I_b \geq V_{cc} / (\beta R), I_c = V_{cc} / R \text{ e } V_c = 0$$

Veja o caso real em que  $V_{cc} = 10 \text{ V}$ ,  $R = 1 \text{ k}\Omega$  e  $\beta = 50$ .  $I_{csat} = 10 \text{ V} / 1 \text{ k}\Omega = 10 \text{ mA}$  e  $I_{bsat} = I_{csat} / \beta = 10 \text{ mA} / 50 = 200 \mu\text{A}$ .

### Exercício:

Procurar manuais (atualizados) na biblioteca e encontrar componentes com as principais funções lógicas. Duas entradas são suficientes. Mostre como o componente seria usado para efetuar as funções básicas: NOT, AND, NAND, OR, NOR e XOR. Inclua a referência (manual(ais) de onde a informação foi retirada).



## 9. Funções booleanas e circuitos digitais

Como escrito no título deste ítem vamos falar um pouco sobre circuitos digitais. Os circuitos digitais podem ser classificados em dois grandes grupos. Os circuitos **combinacionais (ou combinatórios)** e os circuitos **sequenciais**. Os circuitos combinatórios são circuitos digitais nos quais a saída depende apenas das entradas atuais e nos circuitos sequenciais a saída depende das entradas e do estado interno do circuito, ou seja, são circuitos com memória.

A representação esquemática dos circuitos combinatórios é a seguinte:



Os circuitos combinatórios são circuitos desprovidos de memória. Por outro lado, a outra classe de circuitos digitais, os circuitos sequenciais, é formada por circuitos com memória. Neste caso, não basta saber o estado das entradas atuais para se conhecer as saídas. É necessário que se conheça o estado interno do circuito. Nestes circuitos a sequência de apresentação das entradas é fundamental na determinação da saída atual. Além dos blocos lógicos já estudados, outros circuitos básicos com memória são utilizados nos circuitos sequenciais. A descrição e definição formal dos circuitos sequenciais será vista mais adiante quando estivermos estudando a sua síntese.

De modo geral chamaremos de circuitos digitais aqueles formados por combinações dos blocos lógicos básicos descritos anteriormente. A função que descreve matematicamente o circuito é uma função booleana. Se  $f(x,y,z) = x.y + z + x'$  é uma função tal que  $x, y$  e  $z$  assumem apenas dois valores (por exemplo 0 e 1) e consequentemente são **variáveis booleanas**, então  $f(x,y,z)$  é uma função booleana.

Chega de papo! Como fazer para avaliar uma função destas? Simples! Se cada variável assume apenas dois valores, então o número de combinações de  **$n$  variáveis** será  **$2^n$** . Deste modo podemos construir uma tabela na qual as entradas são as variáveis e a saída, o valor da função. Cada variável assumirá apenas os valores 1 e 0 e assim também ocorrerá com a saída (valor de  $f$ ) para cada combinação das entradas. Veja por exemplo a tabela abaixo:

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(Veja com o professor um truque para poder construir tabelas como estas com qualquer número de variáveis, sem ter que realmente pensar nas combinações...ou descubra voce mesmo!!!)

Esta tabela é denominada de **Tabela Verdade** (como se tivéssemos atribuído valor *Verdade* para a saída 1 em oposição a *Falso* para saída 0. O circuito iria **ligar** algo quando a saída fosse 1 ("alta"), etc... Veja mais tarde que a **lógica** poderia ser **"invertida"**!

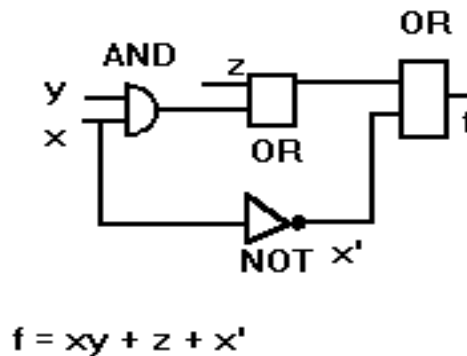
Continuando agora com a lógica não-invertida. E o oposto? Dada a tabela com as saídas desejadas como determinar a função booleana correspondente? Daí vem mais umas coisinhas para incrementar seu cabedal de conhecimento das palavras: **mintermo** e **soma canônica de produtos**. O produto lógico (AND, •) de todas as variáveis da função, complementadas se o seu valor na tabela é 0, é chamado de um mintermo. Pronto! ( o mintermo assume o valor 1 para apenas uma combinação das variáveis). A soma lógica (OR, +) de todos os mintermos da tabela é uma soma canônica e esta soma descreve a função -- Essa é boa!!! Vejamos no caso da função descrita acima qual seria a soma canônica a partir da tabela. Siga em frente e faça alguma coisa.....Por exemplo, olhe para a tabela e comece a escrever os mintermos e vá somando, somando,.....Seria:

$$f = x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz' + xyz$$

Muito interessante. O que esta função tem a ver com a função original,  $f = xy + z + x'$  ? Você quer mesmo saber? Então faça o exercício de avaliar a função canônica e colocar o resultado ao lado da saída da função original, na tabela. Viu? Deu a mesma coisa, e agora? Espere a próxima aula ..... e saberá. Ou melhor ainda, deixe a preguiça de lado e leia a bibliografia até descobrir.

Vamos agora verificar como seria o circuito digital correspondente a cada uma das funções, construído-o com blocos lógicos de apenas duas entradas. Quantas entradas tem os CIs que vocês viram nos manuais? Muito bem, façam como exercício

o circuito com a função mais complexa e o professor, como sempre, fará o circuito para a função mais simples, este circuito está apresentado a seguir:



Uma função booleana pode ser representada por um produto de somas. Basta considerar os termos para os quais a função assume o valor 0. Vamos direto a tabela descrita anteriormente. A combinação  $x = 1$ ,  $y = 0$  e  $z = 0$  assume valor 0. Deste modo o termo  $x + y' + z'$  aparece na função, que neste caso seria:  $f = (x' + y + z) \cdot (\text{alguma outra saída nula se houvesse}) \dots$ . A soma  $x' + y + z$  é chamada de **maxtermo** e o produto de todos os maxtermos será o **produto canônico de somas** (viu? mais "termos"). Compare a função obtida a partir dos *max*- com aquela obtida a partir dos *mintermos*.

As combinações apresentadas na tabela verdade poderiam ser representadas por um número decimal (por exemplo o número da linha). Na realidade se atribuirmos valores a cada posição ocupada pelos 0s e 1s na linha podemos escrever um **número binário**, usando uma notação posicional, como fazemos para escrever os números decimais, ou seja, o número 34 corresponde a  $3 \times 10^1 + 4 \times 10^0 = 30 + 4 = 34$  e 10 é a base numérica, certo? Por analogia, o número binário 101 seria  $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5$  em decimal. Deste modo podemos escrever cada combinação das entradas da tabela verdade na sua notação decimal e com alguma conveniência representar a função por:

$$f(x,y,z) = S(0,1,2,3,5,6,7).$$

Acho que neste ponto alguém já deve estar pensando: "o que vem primeiro o ovo ou a galinha?" Certo, o que você acha? A gente constrói uma tabela verdade e daí projeta um circuito ou a gente escreve tabelas para representar os circuitos? Claro, é exatamente o que você estava pensando... *a tabela representa aquilo que a gente quer* e conseqüentemente é uma das maneiras de representar a função ou o circuito.

Suponha que você tenha que fazer um circuito com as seguintes especificações: a) um painel com três lâmpadas (uma vermelha, outra amarela e outra verde) dispostas verticalmente; b) as lâmpadas deverão ser acesas de acordo com o

acionamento de 3 chaves por uma determinada pessoa (ou animal). Se a chave 1 for ligada a lâmpada **verde** será acesa, caso a chave 2 seja a escolhida a lâmpada **vermelha** será ativada e finalmente se a opção for pela chave 3 a lâmpada **amarela** é que será ligada. Se, por outro lado as três chaves forem ligadas ao mesmo tempo, todas as lâmpadas deverão acender. Como poderíamos expressar este problema na forma de uma tabela? Assim:

C1	C2	C3	G	Y	R
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	1	1	1

E agora? São 3 saídas ! Problema novo! É pra isto que existe professor... e livros! Vamos então escrever os circuitos para cada saída e pronto.

$$G = C1C2'C3' + C1C2C3$$

$$Y = C1'C2'C3 + C1C2C3$$

$$R = C1'C2C3' + C1C2C3$$

Pronto aí estão as somas canônicas para cada saída. Como exercício façam os circuitos equivalentes e testem o que fizeram (com lápis e papel mesmo...calma!). Veja um modo organizado de desenhar o circuito com o professor.

.....Trabalho.....

Vamos agora pensar em outra situação, na qual, para certas combinações das entradas, a saída pudesse ser indiferente. Como assim?... assim como na tabela abaixo:

x	y	z	s
0 0	0	0	0
1 0	0	1	1
2 0	1	0	0
3 0	1	1	1
4 1	0	0	x

---

5	1	0	1	0
6	1	1	0	x
7	1	1	1	0

A função, sem levar em conta os estados para os quais a saída é indiferente seria:

$$f = x'y'z + x'yz$$

Quando esta situação acontece dizemos que se trata de uma combinação ou estado indiferente (*Don't care state*). Os *don't care* são marcados com um **x** que significa troque por 1 ou 0, ***I don't care!***, frase muito comum nos USA:

A mãe

- *Darling please, you don't look fine on these pants...*

O filho adolescente

- *I don't care, mom!*

Usando a notação simplificada para representar a função teríamos:

$$f = S(1,3) + D(4,6),$$

onde D representa as saídas indiferentes ou *don't care states* (veremos o que fazer com eles depois...).

### Mapas de Karnaugh

Continuando um pouco com a idéia de representação das funções booleanas, vamos definir o que se chama de **mapa de Karnaugh**. Trata-se de uma forma modificada da tabela verdade. Uma forma conveniente de apresentação (Porque você não inventa outra melhor? Eu sei. É porque você ainda não entende o suficiente sobre o assunto? Certo?). Os mapas de n variáveis irão conter  $2^n$  células. Vejamos como seria representada a tabela a seguir:

	a	b	c	y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0

---

3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Mapa para tres variáveis:

<b>a</b> <b>c</b> \ <b>b</b>	<b>00</b>	<b>01</b>	<b>11</b>	<b>10</b>
	<b>0</b>	<b>2</b>	<b>6</b>	<b>4</b>
<b>1</b>	<b>1</b>	<b>3</b>	<b>7</b>	<b>5</b>

Observe que cada célula pode ser marcada com o decimal correspondente à combinação binária da tabela verdade. No final, o mapa de Karnaugh deve conter em suas células a saída que se deseja representar, ou seja:

<b>a</b> <b>c</b> \ <b>b</b>	<b>00</b>	<b>01</b>	<b>11</b>	<b>10</b>
	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>

Vamos então ver agora como seriam as configurações para 4, 5 e 6 variáveis:

Mapa para 5 variáveis:

A = 0					A = 1					
<div>b c d e</div>	c	00	01	11	10	<div>c b d e</div>	00	01	11	10
	00	0	4	...			16	20	...	
	01									
	11									
	10									

Coloque os valores das combinações das entradas (em decimal) nas diversas células do mapa apresentado acima. **Que relação há entre os valores binários das células adjacentes?** Veja por exemplo, 2 e 6, 16 e 20. Pratique. MUITO !!!

Mapa de Karnaugh para 6 variáveis:

B = 0					B = 1						
A=0	c e f	d	00	01	11	10	00	01	11	10	c e f
	00		0	4	...		16	20	...		00
	01										01
	11										11
	10										10

A=1	00 01 11 10					00 01 11 10				
	00	32		...		48		...		
	01									
	11									
	10									
c e f	d					c e f	d			

A organização das células no mapa de Karnaugh não é fortuita! É muito interessante notar que todas as células vizinhas diferem em apenas uma posição na sua notação binária. Revise o que você já pensou anteriormente e confira. Esta organização é parte da base para a simplificação automática de funções, usando o mapa de Karnaugh. O código usado para identificar as linhas e colunas é

extremamente importante. Graças a este código, células com um lado em comum correspondem a combinações que diferem no valor de uma única variável (você se lembra da definição de **combinação** ou **matching**?). Estas células são chamadas de *adjacentes*. Observe que, no mapa de 3 variáveis as células 0 e 4 são adjacentes. Deste modo a idéia é juntar células com valor 1 nos chamados **subcubos** compostos por  $2^n$  células. Um produto das variáveis que assumem o mesmo valor pode ser gerado, de modo que as variáveis apareçam complementadas se o valor for 0 e não se o seu valor for 1. O valor da variável é aquele que você observa lendo os 1s e 0s dispostos convenientemente no índice das células do mapa. Um conjunto de  $2^n$  células, cada uma adjacente a  $n$  células do conjunto é chamado de subcubo de ordem  $n$ . Dizemos que este subcubo **cobre** estas células. Todo subcubo cuja ocorrência ("1") obriga a função a ter valor 1, ou seja, **Implica** na função dada é um **implicante** da função. Associamos a cada implicante um produto de literais. Uma certa função pode ser expressa pela soma de todos os produtos implicantes correspondentes aos subcubos necessários para cobrir todas as células com valor 1 no mapa.

Exemplo:

		a			
		b			
		c			
		00	01	11	10
0			1		
1			1	1	1

$F[a,b,c] = a'b + ac$

Os subcubos escolhidos foram todos aqueles que cobririam a função completamente, sem intersecção entre eles. Cada subcubo deve ser escolhido como sendo o maior possível (neste caso mais variáveis serão eliminadas). No caso da existencia de *don't care states* eles deverão ser usados convenientemente como 0s ou como 1s, de modo a favorecer a formação dos subcubos. Um implicante que é obtido de um subcubo não contido em nenhum subcubo maior é chamado de **implicante primo** (*prime implicant*).

### Custos e risco de falhas

Como sempre, na prática a teoria é outra. No entanto sem saber a teoria você pode cair no lema do velho ditado: "se você não sabe o que está procurando, quando acha não sabe que achou".

Supondo que saibamos representar e simplificar circuitos digitais, a idéia ao se definir um circuito final é a de procurar reduzir custos, desde que a qualidade seja preservada.



Vamos assumir como custo, uma função que nada mais é que a soma dos terminais de entrada dos blocos lógicos. Tomemos por exemplo a função:

$$f(x,y) = yx' + y$$

Neste caso o custo seria:

$$\text{Custo} = 2 + 1 + 2 = 5$$

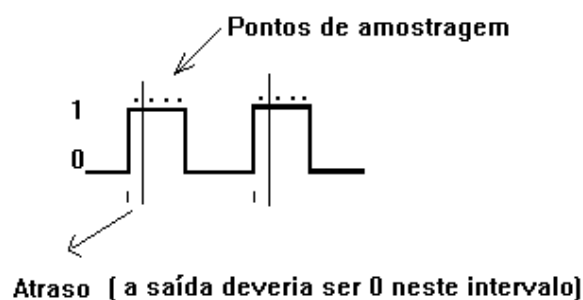
Simplificando a função (e vocês já sabem fazer isso por meio de manipulação algébrica), teríamos:

$$f(x,y) = y$$

e neste caso o custo torna-se:

$$\text{Custo} = 1 \quad (\text{Bem menor...certo?})$$

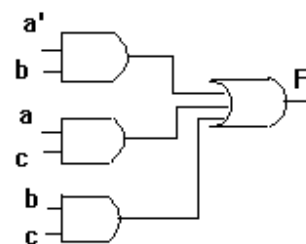
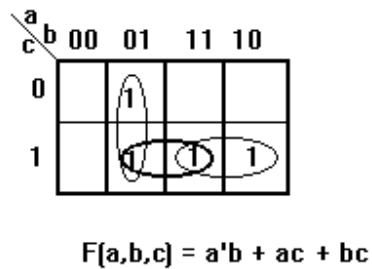
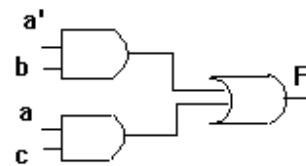
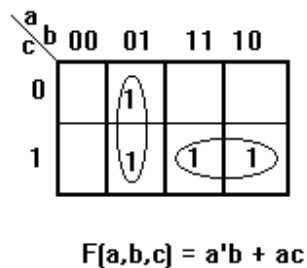
Infelizmente os circuitos digitais não respondem instantaneamente aos sinais de entrada! Isto pode ocasionar **falhas**. Por causa disso deve-se ter em mente os conceitos de TPLH (*time for propagation of the low to high transition*) e TPHL (*time of propagation of the high to low transition*). TPLH é o tempo necessário para a mudança do nível baixo para o nível alto em um determinado componente lógico. Deste modo as vezes temos que sacrificar o baixo custo para garantir o bom funcionamento do circuito. Já ouviu dizer que o barato sai caro? É isso.



Suponha que você esteja projetando um circuito que deva transmitir milhares de dados e que em um determinado ponto a saída do circuito é amostrada, de modo sincronizado, como indicado pelos pontos na parte "1" do sinal (Figura acima). Que desastre! Por causa do atraso introduzido pelo componente, "1" será transmitido, ao invés de "0", para o primeiro ponto. Se aquilo fosse uma "palavra" binária,  $(0111)_2$  seria transformado em  $(1111)_2$ . Se isto fosse seu débito, ao invés de R\$ 7.000,00 (o milhar apareceu apenas para lhe assustar!) você estaria devendo R\$ 15.000,00 (ligeiramente diferente certo?).

### Correção do risco de falhas

Quando a falha é ocasionada por apenas um par de combinações de entrada adjacentes que produzem a mesma saída e não houver na expressão mínima uma célula que contenha ambas as combinações das entradas. O circuito gerado pelo mapa a seguir é um destes casos e pode ser corrigido conforme apresentado no mapa seguinte.



### Exercício:

Resolver os exercícios da série 1 (Que série 1 ? )

## 10. Simplificação de funções booleanas

Agora vamos estudar como fazer para simplificar funções booleanas de modo sistemático. Antes porém tentaremos fazer algum trabalho de simplificação de modo que possamos tomar contato com o mapa de Karnaugh, usado para esta função. Vamos então propor a construção de um circuito, construir a sua tabela verdade, expressão booleana (soma canônica) e mapa de Karnaugh. Em seguida vamos simplificar a função no mapa de Karnaugh e obter a função simplificada. Com isto teremos condições de desenhar o diagrama do circuito com os blocos lógicos conhecidos. Vamos iniciar com algumas definições não formais:

**Prime implicant** (implicante primário ou implicante *primo* )

Dado um circuito combinatório descrito pelo mapa de Karnaugh, agrupamos as células com 1s nos maiores grupos retangulares de tamanho  $2^n$ . A cada grupo destes associamos um produto das variáveis (complementadas ou não) que pode ser obtido da soma canônica de todos 1s do grupo, simplificados pelo teorema que diz que  **$AB + AB' = A$  (combinação)**. Este produto é chamado de *Prime Implicant* da função.

### Soma mínima

A soma mínima corresponde a soma do menor número de *prime implicants* que cobre completamente a função.

Marca-se o conjunto de todos os *prime implicants* e depois escolhe-se um conjunto mínimo. Primeiro identifica-se os chamados essenciais, que são aqueles que sozinhos (sem interseções) cobrem um conjunto de células (subcubo) com 1s. Depois, os não-essenciais serão escolhidos levando-se em conta um critério de custos.

### Exemplo prático:

Queremos sintetizar um circuito com quatro entradas que tenha como saída o valor 1 apenas quando duas entradas forem iguais a 1. A tabela verdade seria:

a	b	c	d	s
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1

---

1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	0

O mapa de Karnaugh para quatro variáveis seria:

$\begin{array}{c} a \\ \swarrow \searrow \\ c \quad b \end{array}$					
		00	01	11	10
$\begin{array}{c} d \\ \swarrow \searrow \\ c \quad b \end{array}$	00	0	0	①	0
	01	0	①	0	①
	11	①	0	0	0
	10	0	①	0	①

Neste caso a vida não sorriu!!! Temos que escolher subcubos de apenas 1 elemento, o que torna a função mínima idêntica à soma canônica. Ficaríamos com:

$$F(a,b,c,d) = abc'd' + a'bc'd + ab'c'd + a'b'cd + a'bcd' + ab'cd'$$

Vamos contudo sintetizar um circuito um pouco diferente. Neste caso vamos assumir que as entradas 1000 e 1101 (abcd) sejam *don't care states*. Neste caso teríamos:

$\begin{array}{c} a \\ \swarrow \searrow \\ c \quad b \end{array}$		00	01	11	10
		00	01	11	10
$\begin{array}{c} d \\ \swarrow \searrow \\ c \quad b \end{array}$	00	0	0	①	⊗
	01	0	①	⊗	①
	11	①	0	0	0
	10	0	①	0	⊗

Os *prime implicants* seriam:

essenciais:  $a'b'cd$ ,  $a'bcd'$ ,  $ac'$ ,  $bc'd$ ,  $ab'$

O produto  $a'bc'd$  não é um prime implicant porque está contido em um subcubo maior!

$$S_m = a'b'cd + a'bcd' + ac' + bc'd + ab'$$

### Exercício:

proponha vários outros mapas e descreva os conjuntos de prime-implicants

## Método de Quine-McCluskey

### Etapas 1. Geração dos implicants primos

Vamos trabalhar com os produtos fundamentais de modo que  $w'xy'z$  seja representado por 0101. O Passo 1 será listar as linhas para as quais a função assume saída igual a 1. Cada linha contém (coluna 2 da esquerda para a direita) o decimal equivalente levando-se em conta a função que queremos minimizar:

$$F(v,w,x,y,z) = \Sigma(0,2,4,6,7,8,10,11,12,13,14,16,18,19,29,30)$$

		V	W	X	Y	Z	marca
0	(0)	0	0	0	0	0	✓
1	(2)	0	0	0	1	0	✓
1	(4)	0	0	1	0	0	✓
1	(8)	0	1	0	0	0	✓
1	(16)	1	0	0	0	0	✓
2	(6)	0	0	1	1	0	✓
2	(10)	0	1	0	1	0	✓
2	(12)	0	1	1	0	0	✓
2	(18)	1	0	0	1	0	✓
3	(7)	0	0	1	1	1	✓
3	(11)	0	1	0	1	1	✓
3	(13)	0	1	1	0	1	✓
3	(14)	0	1	1	1	0	✓
3	(19)	1	0	0	1	1	✓
4	(29)	1	1	1	0	1	✓
4	(30)	1	1	1	1	0	✓
							✓

O Passo 2 consiste em comparar cada par de caracteres binários dos produtos fundamentais para verificar se diferem em apenas uma coordenada (posição binária) e combiná-los pelo teorema :

$$XY + XY' = X$$

As comparações são feitas de modo guiado pela primeira coluna à esquerda. Todos de índice 0 com todos de índice 1, todos de índice 1 com os de índice 2 e assim por diante. Uma marca (✓) é colocada na coluna mais à direita para indicar que o elemento da referida linha foi combinado com algum outro.

Com isto gera-se uma nova tabela com produtos fundamentais formados pelos caracteres binários que são iguais nos pares comparados e um traço no local correspondente ao caractere que foi diferente. Por exemplo ao comparar :

01000      v' w' x' y' z'  
00000      v' w' x' y' z'

Gera-se

0-000      v' x' y' z'

Assim, a tabela a seguir contém o resultado deste segundo passo.

	V	W	X	Y	Z	marca
(0,2)	0	0	0	-	0	✓
(0,4)	0	0	-	0	0	✓
(0,8)	0	-	0	0	0	✓
(0,16)	-	0	0	0	0	✓
(2,6)	0	0	-	1	0	✓
(2,10)	0	-	0	1	0	✓
(2,18)	-	0	0	1	0	✓
(4,6)	0	0	1	-	0	✓
(4,12)	0	-	1	0	0	✓
(8,10)	0	1	0	-	0	✓
(8,12)	0	1	-	0	0	✓
(16,18)	1	0	0	-	0	✓
(6,7)	0	0	1	1	-	
(6,14)	0	-	1	1	0	✓
(10,11)	0	1	0	1	-	
(10,14)	0	1	-	1	0	✓

(12,13)	0	1	1	0	-	
(12,14)	0	1	1	-	0	✓
(18,19)	1	0	0	1	-	
(13,29)	-	1	1	0	1	
(14,30)	-	1	1	1	0	

Agora, novas combinações são investigadas (faça em ordem crescente e não inclua na tabela as repetições) e aquelas possíveis são marcadas. O produto que pode ser combinado com outro **não** é implicante primo e portanto deve ser marcado.

No Passo 3 os caracteres da tabela acima são comparados novamente, do mesmo modo para buscar possíveis combinações. Deste modo uma nova tabela será montada como a seguir:

	V	W	X	Y	Z	marca
(0,2; 4,6)	0	0	-	-	0	✓
(0,2; 8,10)	0	-	0	-	0	✓
(0,2; 16,18)	-	0	0	-	0	
(0,4; 8,12)	0	-	-	0	0	✓
(2,6; 10,14)	0	-	-	1	0	✓
(4,6; 12,14)	0	-	1	-	0	✓
(8,10; 12,14)	0	1	-	-	0	✓

Repare que quase todas as linhas estão marcadas e apenas uma combinação está apresentada na última tabela. Isto porque após combinar (0,2,4,6) com (8,10,12,14) as outras combinações geram exatamente 0 - - - 0 que já existia. Portanto não serão colocados na nova tabela. Agora, no Passo 4 tudo se repete com a tabela anterior e o resultado é o que se apresenta abaixo:

	V	W	X	Y	Z	
(0,2,4,6,8,10,12,14)	0	-	-	-	0	

Os produtos não marcados são os implicantes primos da função:

$$F = (0,2,4,6,8,10,12,14) + (0,2,16,18) + (14,30) + (13,29) + (18,19) + (12,13) + (10,11) + (6,7)$$

No entanto esta ainda não é a soma mínima. Passemos para a segunda etapa do método que corresponde a cobertura dos mintermos.

### Etapa 2. Cobertura de mintermos

Nesta etapa serão procurados os mintermos que são cobertos pelos implicantes primos da função. Um *prime implicant* que cubra apenas um mintermo é chamado de implicante primo essencial. Este deverá fazer, necessariamente, parte da função. Para isto montamos o seguinte tipo de tabela:

#### Prime implicant

#### Mintermo

		0	2	4	6	7	8	10	11	12	13	14	16	18	19	29	30	
(0,2,4,6,8,10,12,14)	0- - -0	x	x	x	x		x	x		x		x						←
(0,2,16,18)	-00-0	x	x										x	x				←
(14,30)	-1110											x					x	←
(13,29)	-1101										x					x		←
(18,19)	1001-													x	x			←
(12,13)	0110-									x	x							•
(10,11)	0101-							x	x									←
(6,7)	0011-				x	x												←

Agora olhamos na tabela e identificamos os mintermos que são cobertos por apenas 1 *prime implicant*, por exemplo, o mintermo 7 que é coberto pelo (6,7) apenas. Lembre-se que a tabela foi montada simplesmente listando PI x (produtos fundamentais) Mintermos e marcando quais mintermos correspondem aos PIs. Por exemplo, colocamos x em 18 e 19 para o PI (18,19) e assim por diante. Todos os PIs que tem mintermos cobertos apenas por eles devem ser marcados na tabela com uma seta na linha correspondente. Estes já fazem parte da função mínima. São essenciais. Marquei com um • a linha que não contém um PI essencial.

Veja que a linha 2, por exemplo, foi marcada por que o mintermo 16 é unicamente coberto pelo PI (0,2,16,18) e a linha 4 foi marcada por que o mintermo 30 é unicamente coberto pelo PI (13,29) e assim por diante. As setas estão indicando quem será retirado da tabela porque já faz parte da função. Neste caso apenas a linha 6 não satisfaz este requisito (•).

A linha (10,11) e coluna 11 caracterizam a linha como linha essencial e a coluna como distinta (importante). Nesta função todas as colunas (mintermos) estão em alguma linha essencial e o único PI que não é essencial é o (12,13).

Desta forma a função final é:

$$F_{\min} = v'z' + w'x'z' + wxyz' + wxy'z + vw'x'y + v'wx'y + v'w'xy$$

Lembre-se que todas as colunas estavam marcadas com x em linha essencial.



## 11. Circuitos sequenciais

Os circuitos lógicos estudados até o momento são desprovidos de memória. Neste caso, a saída é função unicamente do estado atual das entradas. Nos circuitos lógicos sequenciais, elementos de memória podem informar o estado interno do circuito. Assim a saída será função das entradas e do estado anterior dos elementos de memória. Em outras palavras o circuito lembra estados anteriores.

A Figura 8.1 ilustra um esquema geral de um circuito sequencial, na qual, uma rede combinacional é realimentada por elementos de memória. As saídas  $z_i$ , neste caso, serão função das entradas  $x_j$  e do estado dos elementos 1 a p dados por  $y_1 \dots y_p$ .

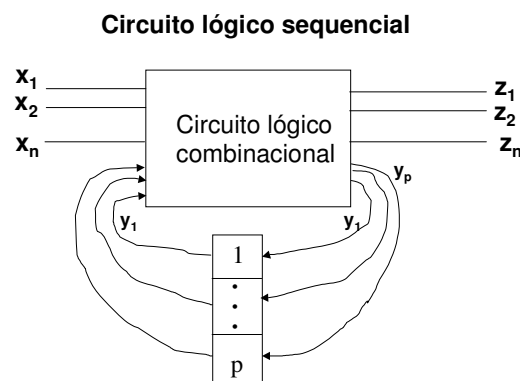


Figura 8.1 Esquema geral de um circuito lógico sequencial. A rede ou circuito combinacional é associada a elementos de memória (p elementos). O conjunto dos elementos de memória compõe um registrador com definido no capítulo x.

Os elementos básicos de memória são células binárias como definido no capítulo (x). Vamos agora definir outras células de memória que do ponto de vista prático são fundamentais para a construção de circuitos digitais, em particular dos registradores, elementos básicos das máquinas digitais. Estas células são os diversos tipos de Flip-Flops.

### Flip-Flop - RS

A sigla RS vem do inglês "Reset" que significa inicializar com zeros, "apagar" e "Set" que significa "ligar", carregar, colocar "1". Apresentamos a seguir duas maneiras de sintetizar um flip-flop - RS também chamado de "latch".

No primeiro caso o FF-RS é sintetizado com portas NOR como apresentado na Figura 8.2.

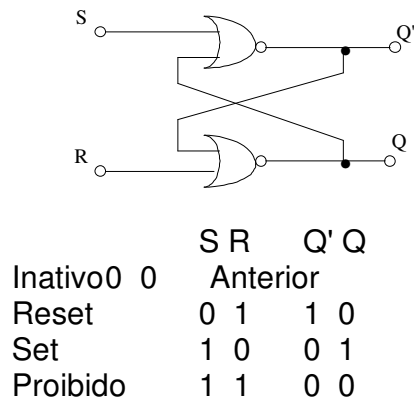


Figura 8.2 Flip-Flop RS sintetizado com portas NOR. Note que Q' e Q estão trocados com relação ao latch NAND. Ativo alto.

A Figura 8.3 ilustra o Flip-Flop RS sintetizado com portas NAND. Note que as entradas estão complementadas.

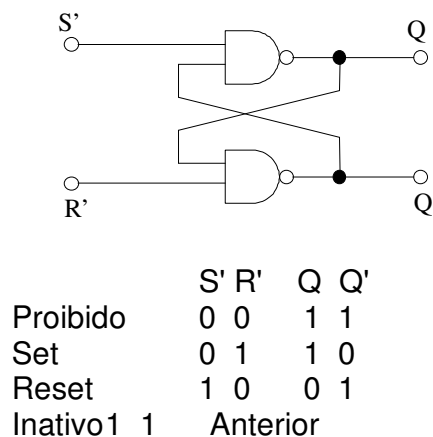


Figura 8.3 Flip-Flop RS sintetizado com portas NAND. S'e R' são notações para indicar que o latch é ativo baixo

### Flip-Flop RS com ativador ou clock

Neste caso se  $CK = 0$  haverá bloqueio, ou seja, independente de R e S o circuito mantem o estado. Se  $CK=0$  o que ocorre é que S1 e R1 serão ambas iguais a

1, forçando a condição de inatividade do FF e portanto manutenção do seu estado. Para  $CK = 1$  o FF se comporta como um FF-RS comum.

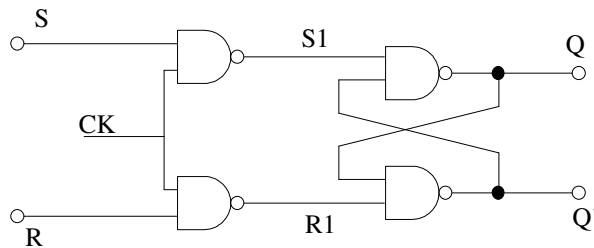


Figura 8.4 Flip-Flop RS com controle. O controle é feito pelo sinal de entrada CK. Note que nesta configuração o aparecimento do circuito direcionador (onde entra o CK) torna o FF ativo alto.

Vejamos o que ocorre quando, no FF da Figura 8.4 a condição de  $S = R = 1$  e CK vai de 1 para 0 se estabelece.

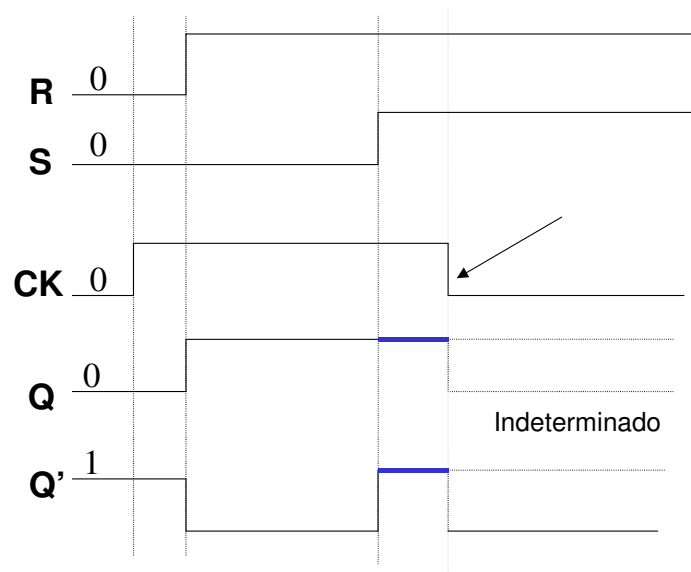


Figura 8.5 Temporização e sinais correspondentes no FF-RS com controle. A seta indica um instante em que CK vai de 1 para 0 com S e R iguais a 1. Neste caso Q fica imprevisível.

### Flip-Flop tipo D

Procurando contornar este problema foi desenvolvido o flip-flop tipo D. Neste caso, o controle (CK) permite ou não que um certo dado (D) seja "copiado". Este FF

tem o inconveniente de ter as entradas para o *latch*, alteradas sempre que  $CK = 1$ , se o dispositivo for sensível a nível.

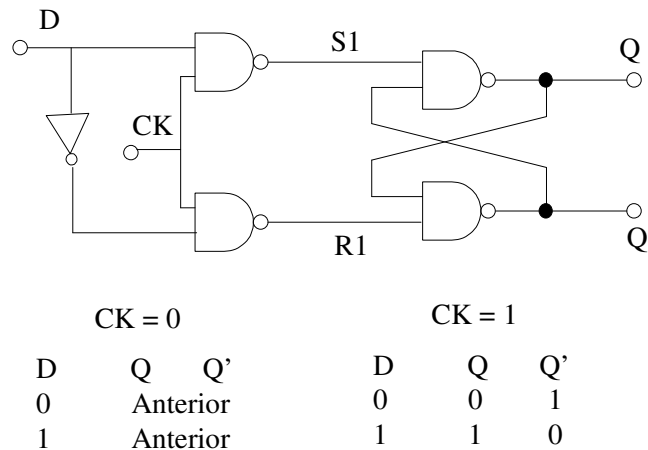
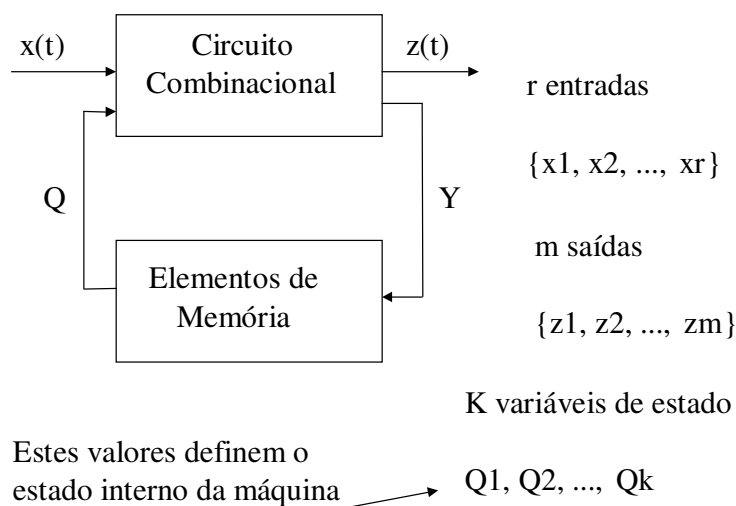


Figura 8.6. Flip-Flop tipo D, sensível a nível. Neste caso não há estado indeterminado e a saída Q "copia" o dado enquanto  $CK = 1$ .

Vamos agora analisar o diagrama temporal relativo ao FF-D sensível a nível como apresentado na Figura 8.6. Observe que o valor de Q muda, copiando D, sempre que CK for igual a 1.

## Máquinas Síncronas

Máquina seqüencial, síncrona ou máquina de estados finitos é o modelo abstrato do circuito seqüencial real. Seu comportamento é descrito como uma seqüência de eventos que ocorrem em intervalos discretos, designados  $t_1, t_2, \dots, t_n$ . Suponha que uma máquina  $M$  receba alguns sinais de entrada e responda produzindo alguns sinais de saída. Se no instante  $t$  for aplicado um sinal  $x(t)$  a  $M$ , sua resposta  $z(t)$  depende de  $x(t)$ , bem como das entradas de  $M$  nos instantes anteriores. Como  $M$  pode ter uma variedade infinita de estados anteriores ou passados, seria necessário uma capacidade de armazenamento infinita. Vamos nos concentrar nas máquinas cujo passado pode ser resumido em um conjunto finito de variáveis (Máquinas determinísticas ou markovianas).



### Tabelas e diagramas de estado

Tabela  $p = 2^r$  colunas (1 para cada ocorrência do vetor de entrada)  
 $n = 2^k$  linhas (1 para cada estado)

Diagrama de estados é um grafo orientado, no qual cada estado da máquina corresponde a um dos nós. De cada nó emanam  $p$  arcos orientados, correspondentes às transições de estado causadas pela ocorrência da entrada. O arco é rotulado com a entrada que determina aquela transição e com a saída gerada nas máquinas determinísticas ou markovianas.

$$A) Q(t+1) = f [Q(t), x(t)]$$

Onde  $f$  é a função de transição de estados. O valor da saída  $z(t)$  é função do estado presente  $Q(t)$  e muitas vezes, das entradas presentes  $x(t)$ .

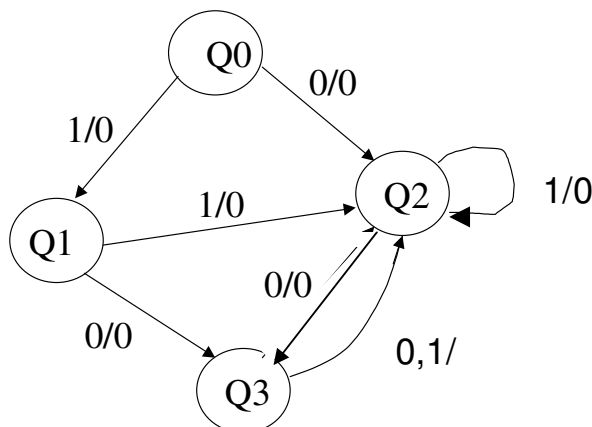
$$B) z(t) = g[Q(t)]$$

$$C) z(t) = g[Q(t), x(t)]$$

onde  $g$  é a função de saída.

Máquinas que seguem A e B são chamadas de Máquinas de Moore e as que seguem A e C denominam-se Máquinas de Mealy.

Mealy

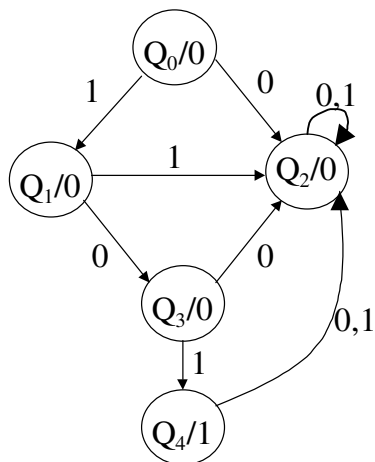


$$Q(t) = f [Q(t), x(t)]$$

$$z(t) = g [Q(t), x(t)]$$

(Q(t) , z(t))		
Q(t)	X = 1	X = 0
Q0	Q1,0	Q2,0
Q1	Q2,0	Q3,0
Q2	Q2,0	Q3,0
Q3	Q2,1	Q2,0

**Moore**



$$Q(t+1) = f [Q(t), x(t)]$$

$$z(t) = g[Q(t)]$$

Q(t+1)			
Q(t)	X = 1	X = 0	Z(t)
Q0	Q1	Q2	0
Q1	Q2	Q3	0
Q2	Q2	Q2	0
Q3	Q4	Q2	0
Q4	Q2	Q2	1

## 12. Códigos

### 12.1 Decimal codificado em binário (BCD, Binary Coded Decimal)

Neste sistema, conjuntos de dígitos binários são tratados como pseudo-decimais. Um grande número de máquinas, no início, usou BCD e talvez até hoje este sistema persista em grandes máquinas comerciais. Basicamente, o que se faz é associar um único número de **m** dígitos, **m ≥ 4**, a cada dígito decimal. O mais usado é o mais óbvio, chamado código 8421. Neste caso  $0000 \equiv 0$ ;  $0001 \equiv 1$  ...  $1001 \equiv 9$ . Assim o decimal 243 seria representado em BCD como 0010 0100 0011. Em BCD cada decimal precisa ser representado por, pelo menos, 4 dígitos binários. Neste caso há  $16! / 6!$  maneiras diferentes de codificar os 10 dígitos decimais com 4 binários. Destes,  $3 \times 10^9$  códigos apenas alguns tem sido realmente usados e com propósitos específicos. Normalmente há duas classes de BCDs: códigos pesados e não-pesados. O 8421 é um código pesado. Cada posição vale o produto do dígito com o correspondente valor da base (no caso 2) elevada ao índice da posição. Cada dígito decimal é equivalente à soma dos pesos dos 1's que representam a sequência binária. Por exemplo: 1011 pode ser visto assim:

$$\begin{array}{cccc} \mathbf{8} & \mathbf{4} & \mathbf{2} & \mathbf{1} \\ 1 & 0 & 1 & 1 \\ 8 + 0 + 2 + 1 = (11)_{10} \end{array}$$

De modo geral, em um código BCD pesado cada dígito decimal corresponde a uma sequência  $c_1 c_2 c_3 c_4$  com valor

$$\sum_{i=1}^4 c_i w_i$$

onde  $w_i$  é o peso e  $c_i$  é o dígito binário.



Se forem usados apenas números positivos para os pesos há apenas 17 BCD pesados com 4 dígitos ( $w_i > 0$ ). Por exemplo:

	w1	w2	w3	w4	
	<b>2</b>	<b>4</b>	<b>2</b>	<b>1</b>	
0	0	0	0	0	
1	0	0	0	1	ou
2	1	0	0	0	0 0 1 0
3	0	0	1	1	1 0 0 1
4	0	1	0	0	1 0 1 0
5	1	0	1	1	0 1 0 1
6	1	1	0	0	0 1 1 0
7	0	1	1	1	1 1 0 1
8	1	1	1	0	
9	1	1	1	1	

	w1	w2	w3	w4	
	<b>7</b>	<b>4</b>	<b>2</b>	<b>1</b>	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	ou
7	1	0	0	0	0 1 1 1
8	1	0	0	1	
9	1	0	1	0	

Observe que não há unicidade nestes códigos. Apenas o código 8421 apresenta unicidade na representação BCD.

É também possível trabalhar com pesos negativos. Deste modo o número de códigos cresce para 88 com 71 pesos negativos e positivos. Um destes códigos é o que está apresentado abaixo:

w1	w2	w3	w4
<b>8</b>	<b>6</b>	<b>-4</b>	<b>1</b>

---

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

Dos 71 códigos deste tipo, 21 apresentam unicidade na representação dos dígitos decimais, como o apresentado acima.

Muitos outros sistemas BCD tem sido usados e sugeridos. Um código biquinário no qual cada dígito decimal é representado por dois grupos de códigos binários, um na base dois e outro na base 5 foi usado há muito tempo em máquinas dos laboratórios da *Bell Telephone*. Um código 5 reduzido de 2 também foi usado com propósitos de comunicação. Este código é formado por grupos de códigos de 5 dígitos binários, todos contendo apenas dois 1's para cada dígito decimal. Assim, se um código é detectado contendo mais ou menos 1's é claro que um erro ocorreu (na transmissão!). Por causa disso este código é chamado de código detetor de erro. Há muitos outros códigos detetores e corretores de erro. O mais popular é sem dúvida o código de teste de paridade. Neste caso, dígitos adicionais são juntados a cada grupo de dígitos binários de modo que o número de 1's se torne sempre par ( ou eventualmente ímpar). Se o número de 1's não for par algum erro ocorreu (por exemplo na transmissão do dado). [Vamos falar mais sobre códigos, mais tarde.](#)

## 13. Alguns exemplos de circuitos digitais

### 13.1 Decodificador

Este circuito pode ser generalizado como um circuito digital com  $n$  entradas e  $2^n$  saídas (Tabela 13.1 e Figura 13.1, decodificador de 2 entradas) das quais apenas uma é ativada para cada combinação das entradas. Nós dizemos que o circuito decodifica (“descobre qual é o código”) a entrada. Imagine, por exemplo, que cada saída de um circuito de 3 entradas esteja ligada ao sistema de acionamento de 8 “caixas-fortes” se a combinação correta (que pode por exemplo ser uma senha) for aplicada na entrada a caixa correspondente é aberta.

Tabela 13.1 - Tabela Verdade  
Do Decodificador de 2 entradas

$E_0$	$E_1$	$S_0$	$S_1$	$S_2$	$S_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Figura 13.1 Decodificador de 2 entradas. A- esquema e B- circuito

### 13.2 Circuitos de Paridade

Para entender os circuitos de paridade é importante antes de tudo demonstrar que  $x \oplus x \oplus x \dots \oplus x$  é igual a zero para um número par de  $x$  e igual a  $x$  para um número ímpar de  $x$ .

a)  $x \oplus x = 0$

$0 \oplus x = x$  portanto  $x \oplus x \oplus x = x$  (número ímpar de  $x$ )

b)  $x \oplus x \oplus x = x$

$x \oplus x = 0$  portanto  $x \oplus x \oplus x \oplus x = 0$  (número par de  $x$ )

Existem dois tipos clássicos de circuitos de paridade: o gerador de paridade (Figura 13.2) e o detector ou verificador de paridade (Figura 13.3). Os circuitos de paridade são usados no processo de detecção de erros de transmissão de dados binários.

Figura 13.2 Gerador de paridade

Figura 13.3 Detector de paridade

### 14.3 Circuitos de prioridade

São circuitos de  $n$  entradas e  $n$  saídas. Se uma única entrada é 1 a saída correspondente é 0. Se mais que uma entrada for 1 então a saída de maior prioridade será 0 (Figura 13.4).

Figura 13.4 Circuito gerador de sinal de prioridade

#### **14.4 Portas de seleção**

Nesta classe de circuitos estão os chamados multiplexadores. Estes circuitos selecionam um conjunto de bits da entrada para ser transferido para a saída (Figura 13.5) sob ativação de sinais de controle.

Figura 13.5 Portas de seleção

#### **14.5 Portas de distribuição**

#### **14.6 Conversor de código**

#### **14.7 Registrador de 3 bits**

#### **14.8 Deslocador à direita com giro**

#### **14.9 Somador de 4 bits com propagação de vai um**

#### **14.10 Máquina digital elementar**

#### **14.11 Deslocador à direita sem giro**

**14.12 Contador módulo 2 (divisor por 2)**

**14.13 Contador modulo 4**

**14.14 Máquina de estados finitos**

**Máquina de vender chiclete (sem trôco)**

## 14. Bibliografia

1. Bonatti, I. & Madureira, M.  
Introdução à análise e síntese de circuitos lógicos. Editora da Universidade Estadual de Campinas - Unicamp, Campinas, SP, 1990.
2. Fregni, E. & Saraiva, A. M.  
Engenharia do projeto lógico digital: conceitos e prática. Editora Edgar Blucher Ltda., São Paulo, SP, 1995.
3. Bartee, T.; Lebow, I. L. & Reed, I. S.  
Theory and design of digital machines. MIT publication, Massachusetts, USA, 1962.
4. Langdon Jr., G. G. & Fregni, E.  
Projeto de computadores digitais. Editora Edgar Blucher Ltda, São Paulo, 1974.
5. Ercegovac, M; Lang, T & Moreno, J H.  
Introdução aos sistemas digitais. Editora Bookman, Porto Alegre, 2000.
6. McCluskey, EJ.  
Logic Design Principles. Prentice Hall, Englewood Cliffs, 1986.
9. Halliday, D.; Resnick, R. & Walker, J.  
Fundamentals of Physics. John Wiley & Sons, Inc., New York, 5<sup>th</sup> ed., 1997
10. Tocci, RJ & Widmer, NS.  
Sistemas Digitais. Princípios e Aplicações. Prentice Hall, São Paulo, 8<sup>th</sup> ed., 2003.
11. Boyer, CB  
A history of mathematics, 2nd ed., John Wiley & Sons, Inc, New York, 1991

