Habilitação técnica em

Eletrônica

4

Eletrônica Digital

Centro Paula Souza

GOVERNO DE SÃO PAULO


CENTRO PAULA SOUZA DO GOVERNO DE SÃO PAULO
CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

CENTRO PAULA SOUZA

GOVERNO DE SÃO PAULO

Eletrônica

Volume 4

FUNDAÇÃO PADRE ANCHIETA

CENTRO PAULA SOUZA

GOVERNO DE SÃO PAULO

Eletrônica

Eletrônica digital

Ronaldo Diago

Valder Moreira Amaral

(autores)

Edson Horta

(coautor)

FUNDAÇÃO PADRE ANCHIETA

2011

Fundação Padre Anchieta

Presidência
João Sayad

Vice-presidência
Ronaldo Bianchi, Fernando Vieira de Mello

DIRETORIA DE PROJETOS EDUCACIONAIS
Direção: Fernando José de Almeida
Gerência: Monica Gardelli Franco, Júlio Moreno
Coordenação Técnica: Maria Luiza Guedes
Equipe de autoria Centro Paula Souza
Coordenação geral: Ivone Marchi Lainetti Ramos
Coordenação da série Eletrônica: Jun Suzuki
Autores: Ronaldo Diago, Valder Moreira Amaral
Coautor: Edson Horta
Revisão técnica: Marcos Vagner Zamboni
Equipe de Edição
Coordenação geral: Carlos Tabosa Seabra, Rogério Eduardo Alves
Coordenação editorial: Luiz Marin

Edição de texto: Roberto Matajs
Secretário editorial: Antonio Mello
Revisão: Conexão Editorial
Direção de arte: Bbox Design
Diagramação: LCT Tecnologia
Ilustrações: Adilson Secco
Pesquisa iconográfica: Completo Iconografia
Capa
Fotografia: Eduardo Pozella, Carlos Piratininga
Tratamento de imagens: Sidnei Testa
Abertura capítulos: © Lize Streeter/Dorling Kindersley/ Getty Images

O Projeto Manual Técnico Centro Paula Souza – Coleção Técnica Interativa, uma iniciativa do Governo do Estado de São Paulo, resulta de um esforço colaborativo que envolve diversas frentes de trabalho coordenadas pelo Centro Paula Souza e é editado pela Fundação Padre Anchieta.
A responsabilidade pelos conteúdos de cada um dos trabalhos/textos inseridos nesse projeto é exclusiva do autor. Respeitam-se assim os diferentes enfoques, pontos de vista e ideologias, bem como o conhecimento técnico de cada colaborador, de forma que o conteúdo exposto pode não refletir as posições do Centro Paula Souza e da Fundação Padre Anchieta.

Dados Internacionais de Catalogação na Publicação (CIP)
(Bibliotecária Silvia Marques CRB 8/7377)

D536

Diago, Ronaldo
Eletrônica: eletrônica digital / Ronaldo Diago, Valder Moreira Amaral (autores); Edson Horta (coautor); Marcos Vagner Zamboni (revisor); Jun Suzuki (coordenador). -- São Paulo: Fundação Padre Anchieta, 2011. (Coleção Técnica Interativa. Série Eletrônica, v. 4)

Manual técnico Centro Paula Souza

ISBN 978-85-8028-048-7

1. Eletrônica digital I. Amaral, Valder Moreira II. Horta, Edson III. Zamboni, Marcos Vagner IV. Suzuki, Jun V. Título

CDD 607

GOVERNO DE SÃO PAULO

GOVERNADOR
Geraldo Alckmin

VICE-GOVERNADOR
Guilherme Afif Domingos

SECRETÁRIO DE DESENVOLVIMENTO ECONÔMICO, CIÊNCIA E TECNOLOGIA
Paulo Alexandre Barbosa

Centro Paula Souza — Governo de São Paulo

Presidente do Conselho Deliberativo
Yolanda Silvestre

Diretora Superintendente
Laura Laganá

Vice-Diretor Superintendente
César Silva

Chefe de Gabinete da Superintendência
Elenice Belmonte R. de Castro

Coordenadora da Pós-Graduação, Extensão e Pesquisa
Helena Gemignani Peterossi

Coordenador do Ensino Superior de Graduação
Angelo Luiz Cortelazzo

Coordenador de Ensino Médio e Técnico
Almério Melquíades de Araújo

Coordenadora de Formação Inicial e Educação Continuada
Clara Maria de Souza Magalhães

Coordenador de Desenvolvimento e Planejamento
João Carlos Paschoal Freitas

Coordenador de Infraestrutura
Rubens Goldman

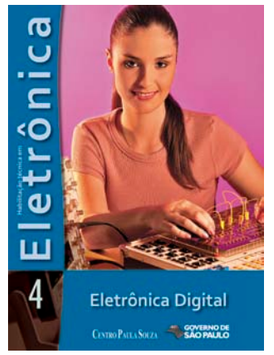
Coordenador de Gestão Administrativa e Financeira
Armando Natal Maurício

Coordenador de Recursos Humanos
Elio Lourenço Bolzani

Assessora de Comunicação
Gleise Santa Clara

Procurador Jurídico Chefe
Benedito Libério Bergamo

Sumário



Capa: Larissa Gabrielle Rizatto, aluna do Centro Paula Souza
Foto: Eduardo Pozella e Carlos Piratininga

Sumário

Capítulo 1

Sistemas numéricos

Os sistemas numéricos são usados para representar a quantidade de determinados elementos. O mais utilizado atualmente pela maioria das pessoas é chamado decimal. Esse nome foi adotado porque a base empregada é composta por dez algarismos, com os quais é possível formar qualquer número por meio da lei da formação.

Existem outros sistemas métricos que são utilizados em áreas técnicas, como eletrônica digital e programação de computadores. Nas próximas seções serão detalhadas as bases mais usadas nessas duas áreas: decimal, hexadecimal, octal e binária. Também veremos os métodos empregados para conversão de números entre essas bases.

1.1 Sistema numérico decimal

Os sistemas de numeração surgiram da necessidade de representar por meio de símbolos as contagens e associações de quantidades que as pessoas realizavam. Os egípcios, os babilônios, os chineses, os maias, os romanos e vários outros povos criaram sistemas de numeração próprios. O que utilizamos é o indo-arábico.

No sistema numérico decimal, os símbolos são representados por dez algarismos, que são: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. Para compor um número, associamos um ou mais algarismos e, dependendo da posição deles, obtemos números com valores diferentes.

A posição que o algarismo ocupa no número determina quantas são as unidades, as dezenas e as centenas desse número. Observe na figura 1.1 a representação do número 5 738.

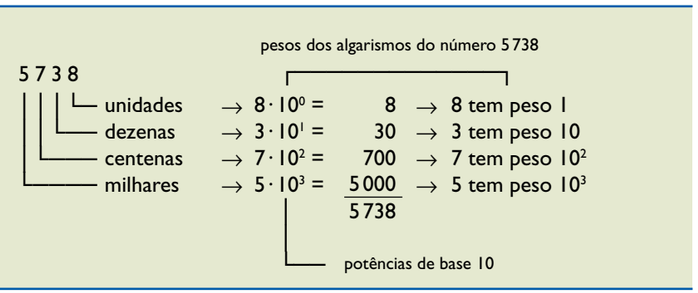


Figura 1.1
Exemplo do número 5 738 no sistema numérico decimal.

Nesse sistema, os números são representados de dez em dez; uma dezena é igual a 10 unidades, uma centena é igual a 100 unidades e um milhar é igual a 1 000 unidades. Em função dessa representação, dizemos que o sistema decimal é um sistema de **base 10**.

Exemplos

1. Nos números decimais a seguir, quais os valores dos pesos dos algarismos 3, 4 e 5?

a) 30 469
b) 179 531

Solução:

a) $30\,469 = 9 \cdot 10^0 + 6 \cdot 10^1 + \underline{4 \cdot 10^2} + 0 \cdot 10^3 + \underline{3 \cdot 10^4}$

3 tem peso ($10^4 = 10\,000$)
4 tem peso ($10^2 = 100$)

b) $179\,531 = 1 \cdot 10^0 + \underline{3 \cdot 10^1} + \underline{5 \cdot 10^2} + 9 \cdot 10^3 + 7 \cdot 10^4 + 1 \cdot 10^5$

5 tem peso ($10^2 = 100$)
3 tem peso 10

2. Qual algarismo no número decimal 54 781 tem peso 1 000?

Solução:

$54\,781 = 1 \cdot 10^0 + 8 \cdot 10^1 + 7 \cdot 10^2 + \underline{4 \cdot 10^3} + 5 \cdot 10^4$

O algarismo 4 tem peso 1 000.

1.2 Sistema numérico hexadecimal

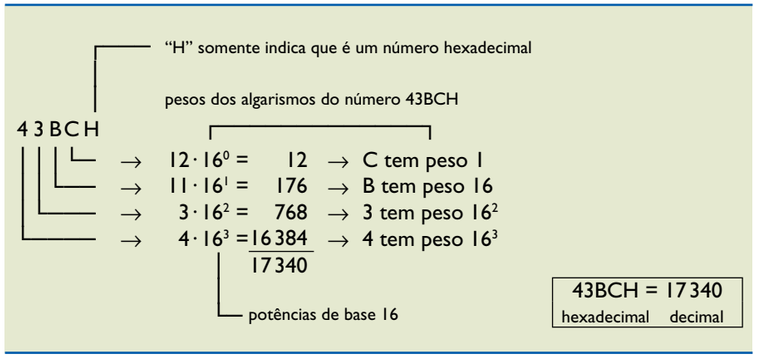
O sistema numérico hexadecimal possui 16 símbolos, representados por 16 algarismos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F.

É possível fazer correspondência entre os algarismos do sistema hexadecimal e os algarismos do sistema decimal:

Algarismos hexadecimais	0,	1,	2,	3,	4,	5,	6,	7,	8,	9,	A,	B,	C,	D,	E,	F
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Algarismos decimais	0,	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13,	14,	15

Para representarmos um número hexadecimal no sistema decimal, devemos proceder como mostra a figura 1.2.

Figura 1.2
43BCH no sistema numérico hexadecimal equivale ao número 17 340 no sistema decimal.



Dizemos que o sistema hexadecimal é um sistema de **base 16**.

Exemplos

1. Nos números hexadecimais a seguir, quais os valores dos pesos dos algarismos 2, B e C?

a) 32CH
b) B3CH

Solução:

a) $32CH = \underline{12 \cdot 16^0} + \underline{2 \cdot 16^1} + 3 \cdot 16^2$

2 tem peso 16
C tem peso $(16^0 = 1)$

b) $B3CH = \underline{12 \cdot 16^0} + 3 \cdot 16^1 + \underline{B \cdot 16^2}$

B tem peso $(16^2 = 256)$
C tem peso 1

2. Encontre o equivalente decimal dos números hexadecimais a seguir usando os pesos de cada algarismo.

a) A2CH
b) 52H

Solução:

a) $A2CH = 12 \cdot 16^0 + 2 \cdot 16^1 + 10 \cdot 16^2 = 12 + 32 + 2560 = 2604 \rightarrow A2CH = (2604)_{10} = 2604$

b) $52H = 2 \cdot 16^0 + 5 \cdot 16^1 = 2 + 80 = 82 \rightarrow 52H = (82)_{10} = 82$

O número decimal pode ser representado sem parênteses e sem índice.

1.3 Sistema numérico octal

O sistema numérico octal possui oito algarismos, representados pelos símbolos: 0, 1, 2, 3, 4, 5, 6, 7.

É possível fazer correspondência entre os algarismos do sistema octal e os algarismos do sistema decimal:

Algarismos octais	0,	1,	2,	3,	4,	5,	6,	7
	↓	↓	↓	↓	↓	↓	↓	↓
Algarismos decimais	0,	1,	2,	3,	4,	5,	6,	7

Para representarmos um número octal no sistema decimal, devemos proceder como mostra a figura 1.3.

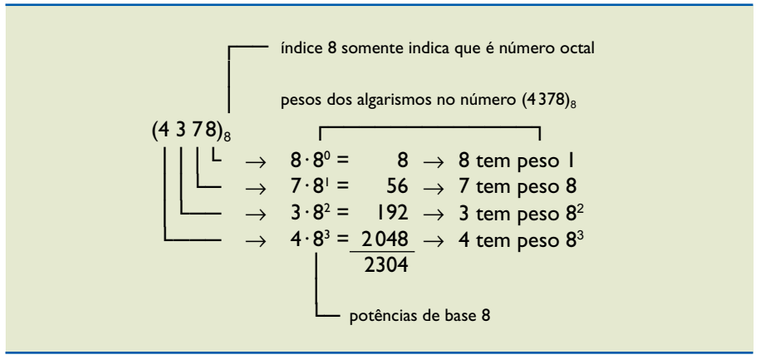


Figura 1.3
Representação do número $(4378)_8$ no sistema numérico octal. Esse número equivale ao 2304 no sistema decimal.

Dizemos que o sistema octal é um sistema de **base 8**.

Exemplos

1. Nos números octais a seguir, quais os valores dos pesos dos algarismos 2 e 7?

a) $(327)_8$
b) $(271)_8$

Solução:

a) $(327)_8 = \underline{7 \cdot 8^0} + \underline{2 \cdot 8^1} + 3 \cdot 8^2$

2 tem peso 8
7 tem peso $(8^0 = 1)$

b) $(271)_8 = \underline{1 \cdot 8^0} + 7 \cdot 8^1 + \underline{2 \cdot 8^2}$

2 tem peso $(8^2 = 64)$
7 tem peso 8

2. Encontre o equivalente decimal dos números octais a seguir usando os pesos de cada algarismo.

a) $(34)_8$
b) $(206)_8$

Solução:

a) $(34)_8 = 4 \cdot 8^0 + 3 \cdot 8^1 = 4 + 24 = 28$

b) $(206)_8 = 6 \cdot 8^0 + 0 \cdot 8^1 + 2 \cdot 8^2 = 6 + 0 + 128 = 134$

1.4 Sistema numérico binário

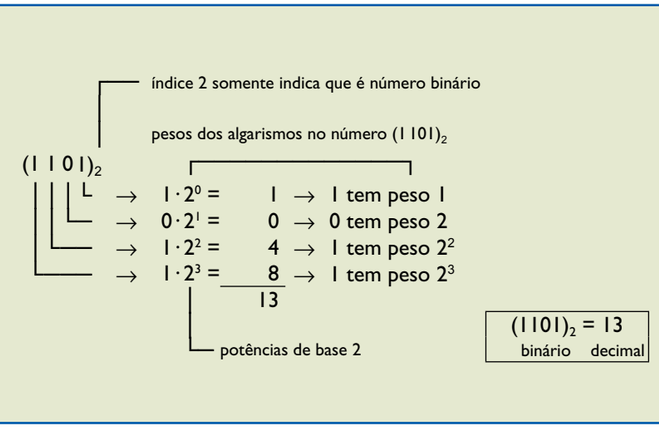
O sistema de numeração binário possui dois símbolos, representados pelos algarismos: 0 e 1.

É possível fazer correspondência entre os algarismos do sistema binário e os algarismos do sistema decimal:

Algarismos binários	0,	1
	↓	↓
Algarismos decimais	0,	1

Para representar um número binário no sistema decimal, devemos proceder como mostra a figura 1.4.

Figura 1.4
Representação do número $(1101)_2$ no sistema numérico binário. Esse número equivale ao 13 no sistema decimal.



Dizemos que o sistema binário é um sistema de **base 2**.

Nesse sistema de numeração, os algarismos podem ser chamados de dígitos. Cada dígito em um sistema binário é denominado bit (***bi****nary digi****t***). Os números binários são representados em grupos de quatro dígitos, completando-se com zero(s) à esquerda, se necessário.

Na representação dos números binários (figura 1.5), o primeiro dígito à direita é chamado dígito menos significativo (LSB, ***l****east* ***s****ignificant* ***b****it*), e o primeiro dígito à esquerda diferente de zero, dígito mais significativo (MSB, ***m****ost* ***s****ignificant* ***b****it*).

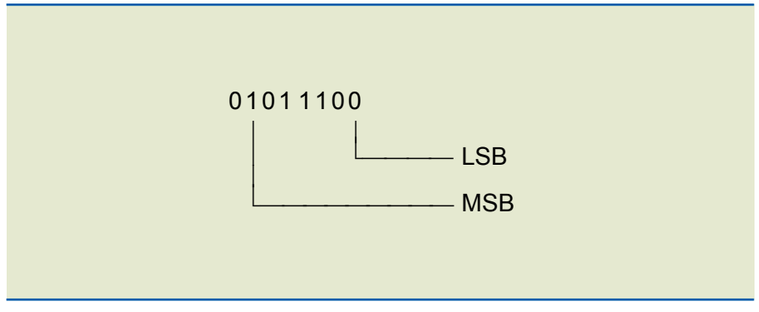


Figura 1.5
Representação do número 01011100 no sistema numérico binário.

O sistema binário é utilizado principalmente na eletrônica digital, na computação, nas telecomunicações, na robótica, na automação etc., ou seja, nas áreas que usam circuitos digitais, que, por sua vez, têm como entradas e saídas somente valores “0” e “1”.

Exemplos

1. Nos números binários a seguir, qual o valor do peso (em decimal) dos algarismos assinalados?

a) 00110111 b) 11111101

Solução:

a) 00110111
tem peso 2
tem peso $(2^5 = 32)$

b) 11111101
tem peso $(2^2 = 4)$
tem peso $(2^6 = 64)$

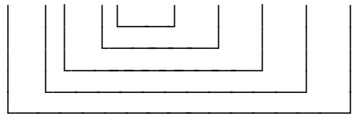
2. Encontre o equivalente decimal dos números binários a seguir usando os pesos de cada algarismo.

a) 10110110

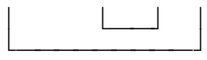
b) 01000010

Solução:

a) $1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 = 2^1 + 2^2 + 2^4 + 2^5 + 2^7 = 2 + 4 + 16 + 32 + 128 = 182$



b) $0\ 1\ 0\ 0\ 0\ 0\ 1\ 0 = 2^1 + 2^6 = 2 + 64 = 66$



3. Responda.

a) George Boole nasceu no século XIX em uma década cuja dígito LSB é 5. Estabeleça, com base nessa informação, qual é o menor intervalo de tempo em que ele nasceu. Observe que foi omitido na informação o MSB da década.

Solução:

Século XIX → 1801 a 1900. Como não podemos estabelecer a década, o menor intervalo de tempo em que com certeza ele nasceu é de 01/01/1801 a 31/12/1900. Portanto, pela informação dada, concluímos que o menor intervalo é de 100 anos.

b) O primeiro computador digital eletrônico de grande escala (ENIAC) foi apresentado no século passado na década de 1940. Estabeleça, com base nessa informação, o menor período de tempo em que com certeza, surgiu o ENIAC. Observe que foi omitido na informação o LSB da década.

Solução:

Pela informação dada, o ENIAC surgiu entre 01/01/1940 e 31/12/1949. Portanto, podemos garantir um intervalo mínimo de 10 anos. Como o enunciado da questão forneceu o MSB da década, foi possível estabelecer um intervalo de tempo mais preciso.

1.5 Conversão de sistemas numéricos (em números inteiros positivos)

1.5.1 Conversão de binário em decimal

Para convertermos número binário em decimal, somamos os pesos somente para os bits de valor "1", obtendo, assim, o equivalente decimal.

Exemplos

1. Converta $(1010)_2$ em decimal.

Solução:

$1\ \ 0\ \ 1\ \ 0$

$2^3 \qquad 2 \qquad \rightarrow (8 + 2) = 10$, portanto $(1010)_2 = 10$

2. Converta $(10111001)_2$ em decimal.

Solução:

$1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1$

$2^7 \quad 2^5\ 2^4\ 2^3 \qquad 2^0 \quad \rightarrow (128 + 32 + 16 + 8 + 1) = 185$

$(10111001)_2 = 185$

1.5.2 Conversão de decimal em binário

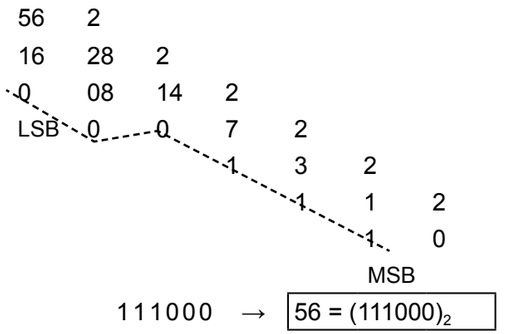
Para convertermos número decimal em binário, **agrupamos os restos** das divisões sucessivas do número por 2, até que a última divisão tenha quociente igual a zero.

Exemplo

Converta o decimal 56 em binário.

Solução:

Observe como foram agrupados os bits da coluna correspondente aos restos das divisões, para formar o binário equivalente. Depois de determinar os restos das divisões, eles são ajustados para representar dois grupos de quatro bits.



1.5.3 Conversão de hexadecimal em decimal

Para convertermos número hexadecimal em decimal, somamos os pesos multiplicados pelos números correspondentes em decimal, obtendo, assim, o equivalente decimal.

Exemplo

Converta (A8E6H) em decimal.

Solução:

$$A8E6H = 10 \cdot 16^3 + 8 \cdot 16^2 + 14 \cdot 16 + 6 \cdot 16^0 =$$
$$= 40960 + 2048 + 224 + 6 = 43238$$

$$\boxed{A8E6H = 43238}$$

1.5.4 Conversão de decimal em hexadecimal

O processo é semelhante ao da conversão de decimal em binário.

Exemplo

Converta (2470) em hexadecimal.

Solução:

2470	16		
87	154	16	
70	10	9	16
6		9	0
LSB		MSB	

9 A 6 → $\boxed{2470 = 9A6H}$

Observe que 6, 10 e 9 são os restos das divisões; 10 foi substituído por seu equivalente hexadecimal A.

1.5.5 Conversão de octal em decimal

Exemplo

Converta $(2075)_8$ em decimal.

Solução:

$$(2075)_8 = 2 \cdot 8^3 + 7 \cdot 8^1 + 5 \cdot 8^0$$
$$= 1024 + 56 + 5 = 1085$$

$$\boxed{(2075)_8 = 1085}$$

1.5.6 Conversão de decimal em octal

O processo é semelhante ao da conversão de decimal em binário.

Exemplo

Converta (1085) em octal.

Solução:

1085	8			
28	135	8		
45	55	16	8	
5	7	0	2	8
LSB			2	0
			MSB	

2 0 7 5 → $\boxed{1085 = (2075)_8}$

1.5.7 Conversão de octal em binário

Para convertermos número octal em binário, convertemos dígito a dígito de octal em binário, da direita para a esquerda, em grupos de três bits. O último grupo completamos com zero(s) à esquerda, se necessário.

Exemplo

Converta $(32075)_8$ em binário.

Solução:

3	2	0	7	5
↓	↓	↓	↓	↓
011	010	000	111	101

$$\boxed{(32075)_8 = (0011\ 0100\ 0011\ 1101)_2}$$

Após a conversão, fazemos a representação usual em grupos de quatro bits, completando com zeros à esquerda.

Agora, calcule o equivalente decimal de $(32075)_8$ e o equivalente decimal de $(0011\ 0100\ 0011\ 1101)_2$. Compare esses valores.

1.5.8 Conversão de binário em octal

Para convertermos número binário em octal, separamos o número binário em grupos de três bits, da direita para a esquerda, completando o último grupo com zero(s), se necessário. Convertemos em octal cada grupo. Lembre-se de que de 0 a 7 os valores octais e decimais são representados pelos mesmos dígitos.

Exemplo

Converta $(1011\ 0010)_2$ em octal.

Solução:

010	110	010
↓	↓	↓
2	6	2

$(10110010)_2 = (262)_8$

1.5.9 Conversão de hexadecimal em binário

Para convertermos número hexadecimal em binário, fazemos a conversão dígito a dígito de hexadecimal em binário, da direita para a esquerda, em grupos de quatro bits. O último grupo à esquerda completamos com zero(s), se necessário.

Exemplo

Converta (1ADH) em binário.

Solução:

1	A	D
↓	↓	↓
0001	1010	1101

$1ADH = (0001\ 1010\ 1101)_2$

1.5.10 Conversão de binário em hexadecimal

Para convertermos número binário em hexadecimal, separamos o número binário em grupos de quatro bits, da direita para a esquerda, completando o último grupo com zero(s), se necessário. Convertemos em hexadecimal cada grupo.

Exemplo

Converta $(0001\ 1010\ 1101)_2$ em hexadecimal.

Solução:

0001	1010	1101
↓	↓	↓
1	A	D

$(0001\ 1010\ 1101)_2 = 1ADH$

1.5.11 Conversão de octal em hexadecimal

Para convertermos número octal em hexadecimal, realizamos duas etapas:

octal → binário → hexadecimal

1.5.12 Conversão de hexadecimal em octal

Para convertermos número hexadecimal em octal, realizamos duas etapas:

hexadecimal → binário → octal

1.5.13 Resumo de conversão de sistemas

- Na conversão de qualquer outro sistema em decimal, usamos o peso do dígito.
- Na conversão de decimal em qualquer outro sistema, efetuamos divisões sucessivas.

A figura 1.6 apresenta o resumo de conversão. Não se preocupe em decorá-la pois ela poderá ser consultada sempre que necessário. Entretanto, a associação dos lembretes escritos com o processo de conversão deve estar bem clara.

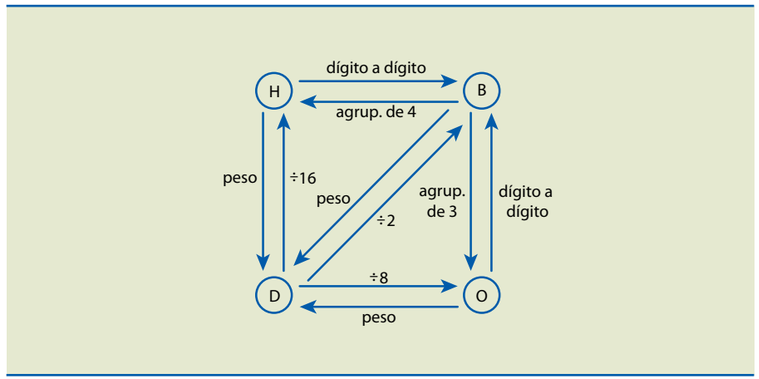


Figura 1.6
Resumo de conversão de sistemas.

A tabela 1.1 também não precisa ser memorizada. Sua construção pode ser feita rapidamente observando na coluna dos valores binários o avanço dos números "1" da direita para a esquerda, ao passar de uma linha para a seguinte. Tente reproduzir a tabela sem consultá-la pois isto é importante.

Tabela 1.1
Resumo das equivalências entre os números binários, decimais e hexadecimais (de 0 a 15 em decimal).

B				D	H
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

Os exercícios a seguir são exemplos de conversão de números positivos não inteiros, apresentados como complemento, uma vez que estão além dos objetivos deste livro.

Exemplos

1. Converta $(1011,1001)_2$ em decimal.

Solução:

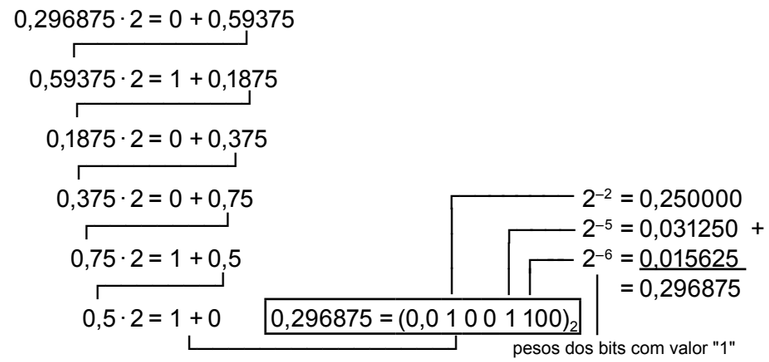
1 0 1 1, 1 0 0 1

2^3 2^1 2^0 2^{-1} 2^{-4} $\rightarrow$ (8 + 2 + 1 + 0,5 + 0,0625) = 11,5625

$(1011,1001)_2 = 11,5625$

2. Converta o decimal (0,296875) em binário.

Solução:



Observe que o lado direito da igualdade é a decomposição do resultado em parte inteira e parte fracionária. O processo deve cessar quando a parte fracionária da decomposição do número for zero ou quando a aproximação obtida for suficiente. O agrupamento de quatro bits é ajustado com o acréscimo de zero(s) **à direita**.

3. Converta (A8E6,39H) em decimal.

Solução:

$A8E6,38H = 10 \cdot 16^3 + 8 \cdot 16^2 + 14 \cdot 16 + 6 \cdot 16^0 + 3 \cdot 16^{-1} + 8 \cdot 16^{-2}$

$= 40960 + 2048 + 224 + 6 + 0,1875 + 0,03125 = 43238,21875$

$A8E6,38H = 43238,21875$

Capítulo 2

Funções lógicas

George Boole (1815-1864), matemático e filósofo britânico, criou um sistema matemático de análise lógica chamado álgebra de Boole ou álgebra booleana. Esse sistema permitiu elaborar expressões conhecidas como funções lógicas, que possibilitaram o desenvolvimento da eletrônica digital. Para iniciar o estudo, vamos analisar o circuito da figura 2.1.

Sejam as variáveis S1, S2 e L, tais que:

S1 = S2 = 0 → chaves abertas
S1 = S2 = 1 → chaves fechadas
L = 0 → lâmpada apagada
L = 1 → lâmpada acesa

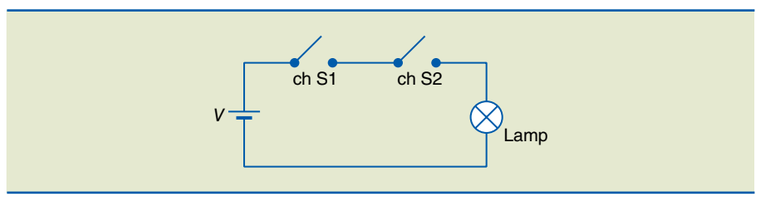


Figura 2.1
Circuito elétrico com duas chaves e uma lâmpada.

Assim, por exemplo:

- Se S1 = 1 (chave S1 fechada) e S2 = 1 (chave S2 fechada) → L = 1 (lâmpada acesa)
- Se S1 = 1 (chave S1 fechada) e S2 = 0 (chave S2 aberta) → L = 0 (lâmpada apagada)

A condição da lâmpada (acesa/apagada) é função (depende) da condição de cada uma das chaves (aberta/fechada) do circuito. Nessa função, não são consideradas quantidades (números), e sim os estados de variáveis, em que somente duas condições são possíveis: "0" ou "1". Essas variáveis, que podem assumir apenas dois estados (0/1, aberto/fechado, sim/não, verdadeiro/falso etc.), são chamadas **variáveis booleanas**, e os estados, **estados lógicos**, associados às variáveis. Quando estão atuando nessas condições, as variáveis booleanas são conhecidas como **funções booleanas**, que podem ser simples ou complexas. As funções booleanas simples são obtidas por meio de um conjunto de circuitos eletrônicos denominados **portas lógicas**. Associando portas lógicas, é possível implementar circuitos eletrônicos definidos por funções booleanas mais complexas.

As variáveis utilizadas nos circuitos são representadas pelas letras A, B, C, ..., N. Uma barra sobre uma variável booleana significa que seu valor sofrerá inversão.

Assim, se A = 0, $\overline{A}$ = 1, e se A = 1, $\overline{A}$ = 0, em que $\overline{A}$ lê-se: não A, A barra, A barrado ou complemento de A.

As funções booleanas apresentam resultados fornecidos pelas combinações possíveis devido a suas variáveis. Esses resultados são normalmente representados em forma de tabela.

Chamamos **tabela verdade** de uma função booleana a tabela que apresenta, geralmente de maneira ordenada, os valores da função y = f(A, B) para todas as combinações possíveis dos valores das variáveis.

Consideremos y uma função booleana das variáveis A e B, cuja tabela verdade é apresentada na tabela 2.1.

A	B	y
0	0	1
0	1	0
1	0	1
1	1	1

Tabela 2.1
Tabela verdade de y = f(A, B)

A tabela verdade é uma das maneiras de estabelecer a correspondência entre os valores da função e os das variáveis. A penúltima linha da tabela, por exemplo, informa que, nas condições A = 1 e B = 0, y = 1. Outra forma de estabelecer a correspondência é a expressão booleana da função, que será abordada mais adiante.

2.1 Portas lógicas

Portas lógicas são circuitos eletrônicos básicos que possuem uma ou mais entradas e uma única saída. Nas entradas e na saída, podemos associar estados "0" ou "1", ou seja, variáveis booleanas. Em eletrônica digital, quando utilizamos portas lógicas, atribuímos às entradas e às saídas valores de tensão. Nos circuitos exemplos de portas lógicas, associaremos ao 5 V o estado "1" e ao 0 V, o estado "0".

A porta lógica mais simples é denominada inversora. Nela, a saída é igual ao complemento da entrada (figura 2.2).

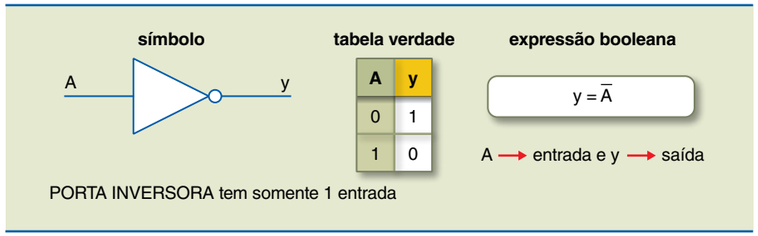
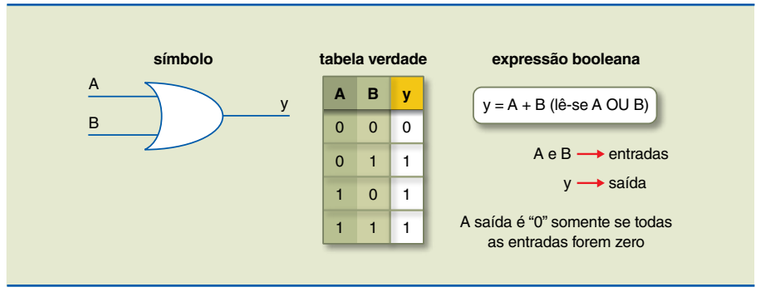


Figura 2.2
Símbolo, tabela verdade e expressão booleana da porta inversora.

A porta OU (OR, em inglês) possui duas ou mais entradas. A saída sempre será igual a "1" quando uma das entradas for igual a "1" (figura 2.3). A saída será "0" somente se todas as entradas forem "0".

Figura 2.3
Símbolo, tabela verdade e expressão booleana da porta OU.



O símbolo "+" representa OU lógico e não significa uma soma aritmética, pois "0" e "1" não são números, mas estados lógicos das variáveis.

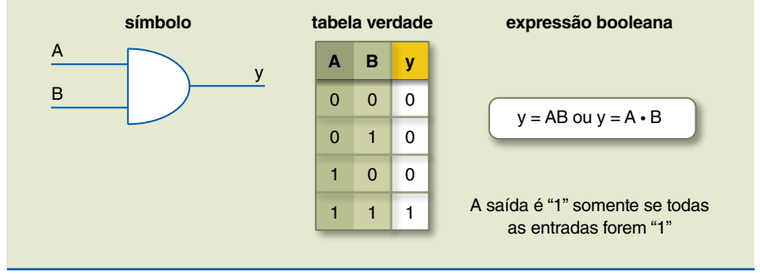
A porta NOU (NOR) corresponde à uma porta OU com a saída invertida (figura 2.4). A saída será "1" somente se todas as entradas forem "0".

Figura 2.4
Símbolo, tabela verdade e expressão booleana da porta NOU.

Observe que a "bolinha" no símbolo nega (complementa) a saída, equivalente à barra na expressão booleana, indicando que a porta NOU tem uma saída que corresponde ao complemento da saída da porta OU.

A porta E (AND) possui uma ou mais entradas e sua saída será "1" somente quando todas as entradas forem iguais a "1" (figura 2.5).

Figura 2.5
Símbolo, tabela verdade e expressão booleana da porta E.

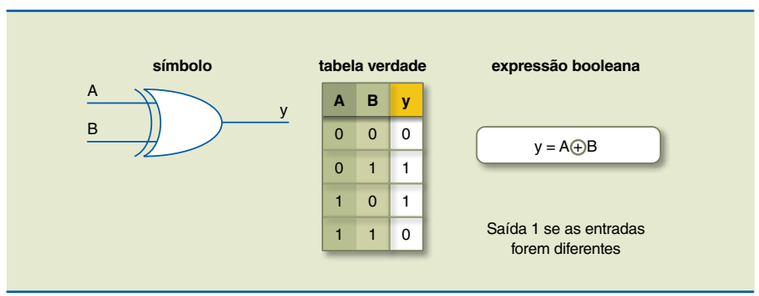


A porta NE (NAND) corresponde a uma porta E com a saída invertida (figura 2.6). A saída será "0" somente se todas as entradas forem "1".

Figura 2.6
Símbolo, tabela verdade e expressão booleana da porta NE.

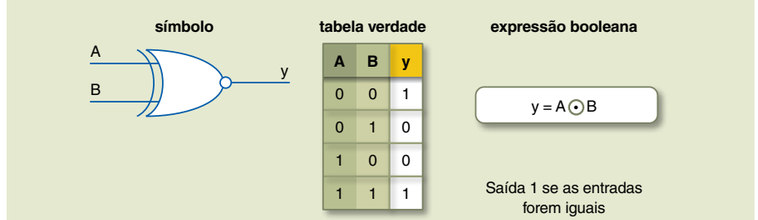
A porta OU EXCLUSIVO (XOR) possui uma ou mais entradas e fornecerá uma saída igual a "1" somente quando as entradas forem diferentes (figura 2.7).

Figura 2.7
Símbolo, tabela verdade e expressão booleana da porta OU EXCLUSIVO.



A porta NOU EXCLUSIVO (XNOR), também chamada de COINCIDÊNCIA, é equivalente a uma porta XOR com a saída invertida (figura 2.8). A saída será "1" se as entradas forem iguais.

Figura 2.8
Símbolo, tabela verdade e expressão booleana da porta NOU EXCLUSIVO.



2.2 Álgebra booleana

Vimos que na álgebra booleana o estudo de circuitos lógicos é baseado em apenas dois valores (0/1, aberto/fechado, sim/não, verdadeiro/falso etc.), que também podem ser representados por dois níveis distintos de tensão, chamados, por

exemplo, nível alto (H – *high*) e nível baixo (L – *low*) ou simplesmente “0” (zero) e “1” (um). A análise das expressões também obedece a esse princípio e, portanto, é perfeitamente aplicável a nosso estudo.

Os símbolos H/L ou 0/1 podem ser empregados para representar situações do tipo:

- sim/não;
- verdadeiro/falso;
- ligado/desligado (*on/off*);
- aceso/apagado.

Obviamente, essas representações devem estar relacionadas a suas respectivas variáveis. Por exemplo, suponhamos que a uma chave do tipo liga/desliga seja atribuída a variável “K”. Com base nessa atribuição, podemos representar o estado da respectiva chave em um circuito como:

- K = 0 (zero) para a condição chave desligada (aberta);
- K = 1 (um) para a condição chave ligada (fechada).

Além disso, as funções booleanas são expressões que representam as relações entre as variáveis envolvidas em determinado processo por meio dos operadores lógicos “AND” (·) e “OR” (+).

Exemplo

Um sistema de alarme deverá soar quando os sensores A e C estiverem ativados ao mesmo tempo ou quando a chave B estiver ligada e pelo menos um dos sensores estiver ativado. Um modo de encontrar a solução para o problema é a tabela verdade. Para isso, constrói-se a tabela verdade com as variáveis de entrada envolvidas no problema proposto (no caso, A, B, C) e verificam-se, de acordo com a expressão, os níveis que a variável de saída (S) deverá possuir (tabela 2.2).

Tabela verdade

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Toda função booleana de N variáveis pode ser escrita na forma canônica disjuntiva ou conjuntiva.

A forma canônica disjuntiva é obtida da tabela verdade de acordo com o seguinte procedimento:

a) Escreva um termo (operação lógica “E”) para cada linha em que a função é igual a “1”.

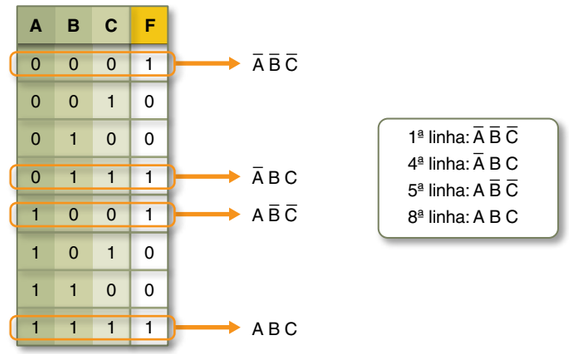
b) Junte os termos obtidos no item anterior com a operação “OU” (+).

Obs.: as variáveis serão barradas ou não conforme seu valor seja “0” ou “1” naquela linha.

Exemplo

Seja a tabela verdade a seguir

Tabela verdade



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

A forma canônica conjuntiva é obtida da tabela verdade de acordo com o seguinte procedimento:

a) Escreva um termo (operação lógica “OU”) para cada linha em que a função tem valor “0”.

b) Junte os termos obtidos no item anterior com a operação “E” (·).

Obs.: as variáveis serão barradas se naquela linha seu valor for “1” e não barrada se seu valor for “0”.

Exemplo

Na tabela verdade do exemplo anterior, verifica-se que a função é igual a “0” na segunda, terceira, sexta e sétima linhas.

Tabela verdade

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

2ª linha: $A + B + \overline{C}$
3ª linha: $A + \overline{B} + C$
6ª linha: $\overline{A} + B + \overline{C}$
7ª linha: $\overline{A} + B + C$

$$F = (A + B + \overline{C}) \cdot (A + \overline{B} + C) \cdot (\overline{A} + B + \overline{C}) + (\overline{A} + \overline{B} + C)$$

2.2.1 Propriedades e teoremas da álgebra booleana

Os teoremas e propriedades da álgebra booleana permitem a simplificação de circuitos lógicos, objetivo final de todo projeto de circuitos digitais. As propriedades mais importantes são apresentadas a seguir.

Propriedade da intersecção

Está relacionada com as portas E. Os casos possíveis são:

$A \cdot 1 = A$
$A \cdot 0 = 0$

Obs.: essa propriedade é aplicável a um maior número de variáveis de entrada.

Exemplos

$A \cdot B \cdot 1 = A \cdot B$
$A \cdot B \cdot 0 = 0$

Propriedade da união

Está relacionada com as portas OU e divide-se em dois casos:

$B + (1) = 1$
$B + (0) = B$

Essa propriedade também é válida para portas OU com mais de duas entradas.

Exemplos

$A + B + (1) = 1$
$A + B + (0) = A + B$

Propriedade da tautologia

Palavra de origem grega usada em lógica para descrever uma proposição que é verdadeira quaisquer que sejam os valores de suas variáveis.

É válida para portas E e portas OU e pode ser verificada nos seguintes casos:

$A \cdot A = A$
$A + A = A$

Essa propriedade é válida para um maior número de variáveis.

Exemplo

$A \cdot B + A \cdot B + C = A \cdot B + C$

Propriedade dos complementos

Se aplicarmos um sinal lógico e seu complemento a uma porta lógica, simultaneamente a saída será “0” ou “1”, dependendo do tipo de porta.

Exemplos

$A \cdot \overline{A} = 0$
$A + \overline{A} = 1$

Propriedade da dupla negação

Essa propriedade afirma que o complemento do complemento de uma variável é igual a ela própria. Em forma de expressão matemática, temos, como exemplo:

$\overline{\overline{A}} = A$

Propriedade comutativa

Essa propriedade é semelhante à da álgebra convencional e pode ocorrer nos seguintes casos:

$A \cdot B = B \cdot A$
$A + B = B + A$

Propriedade associativa

É outra propriedade semelhante à da álgebra convencional. Os casos possíveis são:

$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C$
$A + (B + C) = (A + B) + C = A + B + C$

Propriedade distributiva

Também é semelhante à da álgebra convencional.

Exemplos

$A \cdot (B + C) = A \cdot B + A \cdot C$
$A + B \cdot C = (A + B) \cdot (A + C)$

Propriedade da absorção

Os casos mais elementares são:

$A + A \cdot B = A$
$A + A \cdot B = A + B$
$(A + B) \cdot B = A \cdot B$

Em decorrência dessas identidades, podemos encontrar outras um pouco mais complexas:

$A \cdot B + A \cdot B = A$
$(A + B) \cdot (A + B) = A$
$A \cdot (A + B) = A$
$A \cdot (A + B) = AB$
$A \cdot B + A \cdot C = (A + C) \cdot (A + B)$

Dualidade

Seja F uma função booleana. Define-se a **função dual** de F como aquela obtida quando mudamos os operadores + por · e · por + e os valores "0" por "1" e "1" por "0".

Postulados da dualidade:

1a) $X = 0$ se $x \neq 1$
2a) $X = 1$ se $x = 0$
3a) $0 \cdot 0 = 0$
4a) $1 \cdot 1 = 1$
5a) $1 \cdot 0 = 0 \cdot 1 = 0$

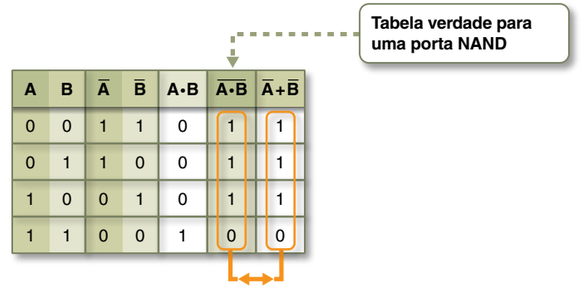
1b) $X = 1$ se $X \neq 0$
2b) $X = 0$ se $X = 1$
3b) $1 + 1 = 1$
4b) $0 + 0 = 0$
5b) $0 + 1 = 1 + 0 = 1$

1º teorema de De Morgan

"O complemento do produto é igual à soma dos complementos"

$\overline{A} \cdot \overline{B} = \overline{A} + \overline{B}$

Podemos comprovar esse teorema pela tabela verdade a seguir:



A	B	$\overline{A}$	$\overline{B}$	A•B	$\overline{A \cdot B}$	$\overline{A}+\overline{B}$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

2º teorema de De Morgan

"O complemento da soma é igual ao produto dos complementos"

$\overline{A + B} = \overline{A} \cdot \overline{B}$

Esse teorema também pode ser comprovado pela tabela verdade.

Como consequência dos teoremas de De Morgan as funções lógicas já conhecidas podem ser reescritas por um bloco equivalente, permitindo, assim, redesenhar os circuitos lógicos caso seja conveniente.

As equivalências básicas são:

a)Portas NAND (figura 2.9).

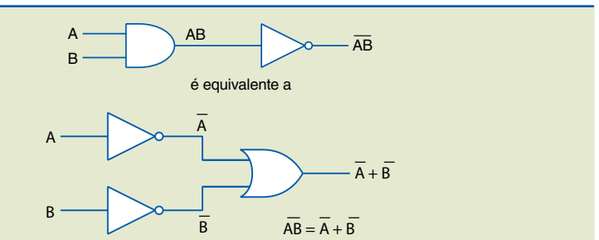


Figura 2.9
Equivalência entre as portas NAND.

Ou seja (figura 2.10):

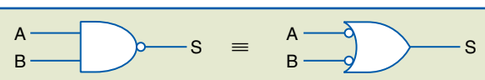


Figura 2.10
Representações simplificadas das portas NAND.

b)Portas NOR (figura 2.11).

Figura 2.11
Representações simplificadas das portas NOR.

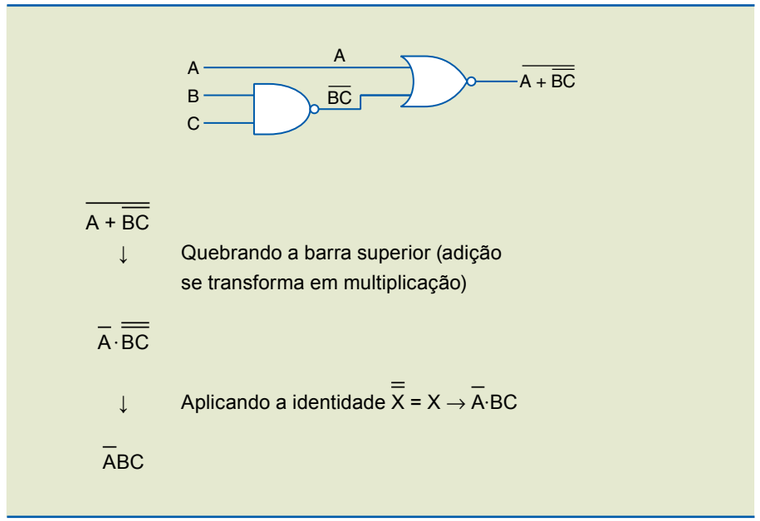
Exemplo

Consideremos a seguinte expressão lógica:

$$\overline{(A + (\overline{B \cdot C}))}$$

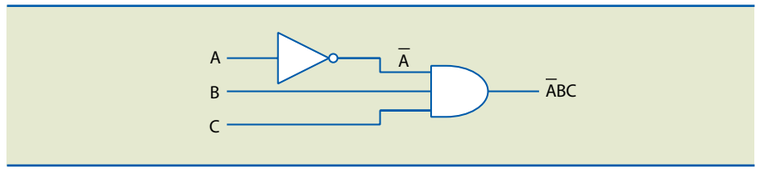
O circuito lógico correspondente implementado com portas lógicas E, OU e INVERSORAS terá o aspecto ilustrado na figura 2.12.

Figura 2.12
Representação do circuito lógico com portas lógicas E, OU e INVERSORAS.



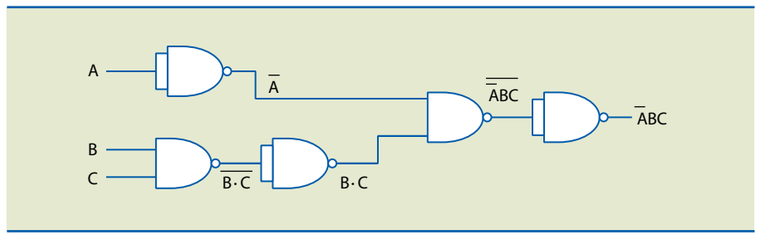
Pela aplicação das identidades do circuito da figura 2.12, o circuito lógico reduz-se conforme apresenta a figura 2.13.

Figura 2.13
Representação simplificada do circuito lógico com portas lógicas E, OU e INVERSORAS.



Reaplicando os teoremas de De Morgan para substituir os blocos lógicos da figura 2.13 pelos equivalentes, obtemos a figura 2.14.

Figura 2.14
Representação simplificada do circuito lógico com portas lógicas E, OU e INVERSORAS com substituição dos blocos lógicos da figura 2.13 por seus equivalentes.



2.3 Descrição de funções lógicas

Os circuitos lógicos podem ser representados por funções booleanas, ou seja, admite-se que todos os circuitos lógicos estabelecem as relações entre entradas e saída obedecendo à função booleana que os representa. Quando necessário, é possível obter a função booleana por meio da tabela verdade do circuito. Além disso, o circuito lógico pode ser descrito pela conexão de portas lógicas básicas, independentemente de sua complexidade. A seguir, são descritas as relações entre as formas de representação de um circuito lógico.

2.3.1 Circuito lógico

Consideremos o circuito lógico da figura 2.15. Vamos obter a função lógica S = f(A, B, C, D), da saída do circuito.

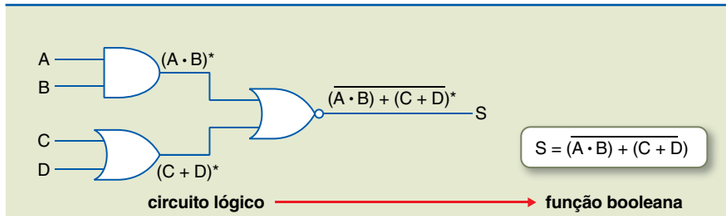


Figura 2.15
Representação da função y = A, B, C, D.

Analisando esse circuito, podemos notar que colocamos na saída de cada porta lógica a expressão booleana correspondente (*), que será a entrada de outra porta lógica, e assim repetimos o procedimento sucessivamente até a expressão booleana da saída.

Vamos analisar outra situação, considerando a função booleana $y = A \cdot B + C \cdot (B + D)$. Como se trata de uma expressão algébrica (álgebra booleana), devemos respeitar na implementação do circuito a ordem das operações, associando a multiplicação à operação "E" e a soma à operação "OU". As operações entre parênteses devem ser feitas agrupadas (figura 2.16).

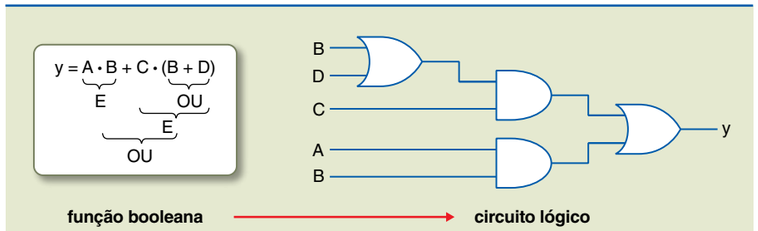


Figura 2.16
Representação da função y = A · B + C · (B+D).

2.3.2 Tabela verdade 2

Vamos obter a tabela verdade da função booleana $y = A \cdot B \cdot C + AC + BC$. Para isso, adotamos o seguinte procedimento:

1) Montamos a coluna completa de todas as combinações possíveis das variáveis (número de linhas = $2^n + 1$, n = número de variáveis).
2) Montamos as colunas auxiliares em quantidade igual ao número de "parcelas" da função booleana.
3) Montamos a última coluna para y.

Os cálculos referentes às colunas das "parcelas" da função booleana, em geral, são feitos mentalmente ou em rascunho.

Tabela verdade de $y = A \cdot B \cdot C + AC + BC$

A	B	C	A•B•C	AC	BC	y
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	1	0	1
1	1	1	0	0	1	1

É possível obter a expressão booleana por meio da tabela verdade. Para isso, vamos considerar a tabela verdade a seguir:.

Tabela verdade

	A	B	C	y
1	0	0	0	0
2	0	0	1	0
3	0	1	0	1
4	0	1	1	0
5	1	0	0	1
6	1	0	1	1
7	1	1	0	0
8	1	1	1	1

Para montarmos a função booleana a partir dos valores da tabela verdade, adotamos o seguinte procedimento:

1) Consideramos somente as linhas da tabela em que $y = 1$.
2) Fazemos "E" das variáveis que têm valor "1" com os complementos das que têm valor "0", por exemplo:

linha 3 → A=0, B=1 e C=0 → $\overline{A} \cdot B \cdot \overline{C}$
linha 5 → A=1, B=0 e C=0 → $A \cdot \overline{B} \cdot \overline{C}$
linha 6 → A=1, B=0 e C=1 → $A \cdot \overline{B} \cdot C$
linha 8 → A=1, B=1 e C=1 → $A \cdot B \cdot C$

3) Fazemos "OU" dos valores obtidos

$y = A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C$

Obs.: a numeração das linhas registradas à esquerda não é necessária; serve somente como referência para a explicação.

2.3.3 Simplificação de funções lógicas

O mapa (ou diagrama) de Karnaugh é uma forma ordenada utilizada para minimizar uma expressão lógica, que geralmente produz um circuito com configuração mínima. É construído com base na tabela verdade e pode ser facilmente aplicado em funções envolvendo duas a seis variáveis. No caso de sete ou mais variáveis, o método torna-se complicado e devemos usar técnicas mais elaboradas.

Representa-se o mapa de Karnaugh por uma tabela em forma de linhas e colunas. Essa tabela, de acordo com o número de variáveis, é dividida em células obedecendo à proporção 2^n, em que n é o número de variáveis de entrada envolvidas.

Mapa para uma variável de entrada (figura 2.17)

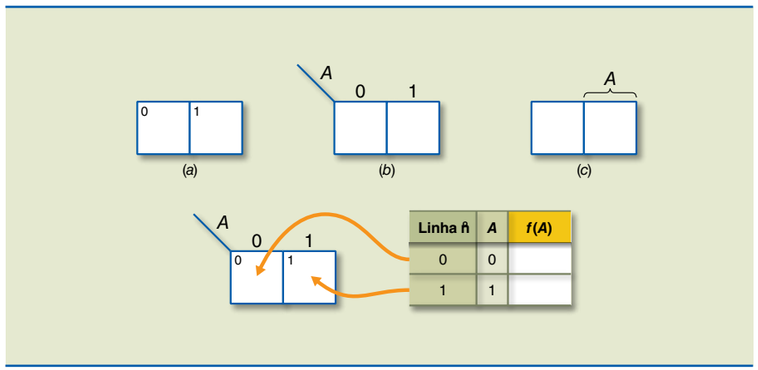


Figura 2.17
Mapa para uma variável de entrada.

Mapa para duas variáveis de entrada (figura 2.18)

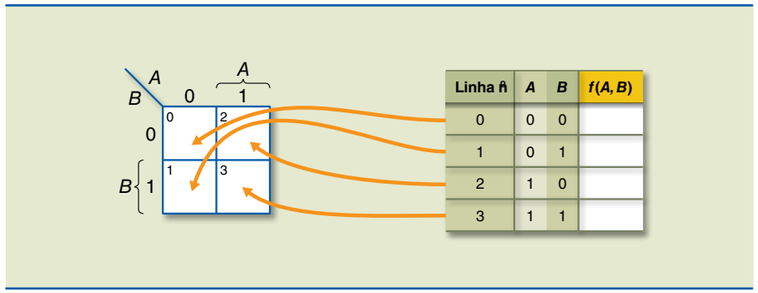


Figura 2.18
Mapa para duas variáveis de entrada.

A figura 2.19 apresenta a tabela verdade e o mapa de Karnaugh correspondente para duas variáveis.

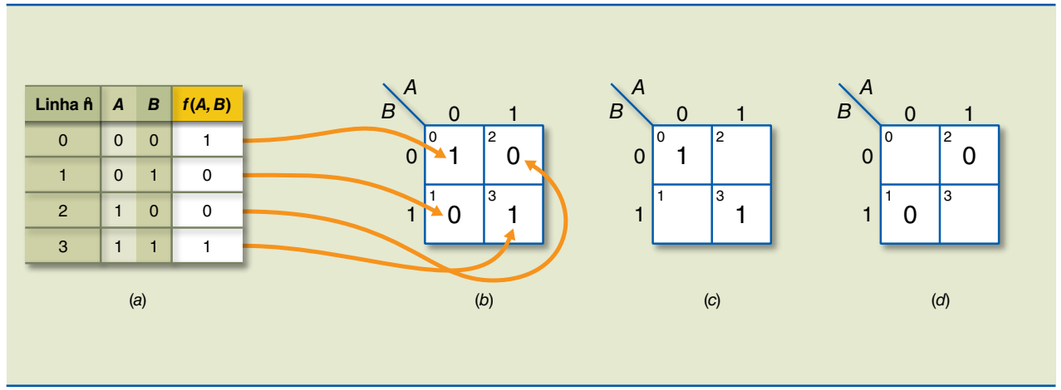


Figura 2.19
Representação do mapa para duas variáveis de entrada.

Mapa para três variáveis de entrada (figura 2.20)

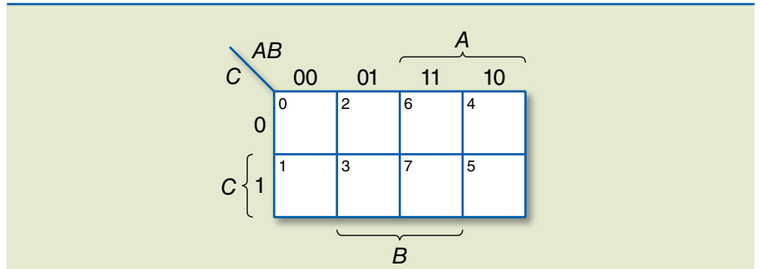


Figura 2.20
Mapa para três variáveis de entrada.

A figura 2.21 apresenta um exemplo de como deve ser representado o mapa para três variáveis, a partir da tabela verdade correspondente.

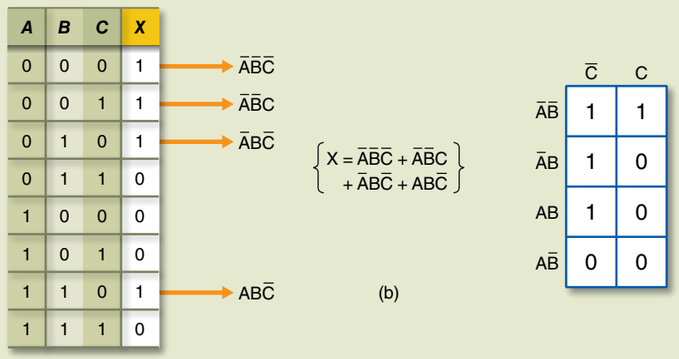


Figura 2.21
Mapa para três variáveis de entrada.

Mapa para quatro variáveis de entrada (figura 2.22)

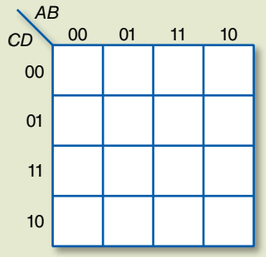


Figura 2.22
Mapa para quatro variáveis de entrada.

A figura 2.23 apresenta um exemplo de como deve ser representado o mapa para quatro variáveis, a partir da tabela verdade correspondente.

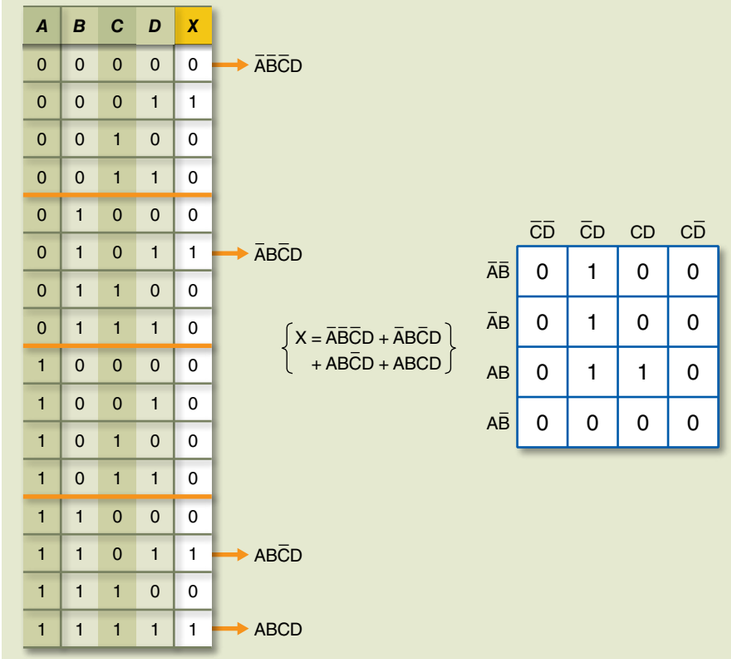
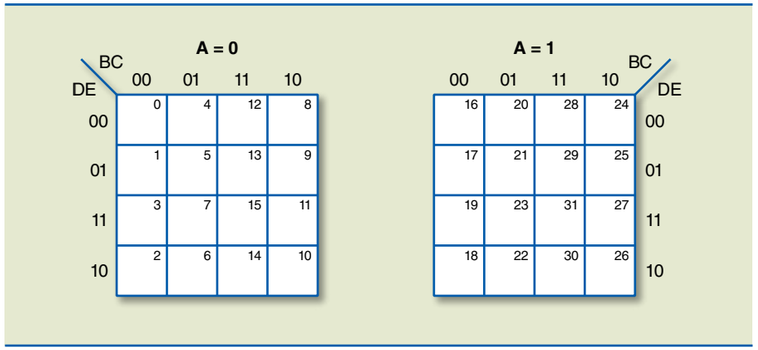


Figura 2.23
Mapa para quatro variáveis de entrada.

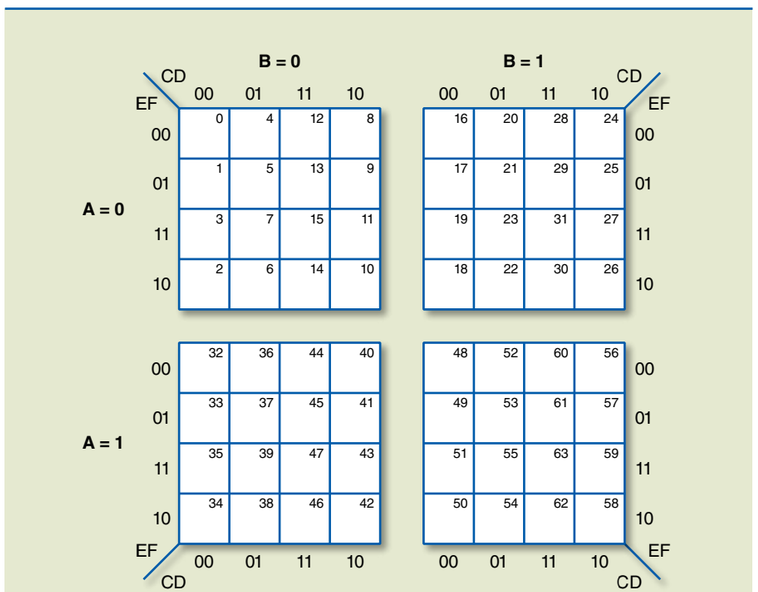
Mapa para cinco variáveis de entrada (figura 2.24)

Figura 2.24
Mapa para cinco variáveis de entrada.



Na figura 2.25, podemos observar a representação do mapa para seis variáveis de entrada.

Figura 2.25
Mapa para seis variáveis de entrada.



A seguir, vamos analisar o processo de minimização utilizando os diagramas de Karnaugh e, depois, ver alguns exemplos.

Minimização de funções utilizando o mapa de Karnaugh

Para realizarmos a minimização de funções lógicas utilizando o método do mapa de Karnaugh, devemos obedecer às seguintes regras:

1) Identificar as células nas quais os níveis de saída são iguais a "1".
2) Formar enlaces ou agrupamentos de células logicamente adjacentes cujos níveis de saída são iguais a "1".

Obs.: duas células são adjacentes se apenas uma das variáveis de entrada correspondentes troca de valor; portanto, as células localizadas nos vértices do mapa também são adjacentes entre si.

3) Os agrupamentos formados devem conter o maior número possível de células logicamente adjacentes, mas esse número tem sempre de ser uma potência de 2, ou seja, agrupamentos que tenham 1, 2, 4, 8, 16, 32, ... elementos.

Agrupamentos possíveis	
HEXA	16 quadros
OITAVA	8 quadros
QUADRA	4 quadros
PAR	2 quadros
TERMO ISOLADO	1 quadro

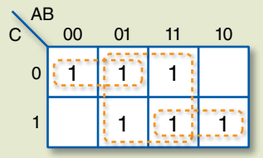
Nota: sempre que um grupo é formado, a variável que muda de estado é a eliminada. Por exemplo: se o grupo engloba parte da região A e parte da região $\overline{A}$, a variável A é eliminada.

4) Cada agrupamento assim formado corresponde a uma função lógica "E" envolvendo as variáveis de entrada entre uma célula e outra que mantêm o nível lógico.
5) A expressão lógica final corresponde a uma função "OU" envolvendo os agrupamentos anteriormente mencionados.

Exemplos de minimização

Exemplos para três variáveis de entrada

1. $Z = f(A, B, C) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B + AB\overline{C} + AC$ (figura 2.26)



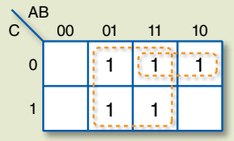
A expressão lógica minimizada é $B + AC + \overline{A}\,\overline{C}$

Figura 2.26
Simplificação das três variáveis de entrada para o exemplo 1.

2. $Z = f(A, B, C) = \overline{A}B + B\overline{C} + BC + A\overline{B}\,\overline{C}$ (figura 2.27)

A expressão lógica minimizada é $B + A\overline{C}$.

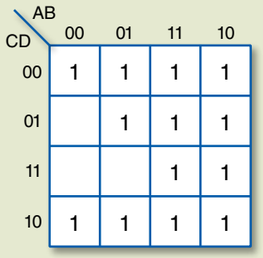
Figura 2.27
Simplificação das três variáveis de entrada para o exemplo 2.



Exemplos para quatro variáveis de entrada

1. Dado o diagrama de Karnaugh da figura 2.28, obtenha a expressão lógica minimizada.

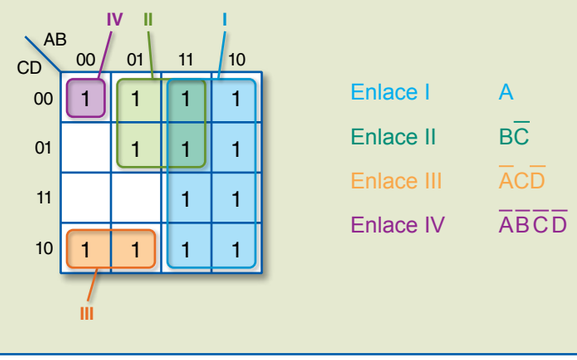
Figura 2.28
Simplificação das quatro variáveis de entrada para o exemplo 1.



Solução:

Para ilustrar o processo, primeiramente não de forma ideal, suponhamos que tivéssemos selecionado os agrupamentos apresentados na figura 2.29.

Figura 2.29
Representação dos quatro enlaces.

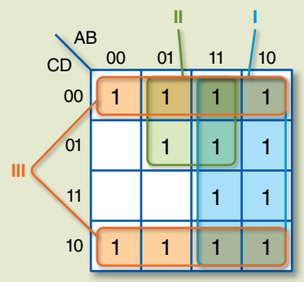


De acordo com os enlaces anteriores, a expressão obtida seria:

$f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}C\overline{D} + B\overline{C} + A$

Mas será essa a expressão mínima? Se selecionarmos adequadamente os enlaces de acordo com as regras expostas anteriormente, obteremos a figura 2.30.

Figura 2.30
Nova representação com os três enlaces.



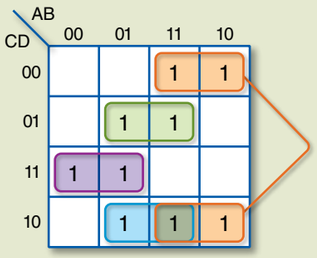
Considerando esses novos enlaces, obteremos a seguinte expressão mínima:

$f = \overline{D} + B\overline{C} + A$

2. Minimize a expressão lógica dada a seguir (figura 2.31).

$f = \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + AB\overline{C}\,\overline{D} + + AB\overline{C}D + ABC\overline{D}$

Figura 2.31
Representação com os quatro enlaces.



Solução:

Expressão lógica minimizada:

$F(A,B,C,D) = B\overline{C}D + A\overline{D} + \overline{A}CD + BC\overline{D}$

Exemplo para cinco variáveis de entrada

Considere as figuras 2.32 e 2.33.

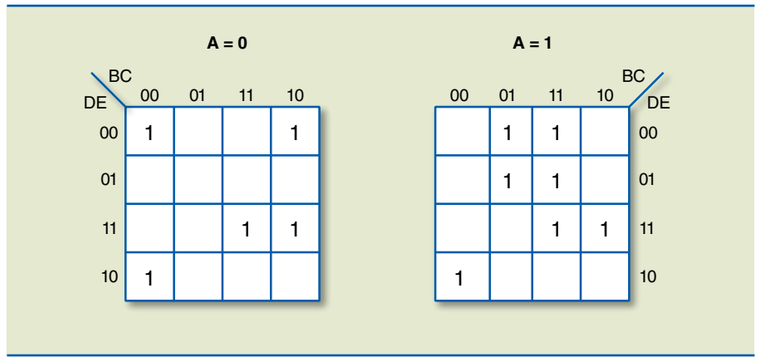


Figura 2.32
Mapa para cinco variáveis de entrada.

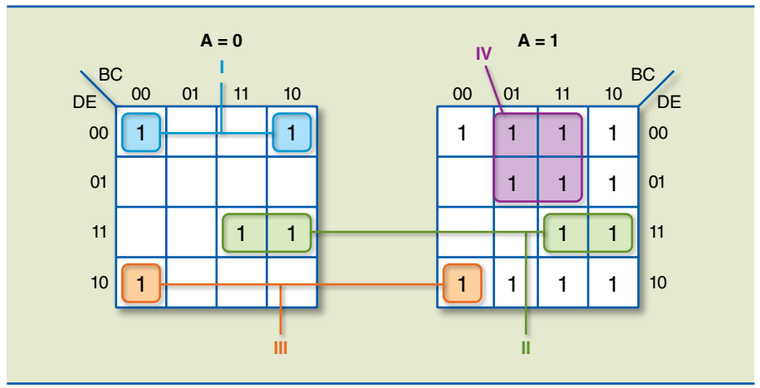


Figura 2.33
Representação dos quatro enlaces.

O resultado obtido é:

$$f = \overline{A}\,\overline{C}\,\overline{D}\,\overline{E} + BDE + \overline{B}\,\overline{C}\,D\,\overline{E} + AC\,\overline{D}$$

Exemplo para seis variáveis de entrada

Considere as figuras 2.34 e 2.35.

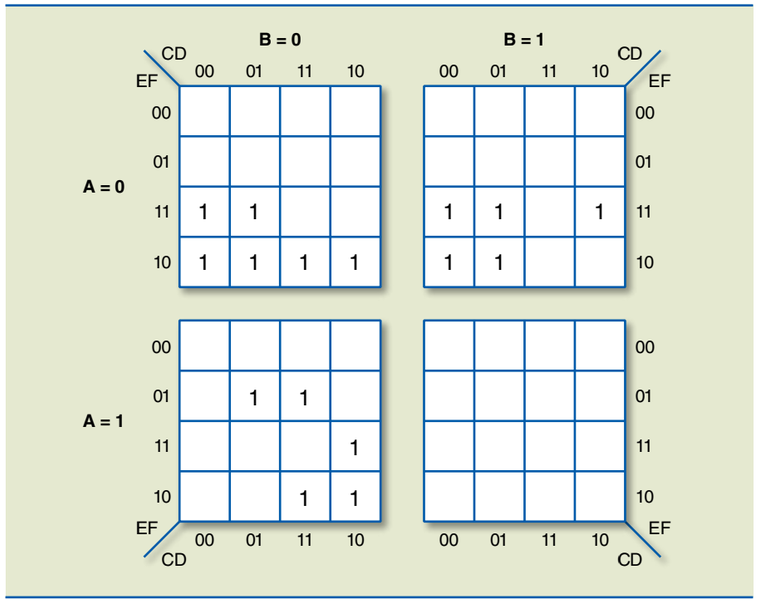


Figura 2.34
Mapa para seis variáveis de entrada.

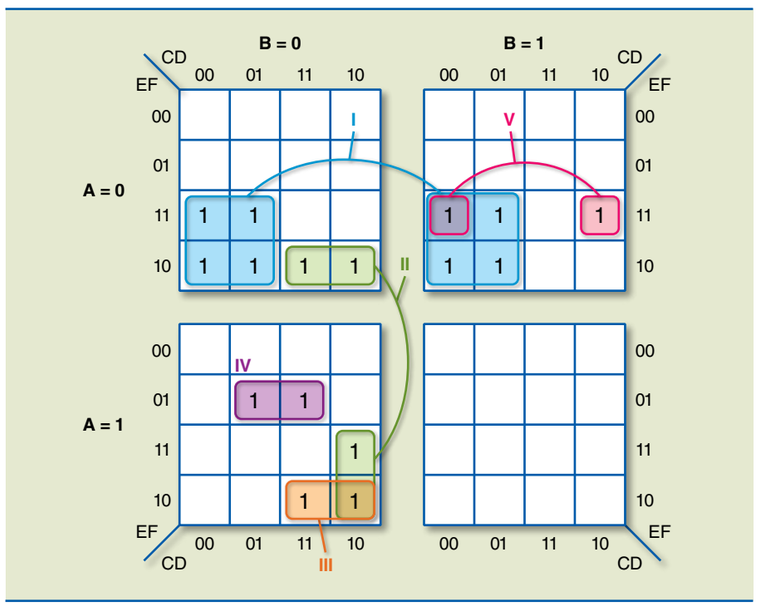


Figura 2.35
Representação dos cinco enlaces.

$$F = \overline{A}\,\overline{C}E + \overline{B}CE\overline{F} + A\overline{B}C\overline{D}E + A\overline{B}D\overline{E}F + \overline{A}B\overline{D}EF$$

Capítulo 3

Circuitos combinatórios

Circuitos combinatórios são aqueles cujas saídas dependem apenas da combinação dos valores das entradas em determinado instante. Neste capítulo serão vistos os principais circuitos combinatórios utilizados em sistemas digitais: codificadores, decodificadores, multiplexadores, demultiplexadores e circuitos aritméticos.

3.1 Codificadores/decodificadores

Os sistemas digitais trabalham com informações representadas por níveis lógicos zeros (0) e uns (1), conhecidos como bits (*binary digits*, ou dígitos binários). Portanto, todas as informações correspondentes a sinais de som, vídeo e teclado (números e letras), por exemplo, devem ser convertidas em bits para que sejam processadas por um sistema digital. Devido ao número de códigos diferentes criados para a representação de grandezas digitais, fez-se necessário desenvolver circuitos eletrônicos capazes de converter um código em outro, conforme a aplicação.

Um codificador é um circuito lógico que converte um conjunto de sinais de entrada em determinado código, adequado ao processamento digital.

3.1.1 Codificador de M-N (M entradas e N saídas)

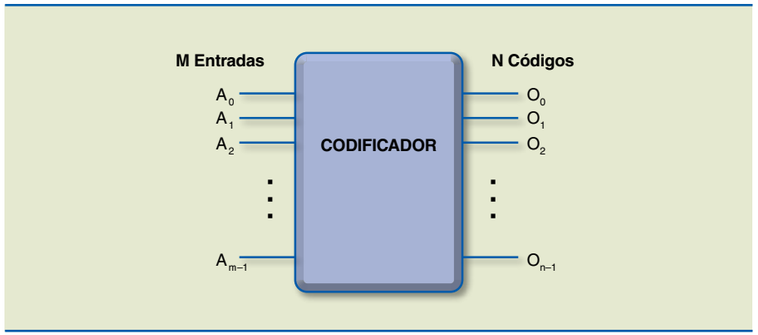


Figura 3.1
Codificador M entradas e N saídas.

3.1.2 Exemplo de codificador decimal-binário

Um codificador decimal para binário possui dez entradas e quatro saídas. A qualquer momento, somente uma linha de entrada tem um valor igual a 1.

Por exemplo, acionando a tecla 6 ($A_6 = 1$), teremos o binário de saída 0110, ou seja, $S_3 = 0$, $S_2 = 1$, $S_1 = 1$ e $S_0 = 0$ (figura 3.2).

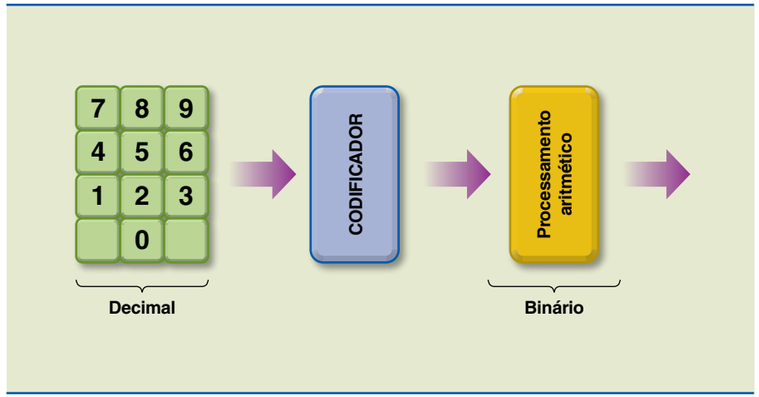


Figura 3.2
Codificador decimal-binário.

O diagrama em blocos do codificador pode ser representado conforme a figura 3.3.

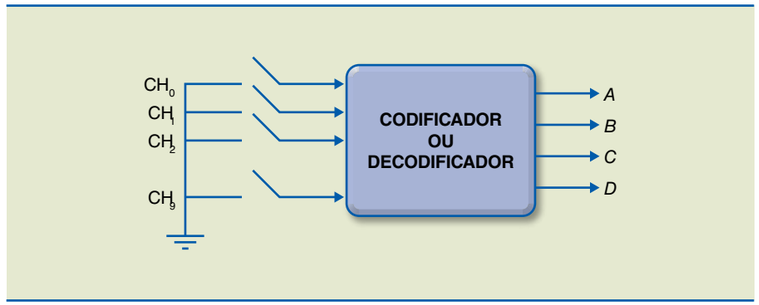


Figura 3.3
Diagrama em blocos do codificador.

Para esse codificador, temos a tabela verdade reproduzida a seguir:

CH_0	CH_1	CH_2	CH_3	CH_4	CH_5	CH_6	CH_7	CH_8	CH_9	A	B	C	D
0	1	1	1	1	1	1	1	1	1	*0*	*0*	*0*	*0*
1	0	1	1	1	1	1	1	1	1	*0*	*0*	*0*	*1*
1	1	0	1	1	1	1	1	1	1	*0*	*0*	*1*	*0*
1	1	1	0	1	1	1	1	1	1	*0*	*1*	*1*	*1*
1	1	1	1	0	1	1	1	1	1	*0*	*1*	*0*	*0*
1	1	1	1	1	0	1	1	1	1	*0*	*1*	*0*	*1*
1	1	1	1	1	1	0	1	1	1	*0*	*1*	*1*	*0*
1	1	1	1	1	1	1	0	1	1	*0*	*1*	*1*	*1*
1	1	1	1	1	1	1	1	0	1	*1*	*0*	*0*	*0*
1	1	1	1	1	1	1	1	1	0	*1*	*0*	*0*	*1*

Codificador com prioridade

Se observarmos com cuidado o circuito do codificador apresentado na figura 3.3, reconheceremos as seguintes limitações: se mais do que duas entradas forem ativadas simultaneamente, a saída será imprevisível ou então não aquela que esperávamos. Essa ambiguidade é resolvida estabelecendo uma prioridade de modo que apenas uma entrada seja codificada, não importando quantas estejam ativas em determinado instante.

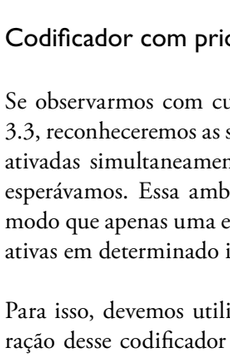
Para isso, devemos utilizar um codificador com função de prioridade. A operação desse codificador é tal que, se duas ou mais entradas forem ativadas ao mesmo tempo, a entrada que tem a prioridade mais elevada terá precedência.

Exemplo de circuito integrado 74147: codificador com prioridade decimal-BCD

A figura 3.4 identifica os pinos do CI 74147 e a tabela verdade correspondente.

Tabela verdade

Figura 3.4
Circuito integrado 74147: codificador com prioridade decimal-BCD.



Entradas									Saídas			
1	2	3	4	5	6	7	8	9	D	C	B	A
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	1	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

Observando a tabela verdade do circuito integrado da figura 3.4, concluímos que nove linhas de entrada ativas (ativas em nível baixo) representam os números decimais de 1 a 9. A saída do CI sugerido é o código BCD invertido, correspondente à entrada de maior prioridade. Caso todas as entradas estejam inativas (inativas em nível alto), então as saídas estarão todas em nível alto. As saídas ficam normalmente em nível alto quando nenhuma entrada está ativa (figura 3.5).

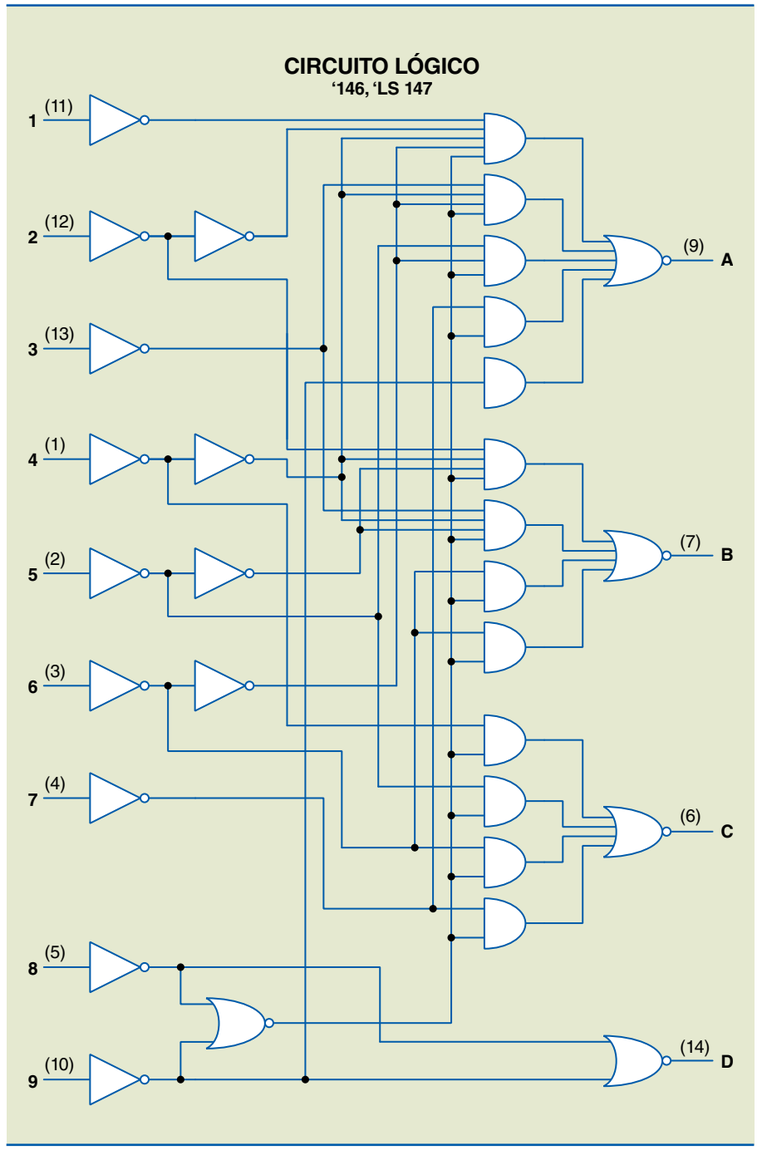


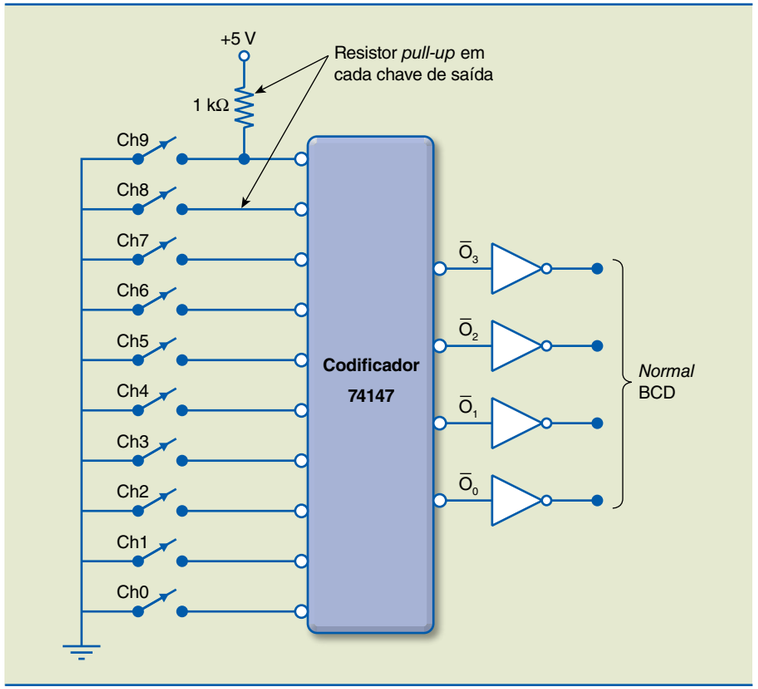
Figura 3.5
Circuito lógico: configuração das portas lógicas do circuito integrado da figura 3.4.

Exemplo de aplicação do CI 74147 em um teclado

Se as chaves estiverem abertas, todas as entradas estarão em nível alto e as saídas em 0000. Se qualquer chave estiver fechada, a entrada correspondente estará em nível baixo e as saídas assumirão o valor do código BCD do número da chave.

O CI 74147 é um exemplo de circuito com prioridade. Dessa maneira, a saída ativa será relativa à chave de maior prioridade entre aquelas que estiverem fechadas em determinado momento (figura 3.6).

Figura 3.6
Exemplo de aplicação do CI 74147.

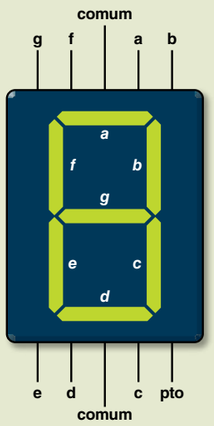


O decodificador também é um circuito combinacional, normalmente usado para habilitar uma, e somente uma, dentre *m* saídas por vez, quando aplicada uma combinação binária específica em suas *n* entradas.

Exemplo de decodificador HEX/BCD – sete segmentos

O *display* de sete segmentos apresenta sete LEDs dispostos de modo que se observe uma estrutura em forma de oito, conforme mostra a figura 3.7.

Figura 3.7
Display de sete segmentos.



Quando queremos, por exemplo, acender o número "0", polarizamos diretamente os *LEDs* (segmentos) que formam o dígito "0" no *display*, ou seja, os segmentos *a*, *b*, *c*, *d*, *e*, *f*, para ser possível visualizar o dígito, conforme ilustrado na figura 3.8.

Figura 3.8
Representação do ***LED*** indicando o número zero.

Para acionar adequadamente o *display* de sete segmentos a fim de visualizarmos o código hexadecimal, é necessário um decodificador com as características apresentadas na figura 3.9 e na tabela verdade correspondente.

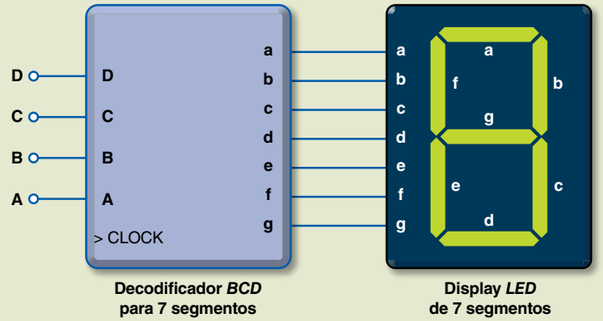
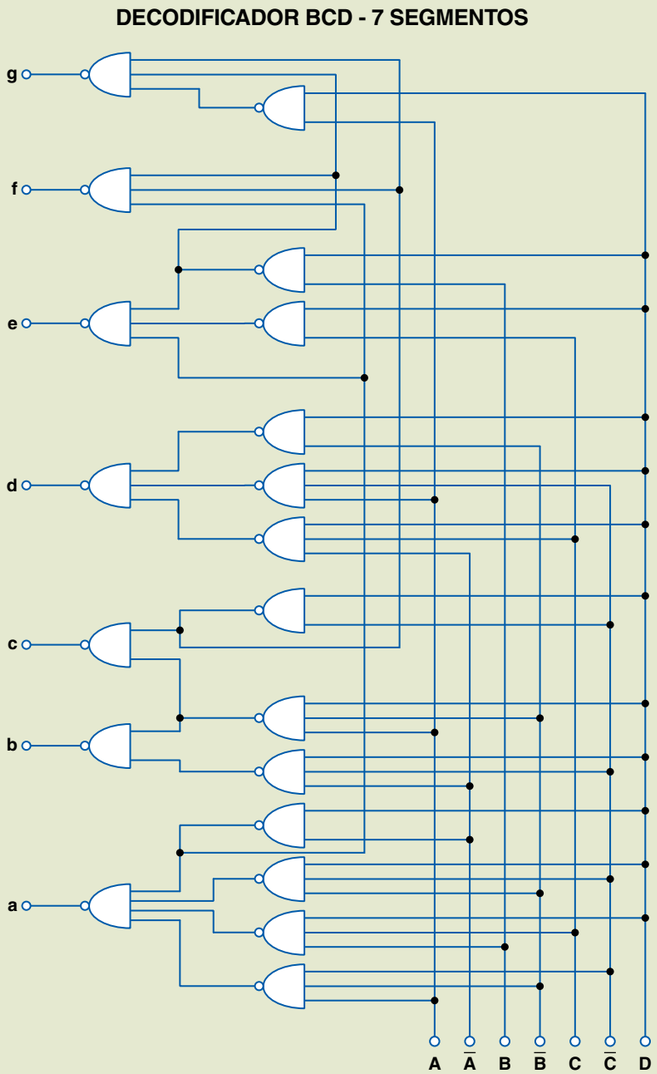


Figura 3.9
Representação do ***display*** e tabela verdade para cada um dos segmentos.

Binary Inputs				Decoder Outputs							Segment Display Outputs
D	C	B	A	a	b	c	d	e	f	g	
Ø	Ø	Ø	Ø	1	1	1	1	1	1	Ø	0
Ø	Ø	Ø	1	Ø	1	1	Ø	Ø	Ø	Ø	1
Ø	Ø	1	Ø	1	1	Ø	1	1	Ø	1	2
Ø	Ø	1	1	1	1	1	1	Ø	Ø	1	3
Ø	1	Ø	Ø	Ø	1	1	Ø	Ø	1	Ø	4
Ø	1	Ø	1	1	Ø	1	1	Ø	1	1	5
Ø	1	1	Ø	1	Ø	1	1	1	1	1	6
Ø	1	1	1	1	1	1	Ø	Ø	Ø	Ø	7
1	Ø	Ø	Ø	1	1	1	1	1	1	1	8
1	Ø	Ø	1	1	1	1	1	Ø	1	1	9

Resolvendo os diagramas de Karnaugh correspondentes aos sete segmentos, obtemos o circuito lógico conforme mostra a figura 3.10.

Figura 3.10
Representação do circuito lógico do decodificador de sete segmentos.



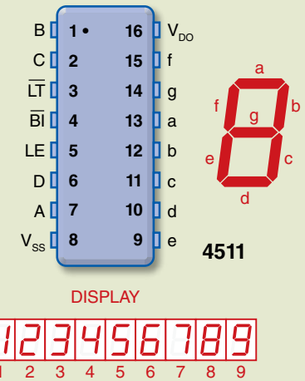
Exemplo de decodificador BCD – sete segmentos

A maior parte das aplicações com *displays* requer que trabalhemos com o código decimal (BCD). Uma possibilidade é utilizar o CI 4511, que é um decodificador BCD – 7 segmentos. A figura 3.11 mostra a representação dos pinos desse circuito e a tabela verdade detalhada.

Figura 3.11
Representação dos pinos do CI 4511 e tabela verdade para cada um dos segmentos.

Tabela verdade

Entradas BCD				Saídas							*Display* tipo cátodo comum
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9



Entrada D = MSB e entrada A = LSB

Para os códigos binários correspondentes aos dígitos maiores do que 9 (1010 até 1111), todas as saídas são colocadas em nível "0" e, consequentemente, todos os segmentos do *display* ficam apagados (função *blank*).

Sinais de controle

Para visualizarmos os códigos, conectamos as entradas LT (*lamp test*) e BI (*ripple blanking input*) em nível lógico "1" e a entrada LE (*latch enable*) em nível lógico "0". Para testarmos os segmentos do *display*, conectamos a entrada LT em nível lógico "0" (todos os segmentos do *display* deverão acender, independentemente do código presente nas entradas D, C, B e A).

A entrada LE pode ser utilizada (quando em nível lógico "1") para armazenar o código presente nas entradas BCD. O *display* permanecerá inalterado até que se aplique nível lógico "0" na entrada LE para um novo código presente nas entradas BCD.

Conexões externas

O diagrama da figura 3.12 ilustra a utilização do CI com *display* de sete segmentos cátodo comum.

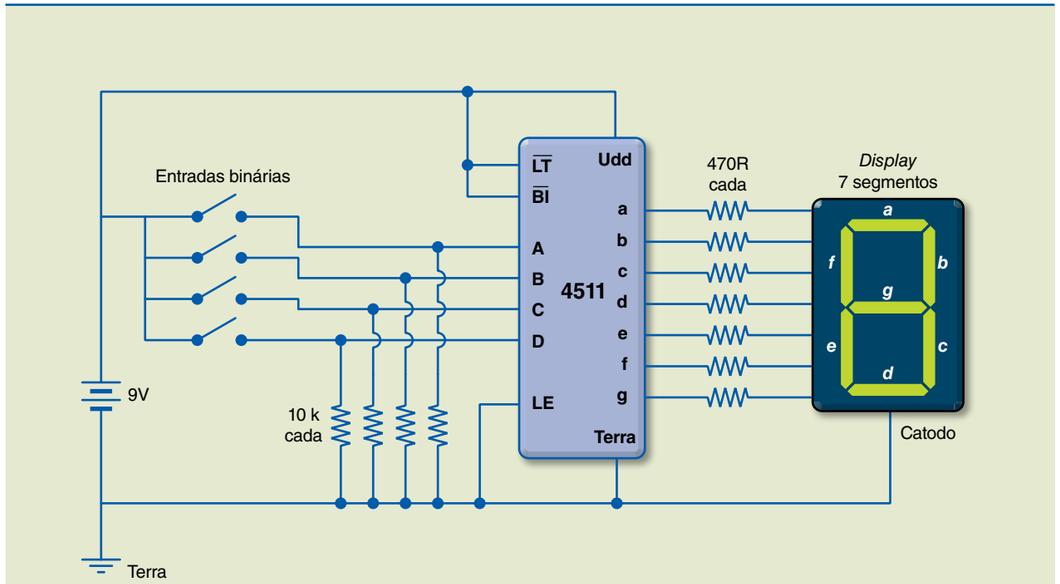


Figura 3.12
CI com ***display*** de sete segmentos cátodo comum.

3.2 Multiplexadores/demultiplexadores

Consideremos a seguinte situação: queremos transferir dados lógicos ("0", "1") de quatro entradas para oito saídas, com a possibilidade de qualquer entrada se comunicar com qualquer saída, tendo para isso uma única via de transferência de dados (figura 3.13).

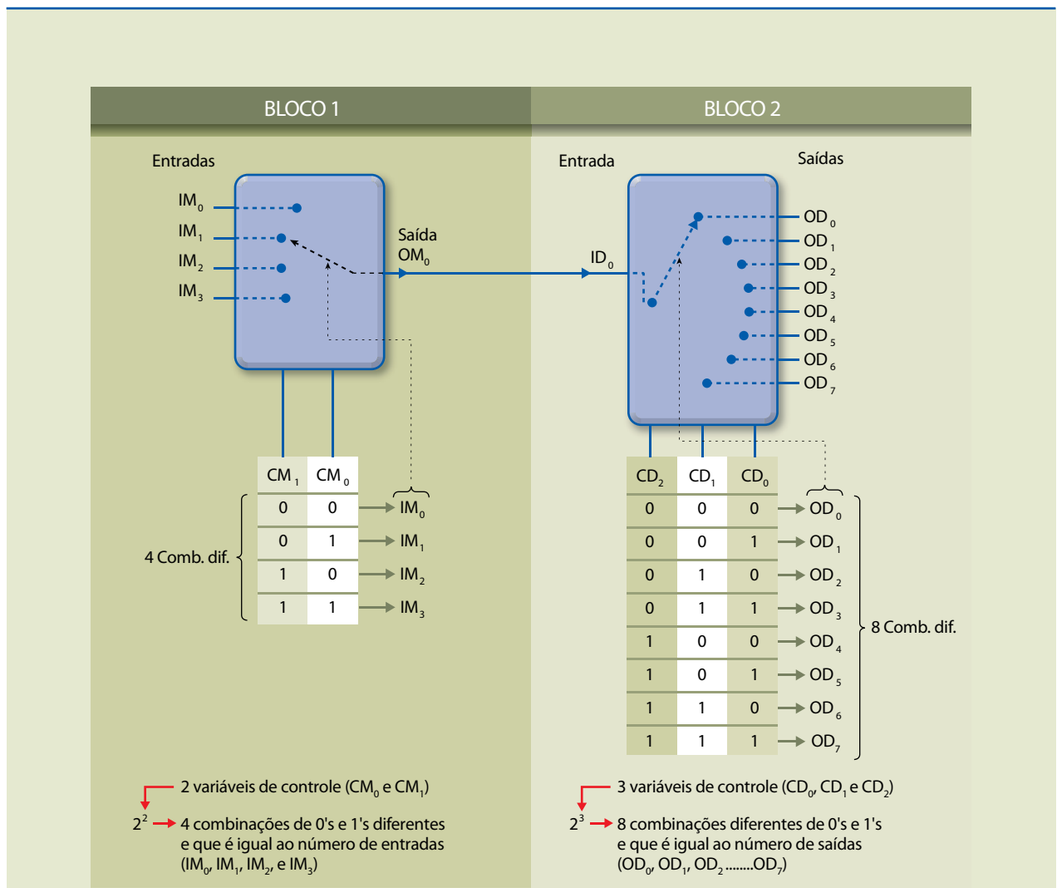


Figura 3.13
Transferência de dados entre os blocos 1 e 2.

Na figura 3.13, o bloco 1 apresenta a ideia básica de um multiplexador (MUX), ou seja, de várias entradas, uma é selecionada e direcionada para a saída. A seleção é representada na figura por uma chave; no circuito real, a seleção é feita por meio das variáveis de controle (seleção). Nesse exemplo, o multiplexador tem quatro entradas (IM_0, IM_1, IM_2, IM_3) e, portanto, precisamos de duas variáveis de controle, pois é possível com elas obter quatro combinações de "0" e "1" diferentes.

O bloco 2 apresenta a ideia básica de um demultiplexador (DEMUX), ou seja, a entrada única de dados é direcionada para uma das várias saídas, para a saída selecionada.

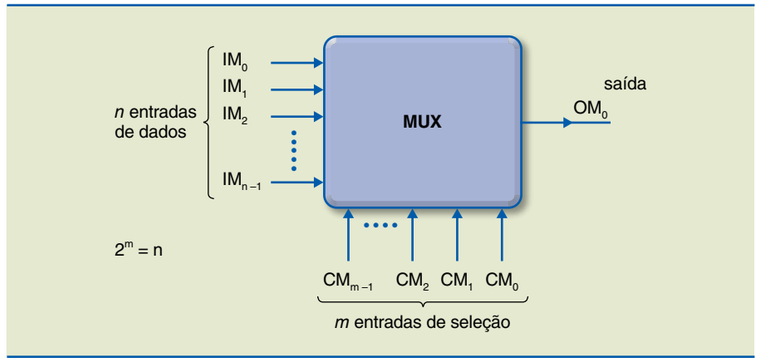
A tabela 3.5 registra, em cada linha, o "caminho" de determinada entrada até certa saída por meio das variáveis de controle de entrada no MUX e das variáveis de controle de saída no DEMUX.

Tabela 3.1
Tabela verdade

ENTRADA						
DADOS	CONTROLE					
	MUX		DEMUX			SAÍDA
	CM_1	CM_0	CD_2	CD_1	CD_0	
IM_2	1	0	0	1	1	OD_3
IM_0	0	0	1	1	0	OD_6
IM_2	1	0	1	1	0	OD_6
IM_3	1	1	0	0	1	OD_1
IM_1	0	1	1	1	1	OD_7
IM_0	0	0	1	0	0	OD_4
IM_0	0	0	0	1	0	OD_2
IM_1	0	1	1	0	1	OD_5T

A figura 3.14 representa um multiplexador de *n* entradas de dados, *m* entradas de controle (seleção) e uma saída.

Figura 3.14
Multiplexador.



Vamos implementar um MUX de oito entradas. Para isso, necessitamos de três variáveis de controle, pois $2^3 = 8$, que corresponde ao número de entradas (tabela verdade).

Variáveis de controle			Saída	Produtos das variáveis de controle
CM_2	CM_1	CM_0	OM_0	
0	0	0	IM_0	$CM_2.CM_1.CM_0$
0	0	1	IM_1	$CM_2.CM_1.CM_0$
0	1	0	IM_2	$CM_2.CM_1.CM_0$
0	1	1	IM_3	$CM_2.CM_1.CM_0$
1	0	0	IM_4	$CM_2.CM_1.CM_0$
1	0	1	IM_5	$CM_2.CM_1.CM_0$
1	1	0	IM_6	$CM_2.CM_1.CM_0$
1	1	1	IM_7	$CM_2.CM_1.CM_0$

Tabela 3.2
Tabela verdade

Observe na tabela verdade que a coluna "Saída" corresponde às entradas selecionadas pelas variáveis de controle, como deve ocorrer em um MUX, ou seja, $OM_0 = IM$ selecionada.

Sabemos que, se todas as entradas de uma porta E forem "1", exceto uma, que poderá ser "1" ou "0", a saída da porta será "1" ou "0". Então, temos, por exemplo, a figura 3.15.

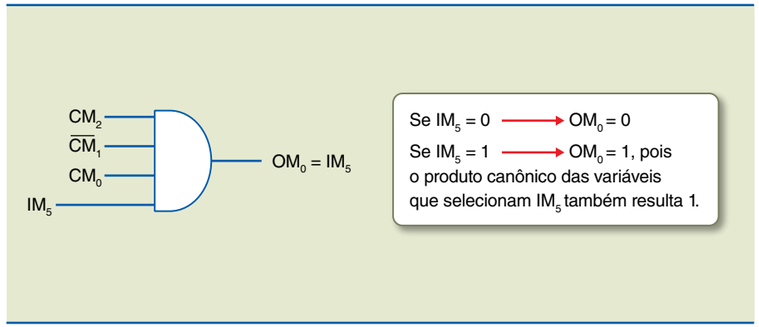
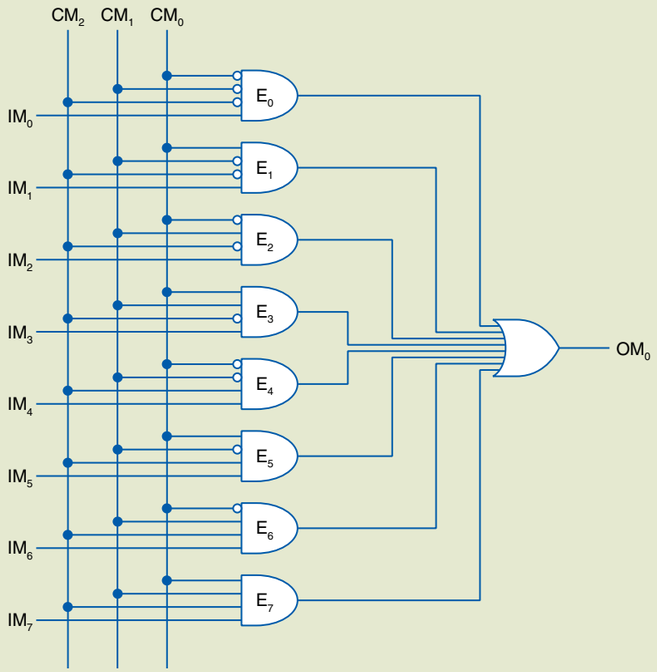


Figura 3.15
Porta E com três entradas.

Assim, podemos implementar o MUX de oito entradas e três variáveis de controle como apresentado na figura 3.16.

Figura 3.16
MUX de oito entradas e três variáveis.



A "bolinha" (○) indica que a entrada foi complementada, substituindo, na representação, a porta inversora.

Podemos implementar o MUX por meio da tabela verdade. Para isso, devemos considerar que a tabela verdade terá como entrada oito variáveis de dados e três variáveis de controle – assim, em princípio, uma tabela verdade com 2^{11} = 2 048 combinações.

Somente as linhas em que a variável de dados selecionada é "1", a saída é 1 e essa condição independe das demais variáveis de dados. Para isso, temos de levar em consideração oito linhas das 2 048, e, portanto, a função booleana de saída é a soma do produto dessas oito linhas:

$$OM_0 = IM_0 \cdot \overline{CM_2} \cdot \overline{CM_1} \cdot \overline{CM_0} + IM_1 \cdot \overline{CM_2} \cdot \overline{CM_1} \cdot CM_0 + IM_2 \cdot \overline{CM_2} \cdot CM_1 \cdot \overline{CM_0} + IM_3 \cdot \overline{CM_2} \cdot CM_1 \cdot \underline{CM_0} + IM_4 \cdot CM_2 \cdot \overline{CM_1} \cdot \overline{CM_0} + IM_5 \cdot CM_2 \cdot \overline{CM_1} \cdot CM_0 + IM_6 \cdot CM_2 \cdot CM_1 \cdot \overline{CM_0} + IM_7 \cdot CM_2 \cdot CM_1 \cdot CM_0$$

Essa função booleana é executada pelo circuito da figura 3.16 (oito portas E e uma porta OU).

É possível implementar funções lógicas diretamente em um multiplexador. Os exemplos a seguir ilustram essa técnica.

Exemplos

1. Seja a função $y = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C$.

Escolhemos um multiplexador com três entradas de controle (seleção), pois a função possui três variáveis independentes (figura 3.17). Fazemos uma tabela verdade, relacionando as variáveis de controle e as de dados.

Tabela verdade
$y = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C$

A	B	C	y	Dados
0	0	0		$I_0 = 0$
0	0	1	$\overline{A} \cdot \overline{B} \cdot C$	$I_1 = 1$
0	1	0		$I_2 = 0$
0	1	1		$I_3 = 0$
1	0	0	$A \cdot \overline{B} \cdot \overline{C}$	$I_4 = 1$
1	0	1	$A \cdot \overline{B} \cdot C$	$I_5 = 1$
1	1	0		$I_6 = 0$
1	1	1		$I_7 = 0$

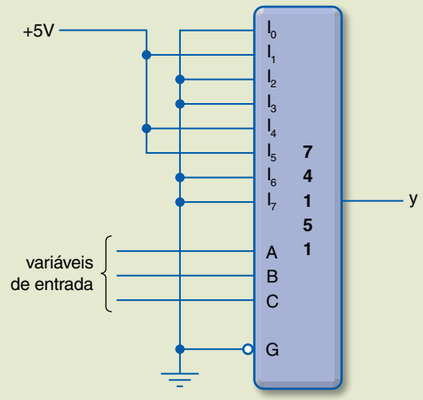


Figura 3.17
Multiplexador com três entradas de controle e tabela verdade correspondente.

As variáveis de dados I_1, I_4, e I_5 são levadas para nível "1", pois correspondem às entradas do MUX que são selecionadas pelas variáveis de controle e que aparecerão na saída conforme estabelecido pela função. As demais variáveis são levadas para o nível "0". As variáveis de controle são as dependentes da função booleana. A variável independente é representada pela saída do multiplexador.

Para implementarmos o circuito da figura 3.17, podemos usar o CI TTL 74151 – multiplexador digital de oito canais (figura 3.18).

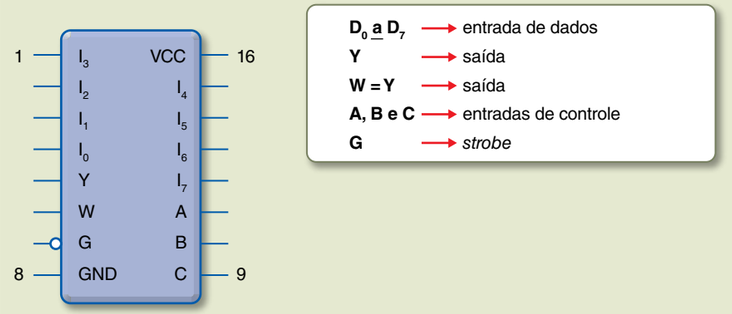


Figura 3.18
Pinagem do CI TTL 74151 – multiplexador digital de oito canais (16 pinos).

Analisando a figura 3.18, temos:

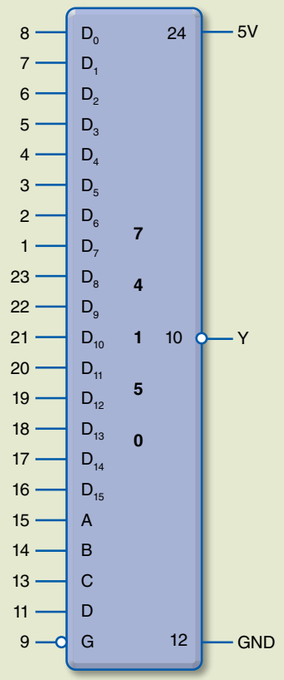
- Y apresenta o valor da variável selecionada;
- W = Y; G é ativo em nível baixo (indicado com a "bolinha" na representação da figura), o que significa que em G = 0 o MUX está liberado para funcionamento normal; para G = 1, Y = 0 independentemente dos valores das entradas A, B e C.

Agora, vamos analisar o CI TTL 74150 (figura 3.19) – multiplexador digital de 16 canais (24 pinos) – e a tabela verdade correspondente.

Figura 3.19
Pinagem do CI TTL 74150 e tabela verdade correspondente.

Tabela verdade

strobe	seleção				saída
G	A	B	C	D	$\overline{Y}$
0	0	0	0	0	D_0
0	0	0	0	1	D_1
0	0	0	1	0	D_2
0	0	0	1	1	D_3
0	0	1	0	0	D_4
0	0	1	0	1	D_5
0	0	1	1	0	D_6
0	0	1	1	1	D_7
0	1	0	0	0	D_8
0	1	0	0	1	D_9
0	1	0	1	0	D_{10}
0	1	0	1	1	D_{11}
0	1	1	0	0	D_{12}
0	1	1	0	1	D_{13}
0	1	1	1	0	D_{14}
0	1	1	1	1	D_{15}
1	X	X	X	X	0



Analisando a figura 3.19, temos:

- a saída Y é complemento da entrada selecionada (ver representação – tem "bolinha");
- o *strobe* é ativo em 0 (ver representação – tem "bolinha");
- G = 1 → Y = 0, independentemente de A, B, C e D.

2. Seja, na figura 3.20, a função $y = A \cdot B \cdot C + A \cdot \overline{B \cdot C} + A \cdot B \cdot \overline{C}$. A tabela verdade representa a função utilizada.

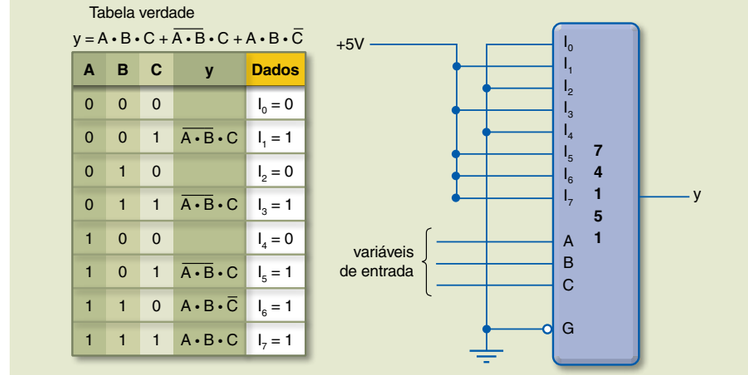


Figura 3.20
Pinagem do CI 74151 referentes à função utilizada e a tabela verdade correspondente.

Pela associação de multiplexadores, é possível aumentar o número de entradas do circuito original, conforme mostra a figura 3.21, e montar um multiplexador de 16 canais utilizando multiplexadores de oito canais cada. Para isso, vamos utilizar o CI 74151, que já conhecemos.

Figura 3.21
Associação de multiplexadores utilizando CI 74151.

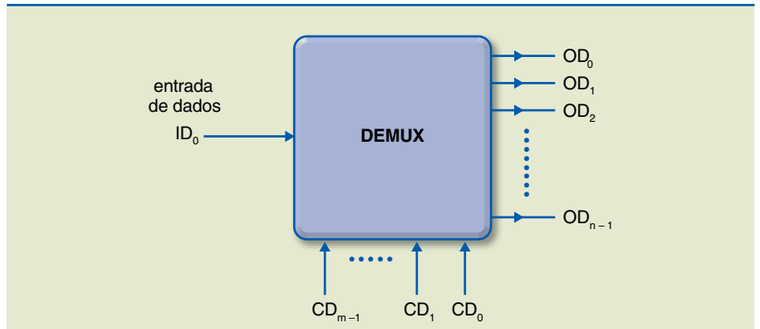
Analisando a figura 3.21, podemos notar que D é o bit MSB (bit mais significativo) dos bits de seleção. Assim, temos como exemplos dois endereços:

D A B C
0 1 0 1 → IF_5 D = 0 seleciona as entradas IF_0 a IF_7
1 0 1 1 → IF_{11} D = 1 seleciona as entradas IF_8 a IF_{15}

O demultiplexador realiza a função inversa do multiplexador, ou seja, a informação recebida em uma única entrada de dados é enviada para uma saída selecionada por variáveis de controle (seleção).

O demultiplexador representado na figura 3.22 tem m entradas de controle e n saídas.

Figura 3.22
Representação do DEMUX.



Vamos implementar um DEMUX de oito saídas. Para isso, necessitamos de três variáveis de controle, pois $2^3 = 8$, que corresponde ao número de saídas. Como são oito saídas, há oito tabelas verdades, que podem ser montadas em uma só com as mesmas entradas e as respectivas saídas.

Entradas				Saídas							
CD_2	CD_1	CD_0	ID_0	OD_7	OD_6	OD_5	OD_4	OD_3	OD_2	OD_1	OD_0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	I	I	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...

Tabela 3.3
Tabela verdade para um DEMUX de oito saídas (representação parcial)

Analisando cada saída, sem a necessidade de montar a tabela verdade completa, concluímos que ela somente será "1" se a entrada de dados for "1", uma vez que o produto canônico correspondente a essa saída será "1". Qualquer outra condição levará a saída para "0". A figura 3.23 apresenta um exemplo.

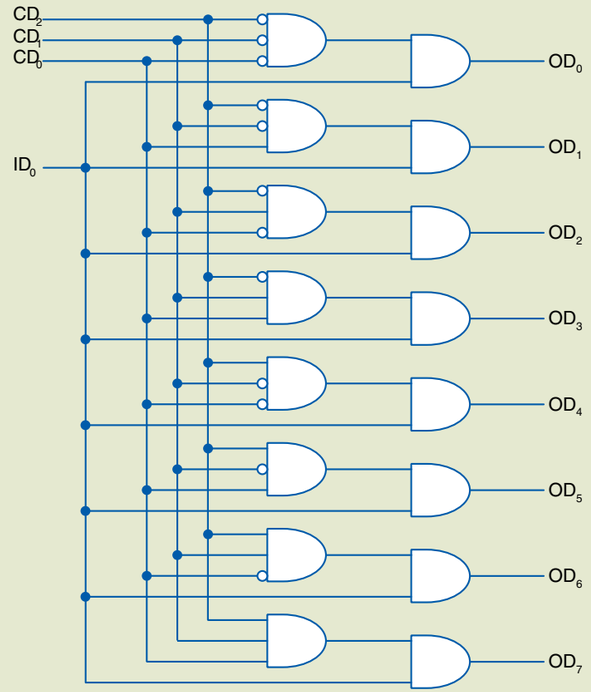
$$OD_3 = \underbrace{ID_0 \cdot CD_0 \cdot CD_1 \cdot \overline{CD}_2}$$

produto canônico das variáveis de controle para seleção de OD_3

Figura 3.23
Exemplo para análise da condição estabelecida no DEMUX de oito saídas.

Com a expressão booleana de cada saída obtida de maneira semelhante, podemos implementar o circuito do DEMUX com portas lógicas (figura 3.24).

Figura 3.24



Entradas						Saídas															
G1	G2	D	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}	Y_{13}	Y_{14}	Y_{15}
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Tabela 3.4
Tabela verdade (CI 74154)

Observando a tabela verdade da figura 3.24, podemos notar que as duas entradas *strobe* G1 e G2 são ativas em nível baixo, e, para seu funcionamento normal, elas devem estar em nível baixo. Se G1 e G2 não estiverem em nível baixo, todas as saídas vão para nível alto. Observe que, em funcionamento normal, somente a saída selecionada está em nível baixo; as demais encontram-se em nível alto.

Vamos usar o CI 74154 (figura 3.25) para executar a função.

$$y = A\cdot\overline{B}\cdot C\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot D + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D} + A\cdot B\cdot C\cdot D$$

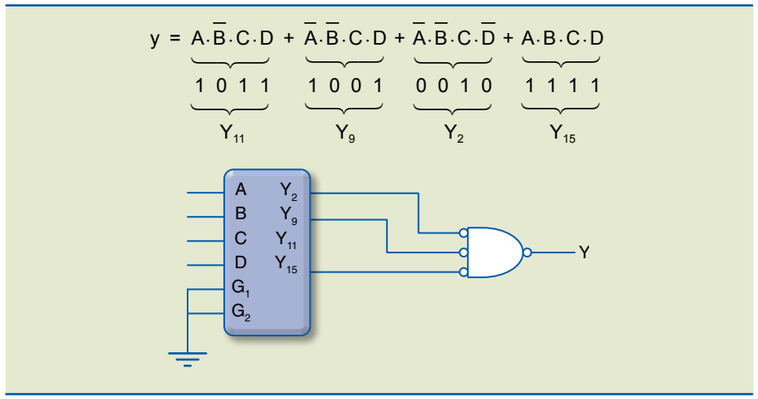


Figura 3.25

Analisando a figura 3.25, podemos perceber que, como Y é saída de uma porta NE, se uma das entradas for "0", Y será igual a "1". Isso só acontece se uma das saídas corresponder a um dos termos da função booleana de Y, selecionada pelas variáveis de controle (A, B, C, D).

Da mesma forma como foi feito com os multiplexadores, é possível a combinação de demultiplexadores para aumentar a capacidade do circuito, conforme exemplo da figura 3.26. Utilizando o 74154, vamos montar um demultiplexador de 32 saídas.

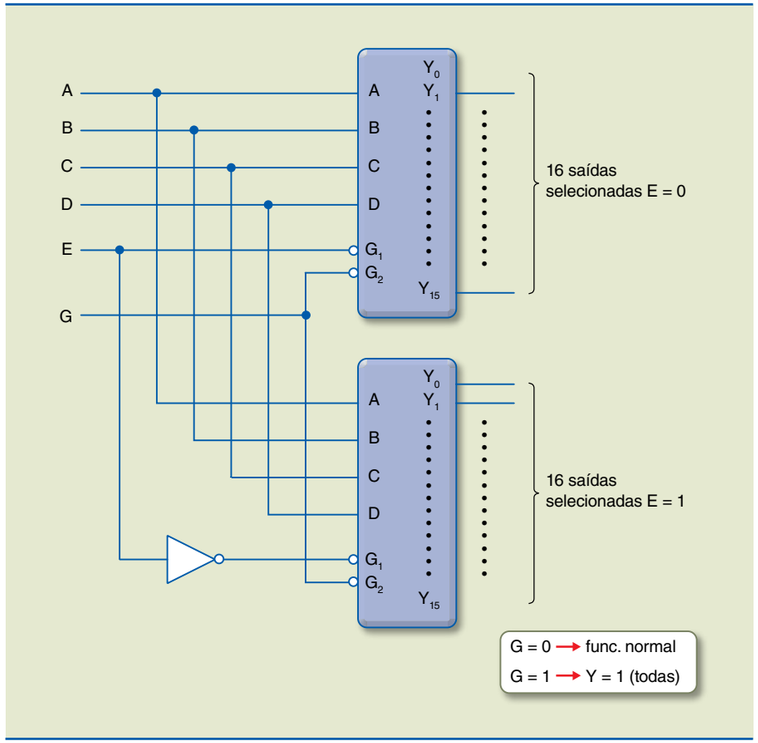
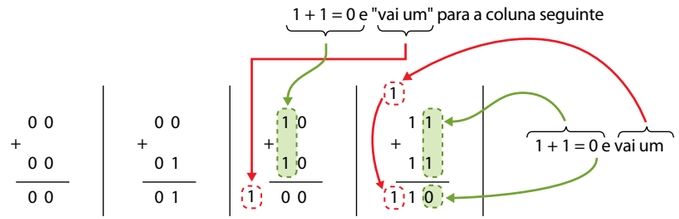


Figura 3.26
DEMUX de 32 saídas.

3.3 Circuitos aritméticos

O microprocessador, componente fundamental de um computador, tem em sua arquitetura interna uma ULA (unidade lógica aritmética), na qual são realizadas as operações lógicas e aritméticas. Associando portas lógicas de maneira conveniente, podemos obter circuitos que realizam operações aritméticas. Devemos lembrar que **portas lógicas** têm como entrada **estados lógicos** que foram associados aos símbolos “0” e “1”, e **circuitos aritméticos** têm como entrada **números**.

A adição, a subtração e a multiplicação de números binários e decimais são efetuadas de modo semelhante, lembrando que o “vai um” em binário ocorre quando a soma dos dígitos é 2 e não 10 como em decimal. Por exemplo:



Agora, vamos calcular:

B_1 = (0101 0011 + 0110 1001) e B_2 = (0101 1101 + 1000 1110):

$$\begin{array}{rcl} 1 \quad 1 & \leftarrow \text{ os “vai um” } \rightarrow & 1 \quad 1 \\ 0101 \quad 0011 & & 0101 \quad 1101 \\ + & & + \\ \underline{0110 \quad 1001} & & \underline{1000 \quad 1110} \\ 1011 \quad 1100 & & 1110 \quad 1011 \end{array}$$

$$B_1 = 1011\ 1100 \qquad B_2 = 1110\ 1011$$

Os microprocessadores não possuem circuitos de subtração, porém essa operação pode ser realizada por meio da adição usando números na forma complemento 2. Para isso, vamos primeiro considerar ,por exemplo, o cálculo de $X = 85 - 37$, ou seja, uma subtração em decimal.

Somando 100 e subtraindo 100 do segundo membro da equação, o valor de X não se altera. Portanto, temos $X = 85 - 37 + (100 - 100) = 85 + (100 - 37) - 100 = 85 + 63 - 100$. O valor $(100 - 37)$ poderia ser obtido complementando os algarismos de 37 para o valor 9 e somando 1, resultando $(62 + 1) = 63$. Assim, temos $X = (85 + 63) - 100 = 148 - 100$. Nesse caso, subtrair 100 equivale a desprezar o último algarismo do 148, resultando $X = 48$, que é o resultado procurado.

Vamos analisar outro exemplo com números decimais, aplicando, agora, a regra usada no exemplo anterior generalizada: $X = 743 - 571$. Somamos ao minuendo o complemento 9 do subtraendo mais 1 e desprezamos o último dígito à esquerda, resultando $X = 743 + (428 + 1) = 1172$. Assim, temos $X = 172$ e chegamos ao resultado correto fazendo um cálculo diferente do usual.

Vamos retornar ao primeiro exemplo:

$X = 85 - 37$ (estamos **subtraindo** do número positivo 85 o número positivo 37) é equivalente a

$X = 85 + (-37)$ (estamos **somando** ao número positivo 85 o número negativo –37)

Observe que, ao desenvolvermos o cálculo no exemplo, tínhamos chegado a $X = 85 + 63$ e desprezamos o último algarismo da esquerda. Comparando $X = 85 + (-37)$ com $X = 85 + 63$ (desprezando o último algarismo), o número 63 poderia ser interpretado como o negativo de 37, pois o resultado foi igual. Com o mesmo raciocínio, poderíamos interpretar no segundo exemplo o número 429 como o negativo de 571.

No processo usado para obtermos o resultado, a complementação do subtraendo foi feita para 9, ou seja, para o valor da base tirando 1 (sistema decimal 10 – 1 = 9).

Procedimento similar é usado na base 2 para transformar uma operação de subtração em uma adição. No caso de binários, que são base 2, a complementação do subtraendo seria para 1, e complementaríamos o processo somando 1, obtendo, assim, a representação complemento 2 do binário a ser subtraído.

Complementar dígitos binários para 1 não é difícil, uma vez que se trata de circuito numérico correspondente a porta lógica inversora com **entradas numéricas** (0 ou 1). Somar com circuitos digitais também é simples. Portanto, a ideia exemplificada em decimais é usada em sistemas binários. O objetivo é transformar operações de subtração em adição, que é mais fácil de implementar com circuitos digitais.

Em binário, quando é necessário trabalhar com números negativos, o primeiro bit da esquerda é reservado para definição do sinal. Assim, quando trabalhamos com binário com sinal, precisamos saber o número de dígitos com que os números serão apresentados. Os binários negativos têm o primeiro bit da esquerda igual a “1”, e os binários positivos, igual a “0”. Usando esse critério, ou seja, ter bem definida a posição do bit de sinal, podemos representar os binários negativos pelo **complemento 2** de seu valor positivo.

Trabalhando com números de oito bits, temos, por exemplo:

1001 1101 corresponde a um número negativo em representação binária.
0010 0110 corresponde ao decimal 38 positivo.

Vamos representar com oito dígitos, em binário, os números decimais 37, –6, –19 e –97 em representação complemento 2.

Binário (+37) 0 0 1 0 0 1 0 1

Binário (-6)

Binário (+6)	0000 0110
Achamos complemento 1	1111 1001
Somamos 1	0000 0001 +
Binário (-6) →	1111 1010

Binário (-19)

Binário (+19)	0001 0011
Achamos complemento 1	1110 1100
Somamos 1	0000 0001 +
Binário (-19) →	1110 1101

Binário (-97)

Binário (+97)	0110 0001
Achamos complemento 1	1001 1110
Somamos 1	1 +
Binário (-97) →	1001 1111

Qual o valor decimal que corresponde ao binário com sinal B_1 = 1001 1101 representado em complemento 2?

B_1 representa um binário negativo, pois o primeiro dígito à esquerda é 1; portanto, o correspondente decimal do complemento 2 dele com sinal negativo é o valor procurado.

Binário B_1	1001 1101
Complemento 1 de B_1	0110 0010 +
Somando 1	1
Representação complemento 2 de B_1	0110 0011
Convertendo em decimal	$-B_1 = (64 + 32 + 1)_{10} = (-99)_{10}$
Portanto	$B_1 = (-99)_{10}$

A representação complemento 2 de um binário significa o valor negativo do binário, independentemente de ele ser positivo ou negativo.

A tabela 3.5 apresenta alguns números binários com sinal representados com oito dígitos e seu respectivo valor decimal.

Tabela 3.5

decimal	binário	decimal	binário
+ 99	0110 0011	– 6	1111 1010
+ 46	0010 1110	– 19	1110 1101
+ 21	0001 0101	– 21	1110 1011
+ 19	0001 0011	– 99	1001 1101

Observe os cálculos representados em números decimais usando os correspondentes binários e confira os resultados tendo como referência os valores decimais.

Exemplos

1. 21 – 19

Solução:

$$21 - 19 = 21 + (-19)$$

$$(0001\ 0101) - (0001\ 0011) = (0001\ 0101) + (1110\ 1101)$$

Transformamos a subtração em uma adição substituindo –19 pelo complemento 2 de +19, que corresponde ao valor negativo.

1111 1 1	← os “vai um”
0001 0101	
1110 1101 +	
10000 0010	

0 0 0 0 0 0 1 0 = $(2)_{10}$ O nono bit, último à esquerda, é desconsiderado, pois, conforme estabelecido no início deste estudo, estamos trabalhando com binários com sinal representados por oito dígitos. Conferindo a conta em decimal, o resultado bate.

2. 46 – 21

Solução:

$$46 - 21 = 46 + (-21)$$

$$(0010\ 1110) - (0001\ 0101) = (0010\ 1110) + (1110\ 1011)$$

Transformamos a subtração em uma adição substituindo –21 pelo complemento 2 de +21.

```
 1 1   1 1 1     ←  os "vai um"
 0 0 1 0  1 1 1 0
                +
 1 1 1 0  1 0 1 1
-------------------
1 0 0 0 1  1 0 0 1
```

$0\ 0\ 0\ 1\ 1\ 0\ 0\ 1 = (25)_{10}$ O nono bit, último à esquerda, é desconsiderado, pois, conforme estabelecido no início deste estudo, estamos trabalhando com binários com sinal representados por oito dígitos. Conferindo a conta em decimal, o resultado bate.

3. 21 – 99

Solução:

$$21 - 99 = 21 + (-99)$$

$$(0001\ 0101) - (1001\ 1101) = (0001\ 0101) + (1001\ 1101)$$

Transformamos a subtração em uma adição substituindo –99 pelo complemento 2 de +99.

```
   1 1 1   1  ←   os "vai um"
 0 0 0 1  0 1 0 1
                +
 1 0 0 1  1 1 0 1
-------------------
 1 0 1 1  0 0 1 0
```

Como esperávamos, a operação resultou um número negativo. Para encontrarmos esse número em decimal, devemos calcular seu complemento 2, obtendo o binário positivo correspondente, e depois converter em decimal e trocar o sinal.

$$1011\ 0010 \rightarrow \text{complemento 1} \rightarrow 0100\ 1101 + 1 = 0100\ 1110 = (78)_{10}$$

Portanto, o resultado da operação em decimal é –78.

4. Determine o correspondente decimal do binário (1110 1011), em que o primeiro bit da esquerda é o bit de sinal.

Solução:

Determinamos o complemento 2 de (1110 1011), obtendo o valor positivo, convertemos em decimal e trocamos o sinal.

$$1110\ 1011 \rightarrow \text{complemento 1} \rightarrow 0001\ 0100 + 1 = 0001\ 0101 = (+21)_{10}$$

Portanto, o binário (1110 1011) corresponde ao decimal –21.

3.3.1 Meio somador

Vamos observar as tabelas do OU EXCLUSIVO, do E, e da soma aritmética de dois binários de um dígito cada um (um bit).

Porta lógica E e OR EXCLUSIVO

entrada		A ⊕ B	A • B
A	B	S	
0	0	0	**0**
0	1	1	**0**
1	0	1	**0**
1	1	1	**1**

SOMA ARITMÉTICA

entrada		(A + B)		
A	B	C	S	(decimal)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	2

Tabela 3.6
Porta lógica E e OU EXCLUSIVO e SOMA ARITMÉTICA

As colunas assinaladas mostram que podemos usar o circuito da porta OU EXCLUSIVO para obter os dígitos menos significativos da soma aritmética (A + B), tendo como entradas variáveis numéricas binárias A e B. Podemos também usar o circuito da porta lógica E para obter o dígito seguinte, dígito "vai um", tendo como entradas variáveis numéricas binárias A e B (figura 3.27).

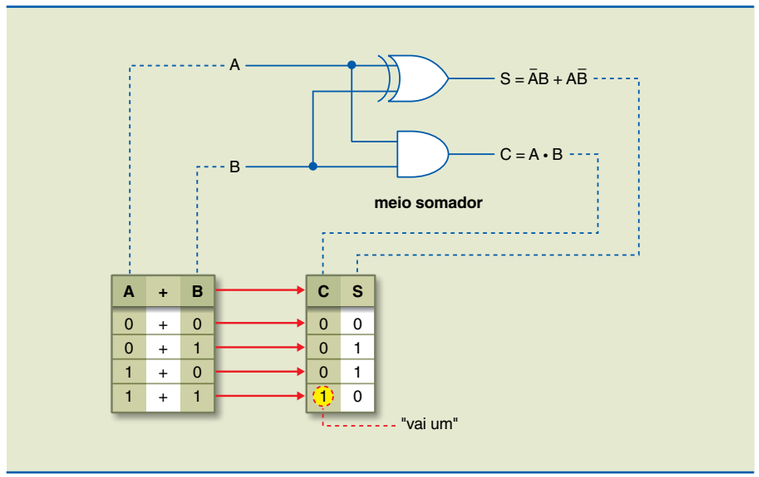
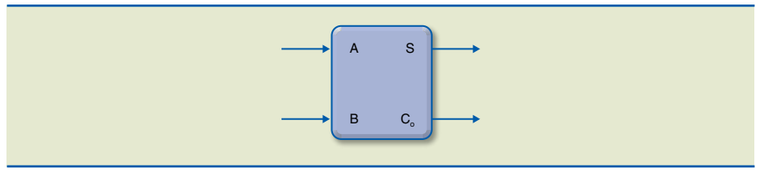


Figura 3.27
Circuito meio somador.

O meio somador é também conhecido como *half adder* (inglês), e o dígito de transporte C, como *carry* (inglês).

Representando o meio somador em um único bloco, temos a figura 3.28.

Figura 3.28
Circuito meio somador.



Não é possível somar mais de dois algarismos com o meio somador. Para atendermos a essa condição, devemos utilizar o somador completo.

3.3.2 Somador completo

Consideremos a soma de dois números binários (1000 1011) + (0101 0111), representada no exemplo:

```
   1 1 1 1 1   →  C
1 0 0 0 1 0 1 1 →  A
0 1 0 1 0 1 1 1 →  B
---------------
1 1 1 0 0 0 1 0 →  S
                   ⌐
                   └→ Meio Somador (Ver Figura 3.27)
```

Os bits da primeira coluna à direita e o "vai um" podem ser obtidos com o meio somador. A partir da segunda coluna, o meio somador não é suficiente, pois há a possibilidade de haver três bits envolvidos na soma caso ocorra "vai um" da coluna anterior. Portanto, precisamos de um circuito aritmético com três entradas e duas saídas.

O circuito com a tabela verdade representada a seguir resolve o problema.

Entradas			Saídas	
A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Na tabela:

- A e B são dígitos dos binários somados;
- C_i é o *carry in*, "vai um" da coluna anterior – entrada no somador;
- C_o é o *carry out*, "vai um" – saída no somador;
- C_i é saída do somador anterior;
- C_o é entrada do somador seguinte.

Analisando a tabela, temos:

$$S = \overline{A}\,\overline{B}\,C_i + \overline{A}\,B\,\overline{C_i} + A\,\overline{B}\,\overline{C_i} + A\,B\,C_i$$

$$C_0 = \overline{A}\,B\,C_i + A\,\overline{B}\,C_i + A\,B\,\overline{C_i} + A\,B\,C_i$$

Passando para o mapa de Karnaugh (figura 3.29).

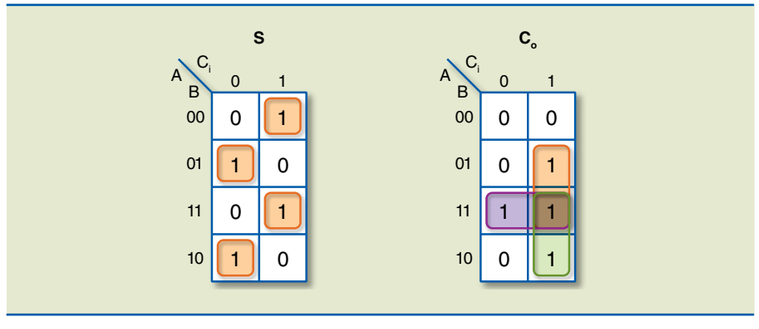


Figura 3.29
Mapa de Karnaugh para S e C_o.

Analisando o mapa de Karnaugh da figura 3.29, podemos notar que esse caso não admite simplificação, pois temos o OU EXCLUSIVO das três entradas:

$$S = A \oplus B \oplus C_i$$

$$C_o = A \cdot B + B \cdot C_i + A \cdot C_i$$

Simplificado pelo mapa de Karnaugh da figura 3.30, temos o AND da combinação duas a duas das entradas.

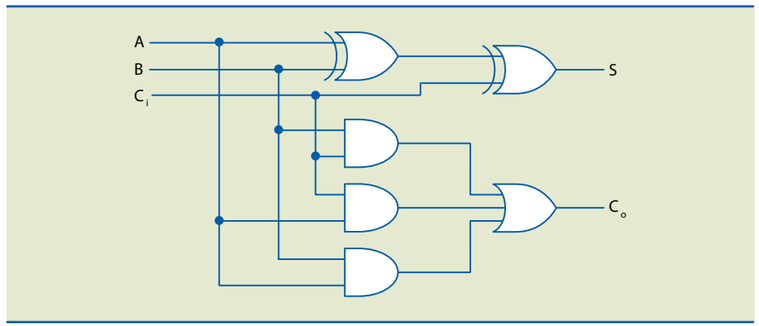


Figura 3.30
Somador completo (SC).

Podemos representar a saída $S = A + B + C_i$ por uma única porta OU EXCLUSIVO de três entradas, em nada alterando o circuito em si, apenas sua representação (figuras 3.31 e 3.32).

Figura 3.31
Somador completo (SC).

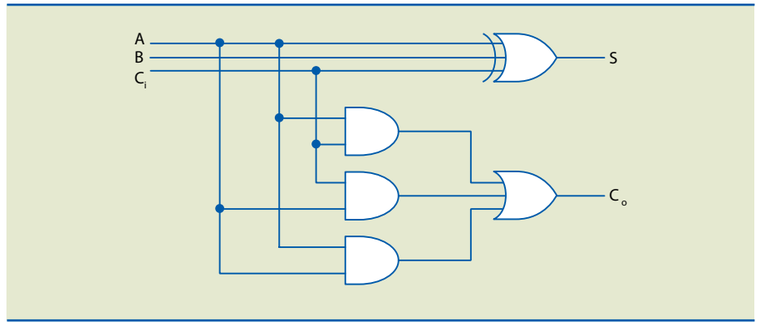
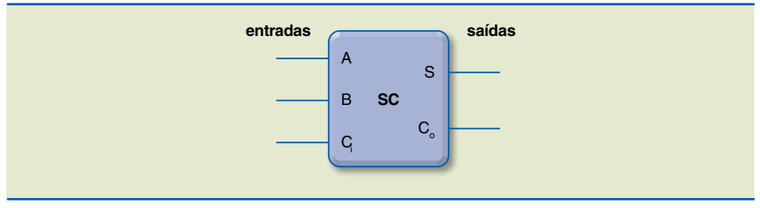


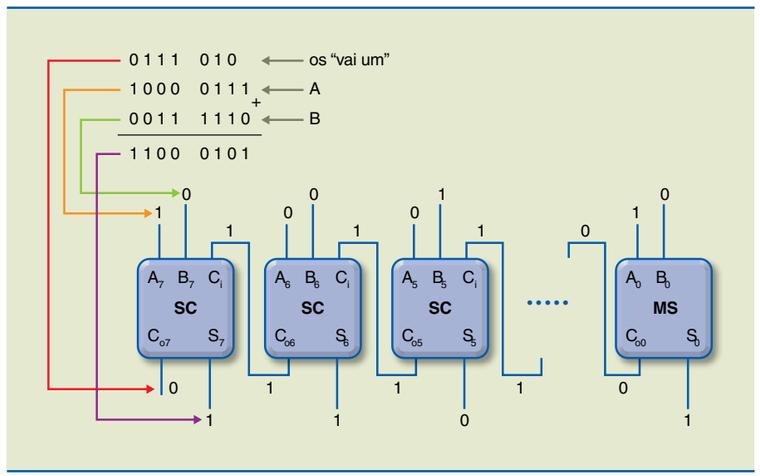
Figura 3.32
Representação simplificada do somador completo (SC).



Para somarmos dois binários, cada um formado por vários bits, podemos usar para a primeira coluna um meio somador, pois para essa coluna não existe *carry in* ("vai um anterior"). Para as demais colunas, empregamos somadores completos.

A figura 3.33 apresenta como exemplo a soma dos binários A + B, sendo A = (1000 0111) e B = (0011 1110).

Figura 3.33
Associação de meio somador com somador completo.



Podemos substituir o meio somador por um somador completo tendo $C_i = 0$. Dessa maneira, o funcional do circuito continua o mesmo, pois o MS equivale ao CS se $C_i = 0$.

3.3.3 Subtrator

Vamos relembrar na tabela seguir a tabela verdade da porta OU EXCLUSIVO.

OU EXCLUSIVO		
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

B = 0 S = A

B = 1 S = A

Analisando a tabela, podemos constatar que, se uma entrada é mantida em "0", a saída corresponde a outra entrada e, se uma entrada é mantida em "1", a saída corresponde ao complemento da outra entrada (porta INVERSORA).

Consideremos o circuito da figura 3.34, em que o MS foi substituído por um SC. Os bits do binário B são mantidos ou complementados, dependendo da variável de controle V.

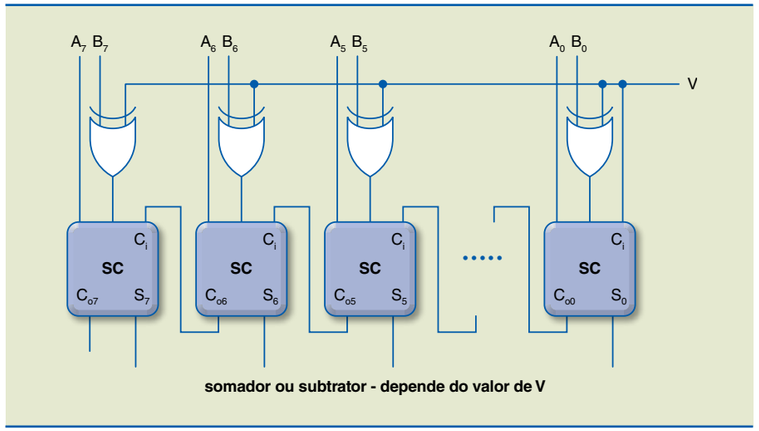


Figura 3.34
Somador ou subtrator – depende do valor de V.

Quando $V = 0$, o circuito é um somador com o mesmo funcional do circuito da figura 3.33, pois a entrada dos blocos do circuito é a mesma. Se $V = 1$, as en-

tradas dos SCs que correspondem ao binário B têm agora o complemento de B. Observemos que V = 1 coloca C_i do primeiro SC em 1, o que equivale a somar 1 ao resultado final. Por exemplo:

S = A + (complemento 1 de B) + 1 = A + complemento 2 de B

Como na representação de binário com sinal o complemento 2 corresponde ao negativo de um binário positivo, concluímos que o circuito da figura 3.34 pode ser um circuito somador ou subtrator, dependendo da variável de controle V.

Capítulo 4

Circuitos sequenciais

Os circuitos lógicos combinacionais permitem funções como decodificação, soma e subtração, comparação e muitas outras. Entretanto, funções mais avançadas (que dependem do tempo, memorização de dados, sequência de operações etc.) não podem ser implementadas com o mesmo princípio. Nesse caso, devemos recorrer ao projeto de circuitos lógicos sequenciais.

Em um circuito sequencial, os valores das saídas em determinado instante dependem não só da combinação das variáveis de entrada, mas também do valor anterior, isto é, do valor que a saída tinha antes da aplicação da nova combinação de valores nas entradas. Para isso, é necessário utilizar dispositivos de memória elementares capazes de armazenar as variáveis de saída internamente a cada transição de estado (figura 4.1).

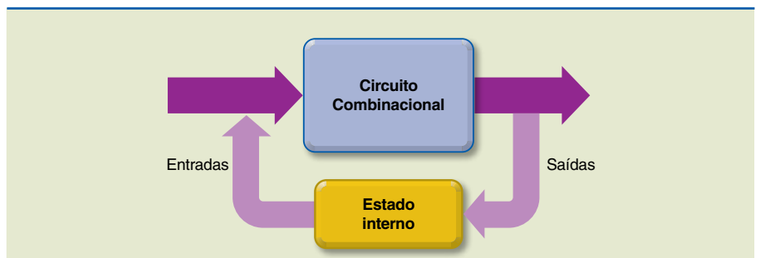


Figura 4.1
Dispositivos de memória elementar.

4.1 Elementos de memória

O *latch* RS é um elemento de memória simples com capacidade de armazenamento temporário de um bit. Esse dispositivo consiste em duas portas NOR acopladas por realimentações cruzadas (figura 4.2).

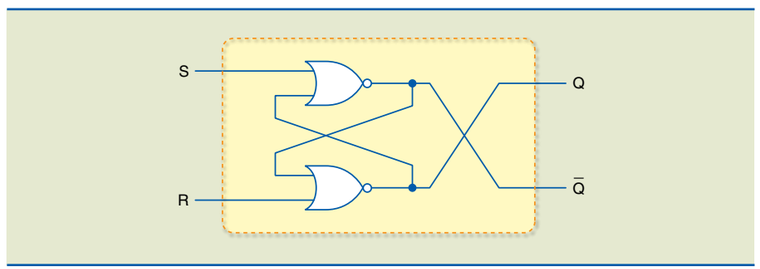


Figura 4.2
Detalhe interno do *latch* RS mostrando duas portas NOR acopladas por realimentações cruzadas.

Analisando a figura 4.2, podemos notar que as entradas S (*set*) e R (*reset*) ficam normalmente em nível "0", sendo ambas ativas em nível lógico "1". Fazendo S = 1, obtém-se Q = 1. Esse nível é mantido após a retirada do nível "1" da entrada S e permanece até que seja aplicado nível "1" na entrada R. Fazendo R = 1, obtém-se Q = 0. Esse nível é mantido após a retirada do nível "1" da entrada R e permanece até que seja aplicado nível "1" na entrada S.

A tabela verdade referente ao *latch* RS (figura 4.3) considera as entradas ativas em nível lógico alto.

Tabela verdade

Reset	Set	Q	/Q
0	0	mantém	
0	1	1	0
1	0	0	1
1	1	inválido	

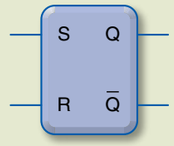


Figura 4.3
Representação do *latch* RS mostrando somente as entradas e saídas e a tabela verdade correspondente.

Analisando a tabela, podemos notar que S = 1 e R = 1 é inválido. Isso acontece porque:

- Nesse caso particular, as duas saídas Q e Q' seriam iguais a "0", o que implicaria de imediato a inconsistência com a teoria das saídas Q e Q'.
- Outro ponto crítico ocorre quando passamos desse estado para S = 0 e R = 0. Nesse caso, seguindo a tabela verdade e o comportamento do *latch*, a saída deveria permanecer inalterada, o que não acontece, gerando um estado indefinido para Qn + 1 e Q'n + 1.

Devido a essa ambiguidade, a condição S = 1 e R = 1 não é usada para *latch* RS.

O circuito do *latch* RS com portas NAND é mostrado na figura 4.4.

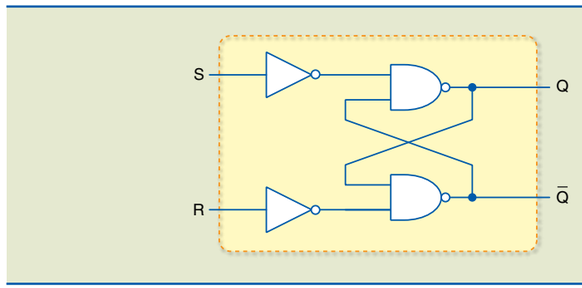
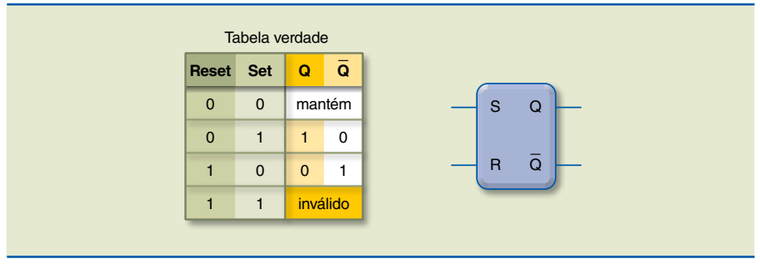


Figura 4.4
Detalhe interno do circuito do *latch* RS com portas NAND.

O circuito da figura 4.4 é equivalente ao apresentado no item anterior, portanto sua tabela verdade e símbolo lógico não se alteram (figura 4.5).

Figura 4.5
Representação do *latch* RS mostrando somente as entradas e saídas e tabela verdade correspondente.



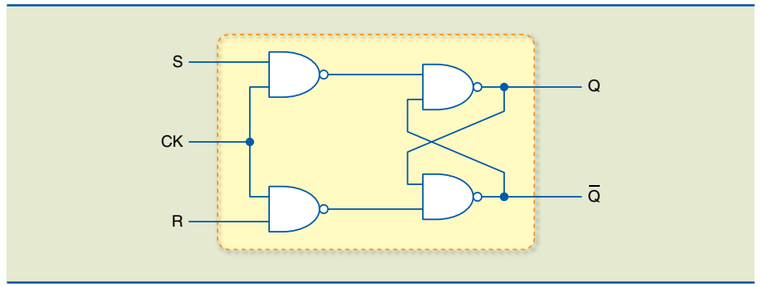
Analisando a tabela, podemos notar que S = 1 e R = 1 é inválido. Isso acontece porque:

- Nesse caso particular, as duas saídas Q e Q' seriam iguais a "1".
- Quando passamos desse estado para S = 0 e R = 0, estamos novamente gerando um estado indefinido para Qn + 1 e Q'n + 1.

Um *latch* controlado possui uma entrada *enable* que diz quando o *latch* poderá armazenar um valor. Caso *enable* = 0, o *latch* permanece em seu estado anterior, mantendo armazenado o bit. Somente quando *enable* = 1 o *latch* funcionará como antes.

A entrada *enable* também pode ser denominada *clock* (CK), ou relógio, quando ela receber um sinal de sincronismo, por isso em alguns diagramas utiliza-se a notação "CK" (figura 4.6).

Figura 4.6
Identificação das entradas S, R e ***clock*** em um *latch*.



A tabela verdade a seguir demonstra as condições das saídas, considerando as entradas R, S e *clock*.

R	S	Relógio	Q	$\bar{Q}$
X	X	0	mantém	
0	0	1	mantém	
0	1	1	1	0
1	0	1	0	1
1	1	1	erro lógico	

Observe que a condição de ambiguidade (ou erro lógico) ainda existe quando S = R = 1.

Um *latch* controlado (tipo D) é implementado colocando-se um inversor entre os terminais S e R de um *latch* RS. Nessa configuração, impede-se que as variáveis de entrada assumam valores idênticos, isto é, S = R = 0 ou S = R = 1. Assim, a entrada D passa a ser única, e os pontos correspondentes a S e R, a assumir sempre valores distintos (figura 4.7):

- Se D = 1, então S = 1 e R = 0.
- Se D = 0, então S = 0 e R = 1.

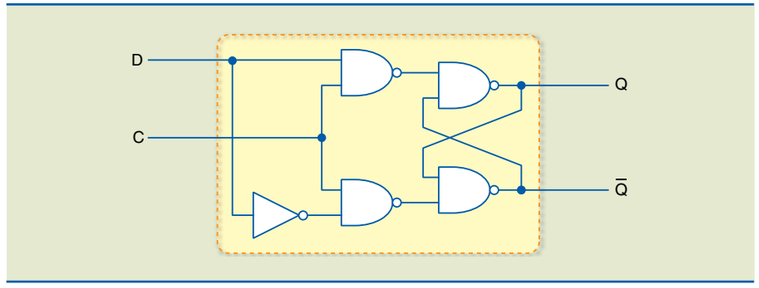


Figura 4.7
Detalhe interno de um *latch* para D = 0 e D = 1.

Observe que o problema da inconsistência foi eliminado, uma vez que é impossível aplicar sinais iguais nas entradas S e R. A figura 4.8 e sua respectiva tabela verdade possibilitam uma análise dessa configuração.

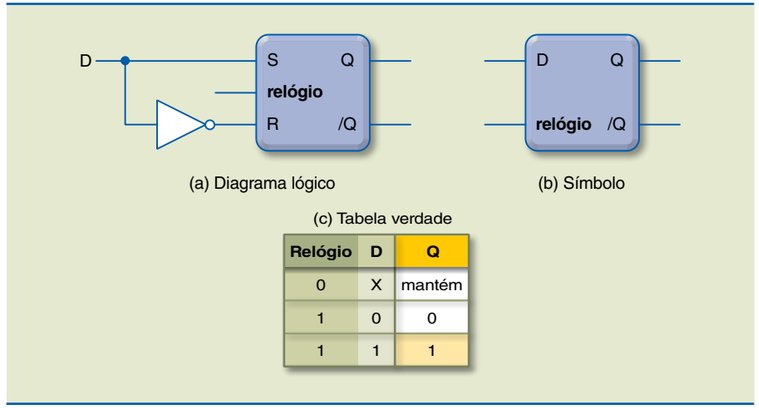


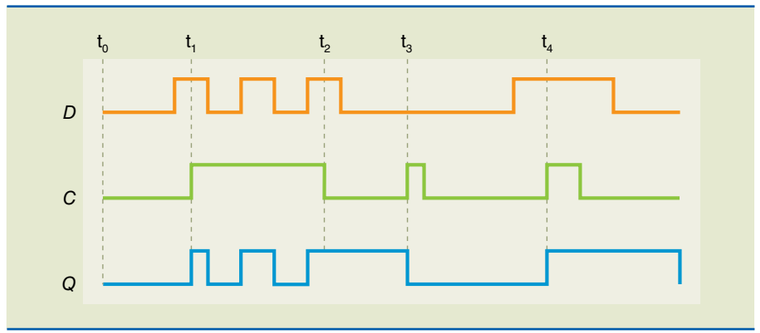
Figura 4.8
Representação do ***latch*** RS: (a) diagrama lógico, (b) símbolo e em (c) tabela verdade.

Analisando a tabela verdade, podemos entender o funcionamento, pois:

- se *enable* = 0, o *latch* permanece no estado anterior;
- se *enable* = 1 e D = 1, temos S = 1 e R = 0; portanto, a saída Q será Q = D (Q = 1);
- se *enable* = 1 e **D** = 0, temos S = 0 e R = 1; portanto, a saída Q será Q = D (Q = 0).

Concluindo, se *enable* = 1, a saída Q acompanha a entrada **D** e, se *enable* = 0, a saída do *latch* permanece inalterada, ou seja, mantém o estado anterior.

Figura 4.9
Formas de onda dos sinais para um *latch* tipo D.



As formas de onda dos sinais para um *latch* **tipo D** são apresentadas na figura 4.9, em que:

- D é a entrada de dados;
- C, o sinal de habilitação ou *clock*;
- Q, a saída do *latch*.

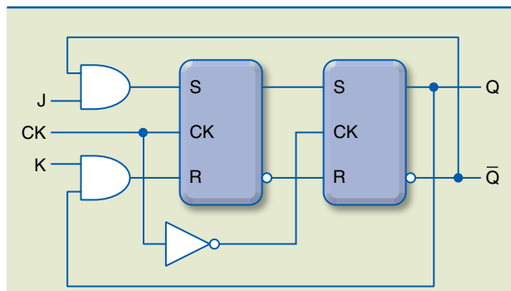
Os circuitos *latches* R, S e D apresentados anteriormente são sensíveis ao nível do sinal aplicado em sua entrada de habilitação (*enable*).

Agora, analisaremos dispositivos que dispõem de entradas de sincronismo sensíveis às **transições de nível lógico**, de "0" para "1" ou de "1" para "0". Esses dispositivos são conhecidos pela terminologia "disparados por borda" (do sinal de relógio) e podem ser de dois tipos:

- Disparados por borda de subida (transição positiva do sinal de *clock*): sensíveis às transições de nível lógico do sinal de *clock*, de "0" para "1".
- Disparados por borda de descida (transição negativa do sinal de *clock*): sensíveis às transições de nível lógico do sinal de *clock*, de "1" para "0".

Vamos iniciar analisando o ***flip-flop* J-K mestre-escravo**. Esse dispositivo possui duas entradas de dados (J e K) e tem como característica principal seus dois estágios internos, denominados mestre e escravo (figura 4.10) com a tabela verdade correspondente.

Figura 4.10
Flip-flop J-K (mestre-escravo) e tabela verdade correspondente.



J	D	Estado futuro Q futuro
0	0	Q
0	1	0
1	0	1
1	1	Q̄

A figura 4.11 mostra detalhes das ligações internas do circuito.

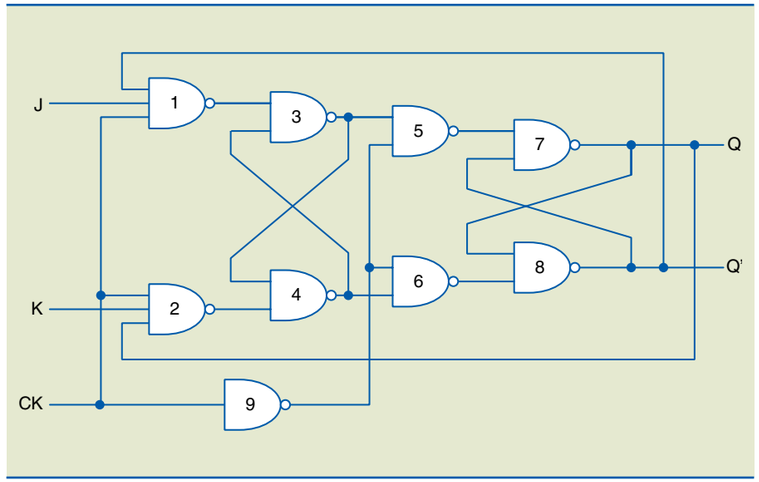


Figura 4.11
Detalhe interno de um *flip-flop J-K* (mestre-escravo).

Analisando a figura 4.11, podemos notar que:

- Se J = 0 e K = 0, as portas **1** e **2** estarão desabilitadas; portanto, após a aplicação do pulso de *clock*, o *flip-flop* não mudará de estado.

- Se J = 1 e K = 0 e Q = 0, a porta **1** habilitará (J = 1 **e** Q' = 1) e a porta **2** desabilitará (K = 0 **e** Q = 0); portanto, após a aplicação do pulso de *clock*, o estado de saída Q mudará para Q = 1.

- Se J = 1 e K = 0 e Q = 1, a porta **1** desabilitará (J = 1 **e** Q' = 0) e a porta **2** desabilitará (K = 0 **e** Q = 1); portanto, após a aplicação do pulso de *clock*, o estado de saída permanecerá inalterado (Q = 1).

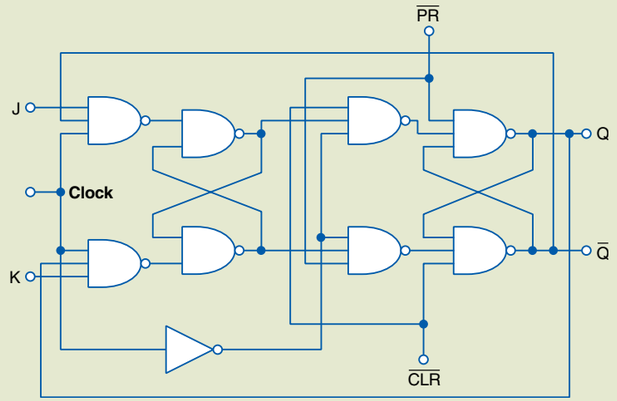
- Se J = 0 e K = 1 e Q = 0, a porta **1** desabilitará (J = 0 **e** Q' = 1) e a porta **2** desabilitará (K = 1 **e** Q = 0); portanto, após a aplicação do pulso de *clock*, o estado de saída permanecerá inalterado (Q = 0).

- Se J = 0 e K = 1 e Q = 1, a porta **1** desabilitará (J = 0 **e** Q' = 0) e a porta **2** habilitará (K = 1 **e** Q = 1); portanto, após a aplicação do pulso de *clock*, o estado de saída Q mudará para Q = 0.

- Se J = 1 e K = 1, para J = K = 1, a cada ciclo de *clock* o estado do *flip-flop* J-K se complementa; portanto, após a aplicação do sinal de *clock*, teremos: se Q = 0, a saída Q mudará para Q = 1; se Q = 1, a saída Q mudará para Q = 0.

Podemos também incluir as entradas de *preset* e *clear* nesse circuito, que passa a ter a configuração da figura 4.12. A tabela verdade inclui as entradas de *preset* (PR) e *clear* (CLR).

Figura 4.12
Detalhe interno de um *flip-flop* J-K (mestre-escravo) com as entradas *clear* e *preset* e a tabela verdade correspondente.



CLR	PR	J	K	CLK	Qn ÷ 1	Q̄n ÷ 1
		X	X	X	1	1
	1	X	X	X		1
1		X	X	X	1	
1	1			↓	Qn	Q̄n
1	1		1	↓		1
1	1	1		↓	1	
1	1	1	1	↓	Q̄n	Qn

O exemplo da figura 4.12 corresponde a um *flip-flop* J-K mestre-escravo sensível à transição negativa do sinal de relógio com entradas de *preset* e *clear* inversoras.

A figura 4.13 apresenta o circuito de *preset* e *clear* e a tabela verdade correspondente.

Figura 4.13
Configuração do *flip-flop* J-K mestre-escravo com entradas *clear* e *preset* e tabela verdade resumida.

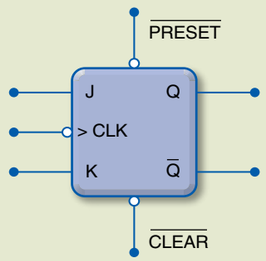


Tabela verdade

PRESET	CLEAR	Resposta do FF
1	1	Operação com clock*
0	1	Q = 1
1	0	Q = 0
0	0	Não usada

*Q irá responder a J, K e CLK

Existem outras configurações de entradas, que variam conforme o tipo de CI e o fabricante, tais como exemplificadas na figura 4.14.

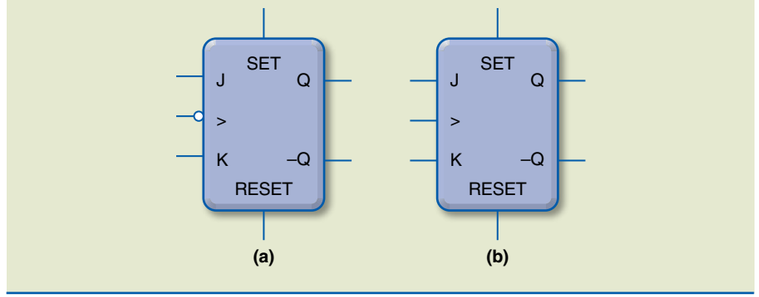


Figura 4.14
(a) *Flip-flop* J-K sensível à borda de descida e (b) *flip-flop* J-K sensível à borda de subida.

As figuras 4.15 e 4.16 apresentam dois exemplos de circuito integrado com dois *flip-flops* J-K: um da família CMOS e outro da TTL e suas tabelas verdade.

Figura 4.15
CI 4027B com dois *flip-flops* J-K sensíveis à borda de subida com entradas de *clear* e *preset* e a tabela verdade correspondente.

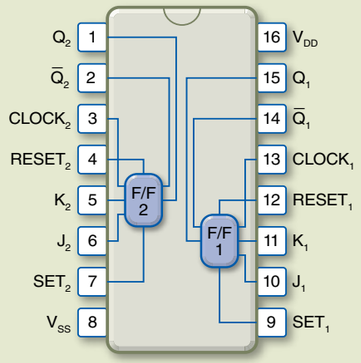


Tabela verdade

Entradas					Saídas	
CLEAR	PRESET	CLOCK	J	K	Q	Q̄
1	0	X	X	X	0	1
0	1	X	X	X	1	0
1	1	X	X	X	1	1
0	0	↑	0	0	Q_0	$\bar{Q}_0$
0	0	↑	1	0	1	0
0	0	↑	0	1	0	1
0	0	↑	1	1	TOGGLE	
0	0	0	X	X	Q_0	$\bar{Q}_0$
0	0	1	X	X	Q_0	$\bar{Q}_0$

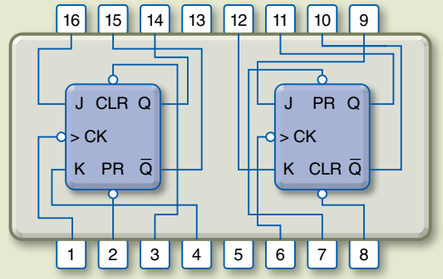


Tabela verdade

Entradas					Saídas	
CLEAR	PRESET	CLOCK	J	K	Q	Q̄
0	1	X	X	X	0	1
1	0	X	X	X	1	0
0	0	X	X	X	1	1
1	1	↓	0	0	Q_0	$\bar{Q}_0$
1	1	↓	1	0	1	0
1	1	↓	0	1	0	1
1	1	↓	1	1	TOGGLE	
1	1	1	X	X	Q_0	$\bar{Q}_0$
1	1	0	X	X	Q_0	$\bar{Q}_0$

Figura 4.16 CI 7476 com dois *flip-flops* J-K sensíveis à borda de descida e tabela verdade correspondente.

Implementação de um flip-flop D a partir do J-K

Um *flip-flop* tipo D sensível à borda pode ser obtido com um inversor entre as entradas J e K, como se pode observar na figura 4.17. Nesse tipo de *flip-flop*, a saída Q assume o nível lógico presente na entrada D toda vez que ocorre transição do sinal de *clock* (nesse exemplo, as transições de estado ocorrem no instante de subida do sinal de *clock*, conforme ilustram os gráficos).

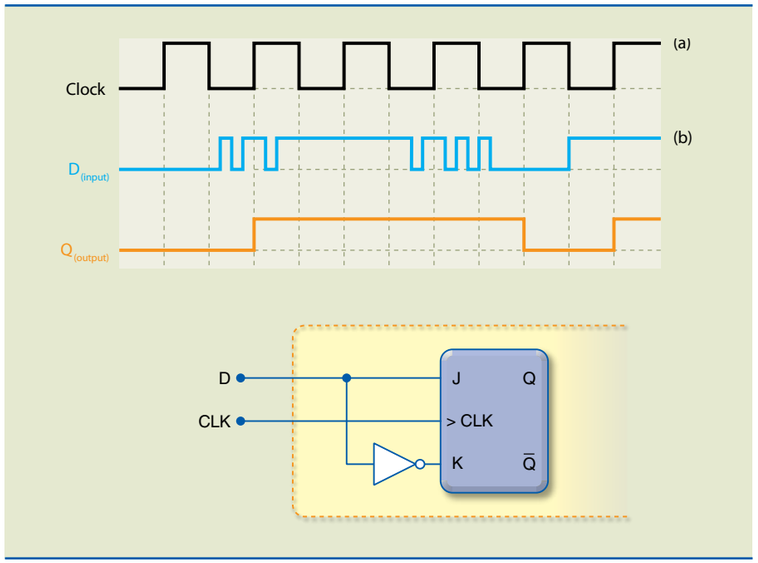


Figura 4.17
(a) *Flip-flop* tipo D a partir do J-K e (b) as formas de onda da entrada e da saída em função do *clock*.

Exemplos de circuitos integrados de *flip-flops* tipo D CMOS e TTL

4013 – Dois flip-flops tipo D sensíveis à borda de subida com entradas de preset e clear

A figura 4.18 apresenta esse dispositivo, e a tabela verdade correspondente.

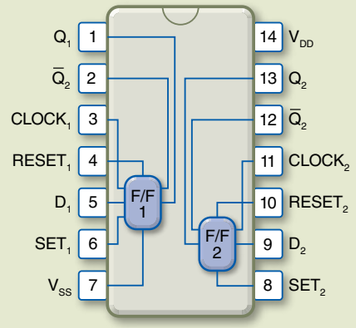


Tabela verdade

Entradas				Saídas	
CLEAR	PRESET	CLOCK	D	Q	Q̄
0	1	X	X	1	0
1	0	X	X	0	1
1	1	X	X	1	1
0	0	↑	0	0	1
0	0	↑	1	1	1
0	0	0	X	Q_0	$\bar{Q}_0$
0	0	1	X	Q_0	$\bar{Q}_0$

Figura 4.18
Dois *flip-flops* tipo D sensíveis à borda de subida com entradas de *preset* e *clear* e a tabela verdade correspondente.

7474 – Dois flip-flops tipo D sensíveis à borda de subida com entradas de preset e clear inversoras

A figura 4.19 apresenta esse dispositivo, e tabela verdade correspondente.

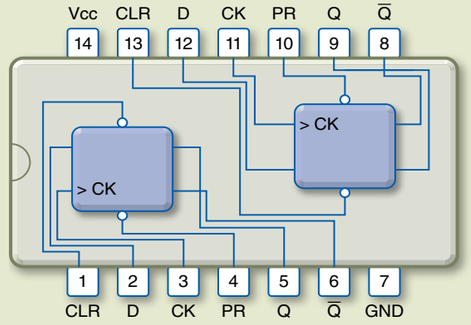


Tabela verdade

Entradas				Saídas	
PRESET	CLEAR	CLOCK	D	Q	Q̄
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	1	1
1	1	↑	1	1	0
1	1	↑	0	0	1
1	1	0	X	Q_0	$\bar{Q}_0$
1	1	1	X	Q_0	$\bar{Q}_0$

Figura 4.19
CI 7474 com dois *flip-flops* tipo D sensíveis à borda de subida com entradas de *preset* e *clear* inversoras.

Implementação de um *flip-flop* T a partir do J-K

O *flip-flop* T ou *toggle* muda sua saída a cada transição do sinal de *clock* (ver exemplo na figura 4.20, na transição positiva). Consequentemente, a frequência do sinal de saída é metade da frequência do sinal de entrada aplicado na entrada T.

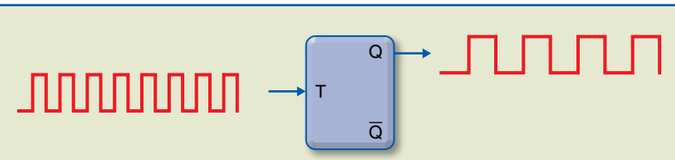


Figura 4.20
Representação da mudança da saída para um *flip-flop* T.

O *flip-flop* T é obtido a partir do *flip-flop* J-K aplicando nível lógico alto tanto na entrada J como na K (figura 4.21).

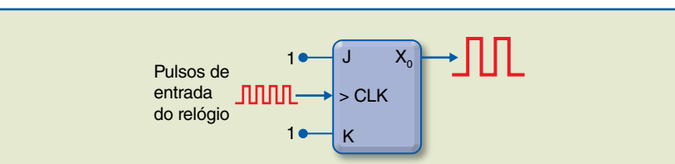


Figura 4.21
Representação mostrando como um *flip-flop* T é obtido a partir do *flip-flop* J-K.

A figura 4.22 mostra as formas de onda correspondentes nas saídas Q e Q', a partir do *clock*.

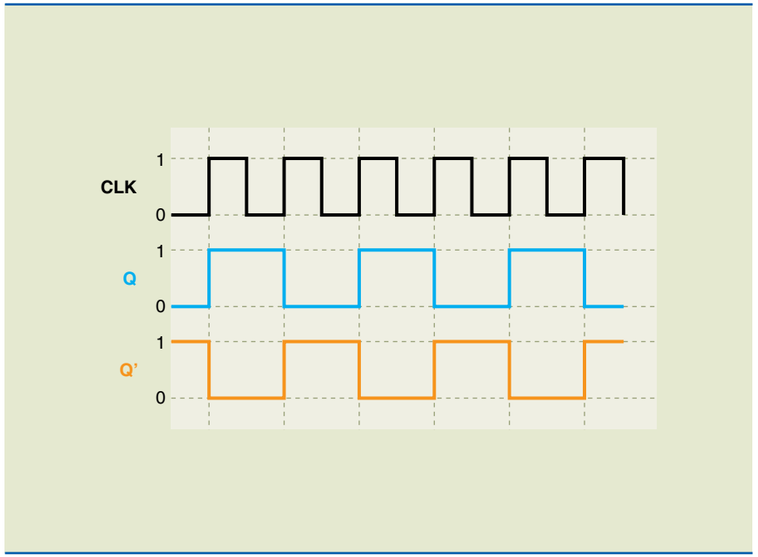


Figura 4.22
Formas de onda correspondentes nas saídas Q e Q', a partir do *clock*.

4.2 Contadores

Contadores são circuitos digitais que geram determinada sequência de estados, sob o comando de um sinal de *clock*. São utilizados na contagem de pulsos provenientes de chaves e de sensores, na construção de temporizadores e relógios digitais, para gerar sequências de pulsos e medir frequência, e também fazem parte de circuitos eletrônicos como conversores analógico-digital e digital-analógico, geradores de endereços de matrizes de memória etc.

Os contadores são basicamente divididos em duas categorias – **assíncronos** e **síncronos** – e podem ser classificados de acordo com a sequência (crescente ou decrescente) e com o módulo (binário, decimal, módulo n).

4.2.1 Contadores assíncronos

Os contadores assíncronos não possuem entradas comuns de sinal de *clock*. O sinal inicial é aplicado no primeiro estágio; os demais recebem o sinal do estágio anterior.

Contador binário

Um contador binário pode ser construído a partir de *flip-flops* J-K conectando a saída de uma célula à entrada de *clock* da célula seguinte. As entradas J e K de todos os *flip-flops* são mantidas em nível lógico "1" para produzir o efeito *toggle* a cada pulso de *clock*. Para cada dois pulsos de *clock* na entrada de determinada célula é produzido um pulso na respectiva saída. Isso resulta uma sequência binária quando o número de *flip-flops* é igual a quatro. Esse dispositivo geralmente é chamado de contador de pulsos (*ripple counter*).

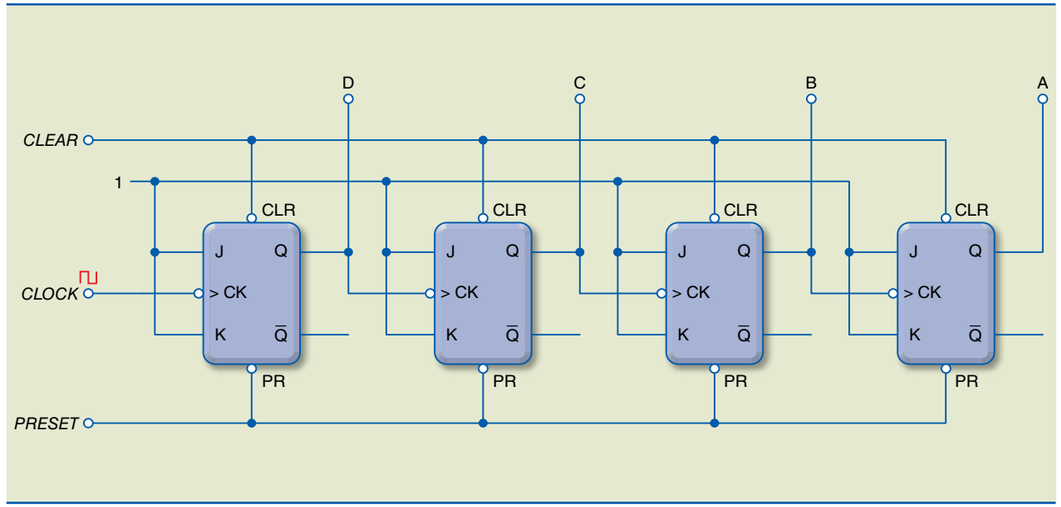


Figura 4.23
Representação de um Contador de pulsos.

As formas de onda nas saídas em função do sinal de *clock* são apresentadas na figura 4.24.

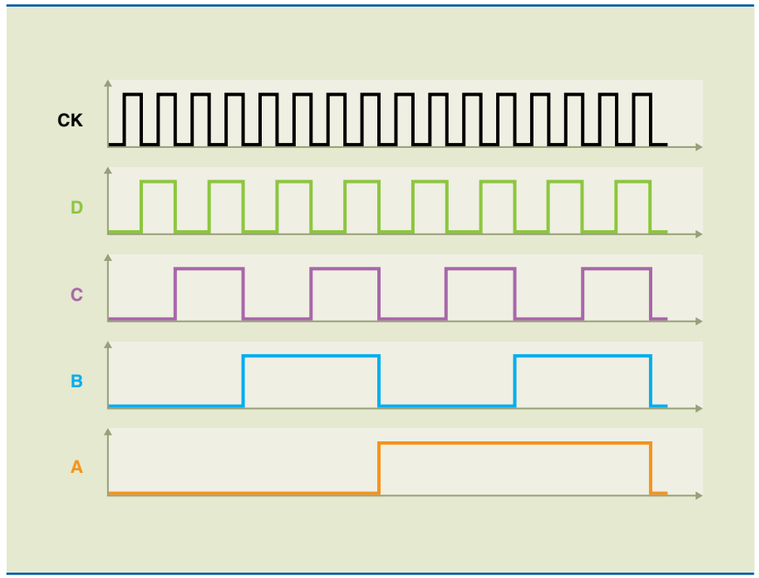


Figura 4.24
Formas de onda nas saídas em função do sinal de *clock*.

Contador de década (BCD *counter*)

Uma das representações de dados numéricas mais utilizadas é o decimal codificado em binário (BCD – *binay coded decimal*). Nessa codificação, cada número decimal inteiro é representado por um código binário de quatro dígitos, conforme a tabela 4.1.

Tabela 4.1

Decimal	BCD 8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Observe, no exemplo da figura 4.25, a equivalência entre um número decimal e sua representação em BCD.

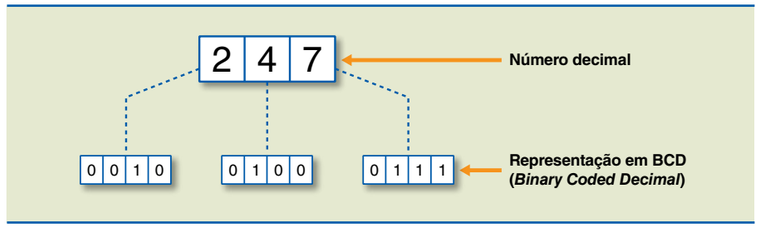


Figura 4.25
Equivalência entre o número 247 e sua representação em BCD.

Um contador BCD ou contador de décadas (figura 4.26) pode ser construído a partir de um contador binário capaz de encerrar a transmissão de pulsos quando a contagem atinge o estado correspondente ao número decimal 9 (1001 em binário).

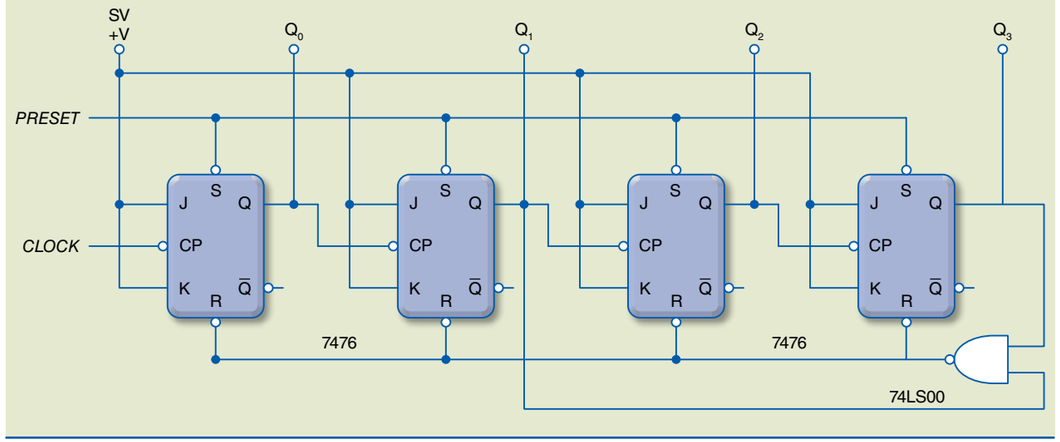


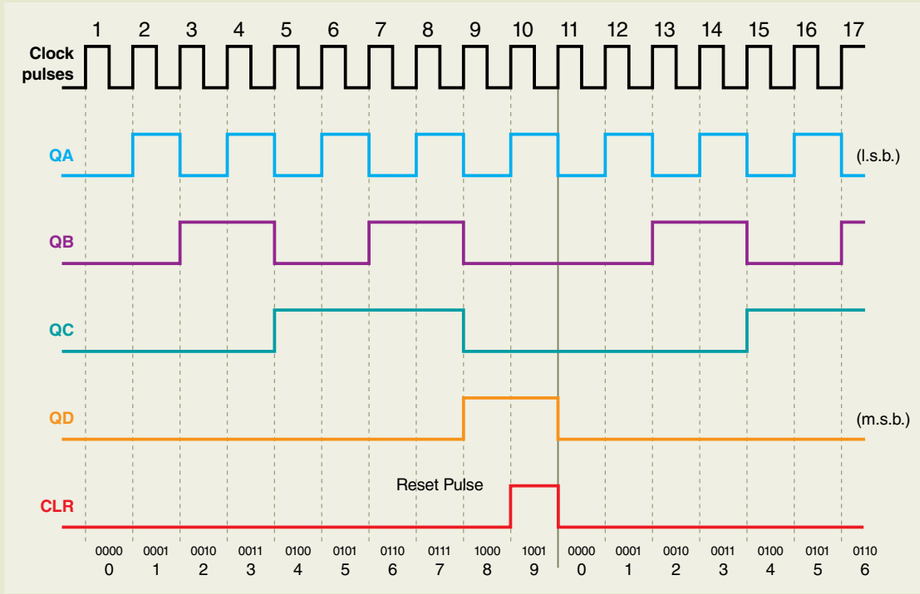
Figura 4.26
Circuito lógico do contador BCD.

Análise do funcionamento

Observando a figura 4.26, podemos analisar o funcionamento do contador BCD. Vamos considerar a situação: uma vez que o pulso seguinte levaria ao estado correspondente ao binário 1010, bastaria conectarmos os dois bits altos (Q_3 e Q_1) às entradas de uma porta NAND cuja saída é ligada à entrada assíncrona de *clear* dos *flip-flops*. Isso provocaria um *reset* automático nos *flip-flops* após o número 9, reiniciando, assim, a contagem.

Observe na figura 4.27 as formas de onda para o *flip-flop* da figura 4.26.

Figura 4.27
Formas de ondas na saída em função do ***clock*** inicial.



4.2.2 Contadores síncronos

Nesse tipo de contador, o sinal de *clock* é comum a todos os *flip-flops* que o compõem, ou seja, todos os estágios são sincronizados simultaneamente.

É possível projetar um **contador** síncrono utilizando *flip-flops* tipo D. Para isso, devemos seguir as etapas:

1) Especificar a sequência do contador.

Por exemplo, a sequência é:

5, 7, 3, 2, 6 → repetidamente, ou seja, em binário: 101, 111, 011, 010, 110.

2) Gerar a tabela de estados (tabela 4.2)

Tabela 4.2

Estado Atual				Estado Futuro			
	A	B	C		A	B	C
0	0	0	0	X	X	X	X
1	0	0	1	X	X	X	X
2	0	1	0	6	1	1	0
3	0	1	1	2	0	1	0
4	1	0	0	X	X	X	X
5	1	0	1	7	1	1	1
6	1	1	0	5	1	0	1
7	1	1	1	3	0	1	1

3) Determinar quais os sinais de entrada necessários para forçar os *flip-flops* a assumir os valores desejados na sequência (tabela 4.3).

Tabela 4.3

Estado Atual				Estado Futuro				Entradas dos *Flip-Flops*		
	A	B	C		A	B	C	D_A	D_B	D_C
0	0	0	0	X	X	X	X	X	X	X
1	0	0	1	X	X	X	X	X	X	X
2	0	1	0	6	1	1	0	1	1	0
3	0	1	1	2	0	1	0	0	1	0
4	1	0	0	X	X	X	X	X	X	X
5	1	0	1	7	1	1	1	1	1	1
6	1	1	0	5	1	0	1	1	0	1
7	1	1	1	3	0	1	1	0	1	1

De acordo com a tabela 4.3, há três funções a serem implementadas: D_A, D_B e D_C, que podem ser apresentadas conforme a figura 4.28.

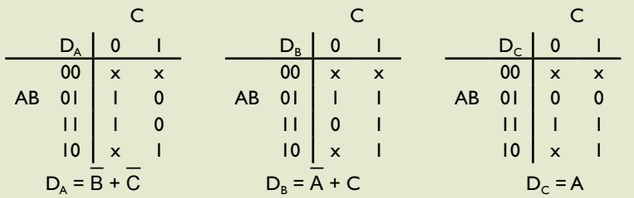


Figura 4.28
Diagramas de Karnaugh para as funções D_A, D_B e D_C.

Dessa maneira, podemos montar um circuito lógico que atenda a essas funções (figura 4.29).

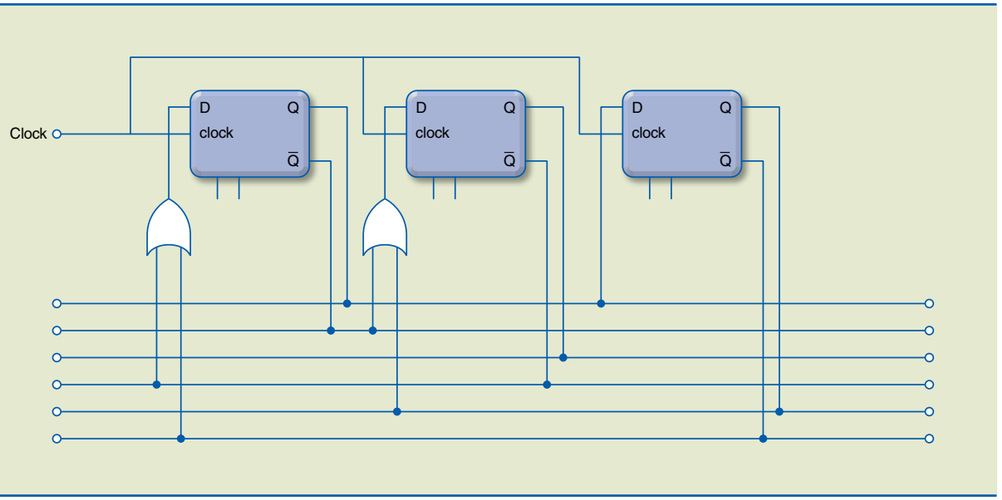


Figura 4.29
Circuito lógico para as funções D_A, D_B e D_C.

Implementação do circuito utilizando flip-flops tipo J-K

A figura 4.30 apresenta um *flip-flop* J-K e a tabela verdade correspondente. Observe que Qa é o valor anterior da saída Q antes da aplicação dos valores das entradas J e K. As mudanças somente ocorrem na variação (descida) de "1" para "0" dos pulsos aplicados na entrada de *clock*.

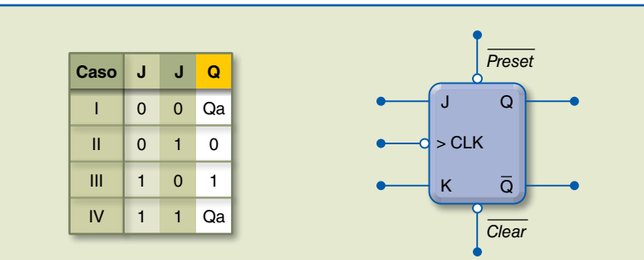


Figura 4.30
Flip-flop J-K.

Com base nas informações da tabela verdade, podemos elaborar uma **tabela de transição de estados do J-K** (tabela 4.4).

Tabela 4.4

Casos	Qa	Q	J	K
I e II	0	0	0	X
III e IV	0	I	I	X
II e IV	I	0	X	I
I e III	I	I	X	0

A tabela de transições (tabela 4.5) apresenta as entradas necessárias para forçar os valores nas saídas dos *flip-flops* a ir para a sequência desejada.

Tabela 4.5

Estado Atual				Estado Futuro				Entradas dos Flip-Flops					
	A	B	C		A	B	C	J_A	K_A	J_B	K_B	J_C	K_c
0	0	0	0	X	X	X	X	X	X	X	X	X	X
I	0	0	I	X	X	X	X	X	X	X	X	X	X
2	0	I	0	6	I	I	0	I	X	X	0	0	X
3	0	I	I	2	0	I	0	0	X	X	0	X	I
4	I	0	0	X	X	X	X	X	X	X	X	X	X
5	I	0	I	7	I	I	I	X	0	I	X	X	0
6	I	I	0	5	I	0	I	X	0	X	I	I	X
7	I	I	I	3	0	I	I	X	I	X	0	X	0

De acordo com a tabela 4.5, podemos elaborar o mapa de Karnaugh identificando as funções a serem implementadas, conforme mostra a figura 4.31.

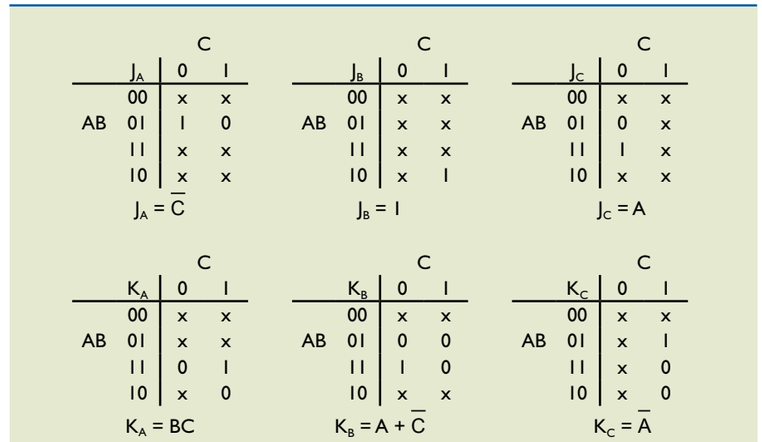


Figura 4.31
Diagramas de Karnaugh para as funções J_A, J_B, J_C, K_A, K_B e K_C.

Dessa maneira, podemos montar um circuito lógico que atenda a essas funções (figura 4.32).

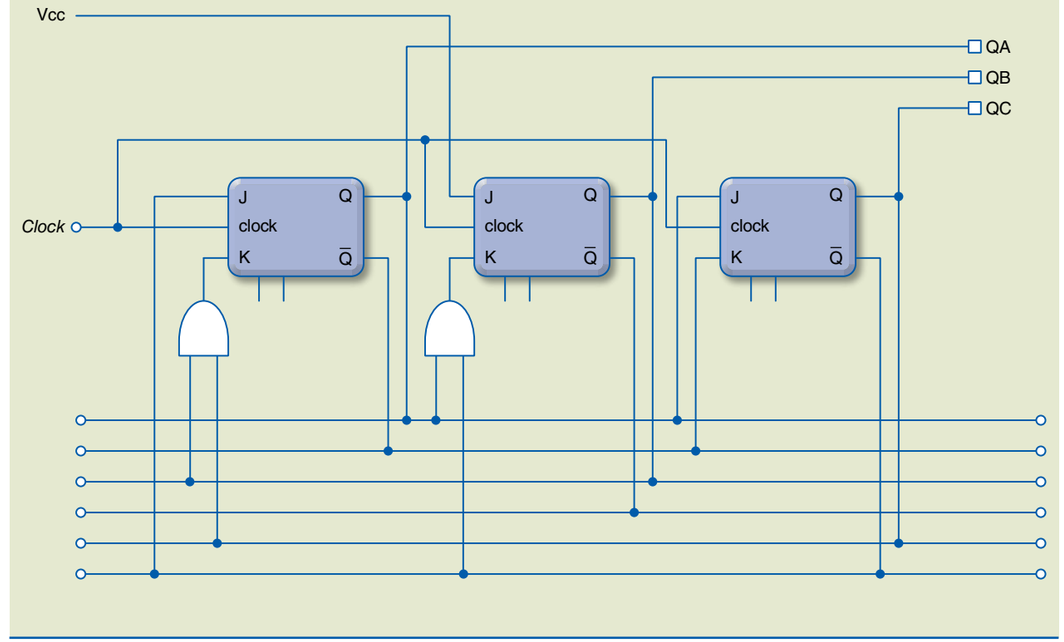


Figura 4.32
Circuito lógico para as funções J_A, J_B, J_C, K_A, K_B e K_C.

Projeto: contador decimal (BCD) síncrono

As informações necessárias para montar um contador decimal (BCD) síncrono são as seguintes:

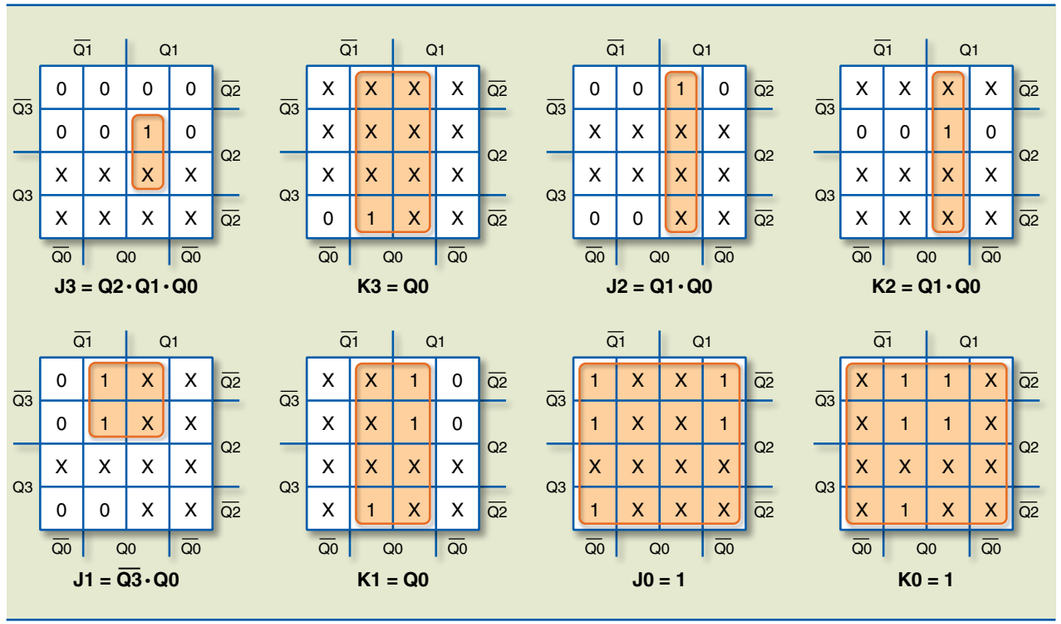
Tabela 4.6

1) Tabela de estados.

	Saídas Atuais				Saídas Futuras				Entradas dos *flip-flops*							
Estados	Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0	J3	K3	J2	K2	J1	K1	J0	K0
1	0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	x
2	0	0	0	1	0	0	1	0	0	x	0	x	1	x	x	1
3	0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
4	0	0	1	1	0	1	0	0	0	x	1	x	x	1	x	1
5	0	1	0	0	0	1	0	1	0	x	x	0	0	x	1	x
6	0	1	0	1	0	1	1	0	0	x	x	0	1	x	x	1
7	0	1	1	0	0	1	1	1	0	x	x	0	x	0	1	x
8	0	1	1	1	1	0	0	0	1	x	x	1	x	1	x	1
9	1	0	0	0	1	0	0	1	x	0	0	x	0	x	1	x
10	1	0	0	1	0	0	0	0	x	1	0	x	0	x	x	1

Figura 4.33
Diagramas de Karnaugh e expressões lógicas.

2) Diagramas de Karnaugh e expressões lógicas (figura 4.33).



3) Circuito lógico (figura 4.34).

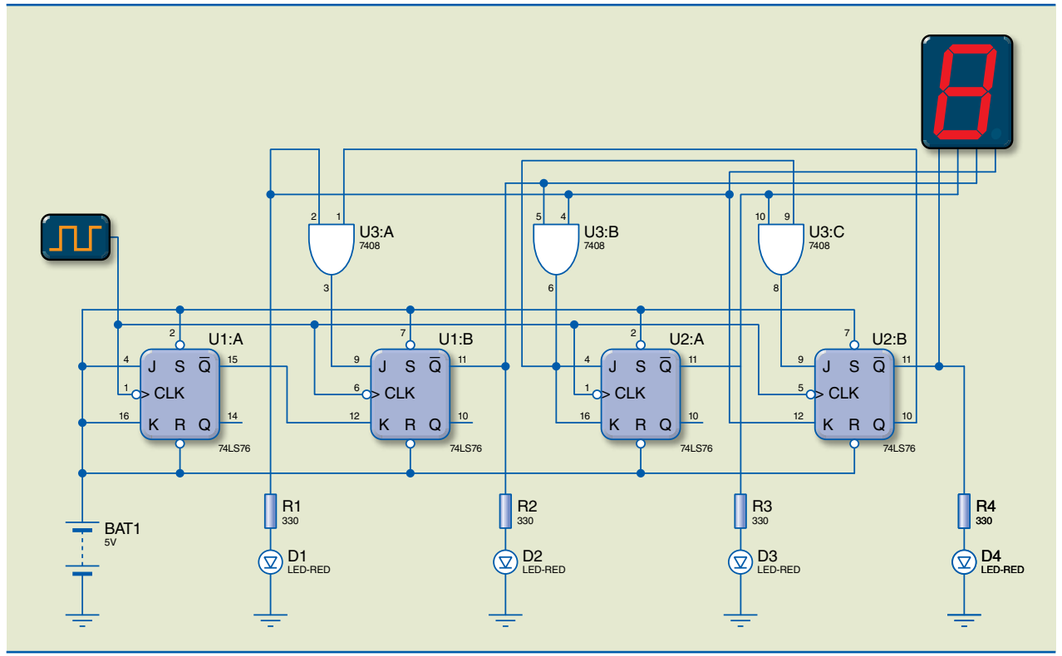


Figura 4.34
Circuito lógico contador decimal (BCD) síncrono.

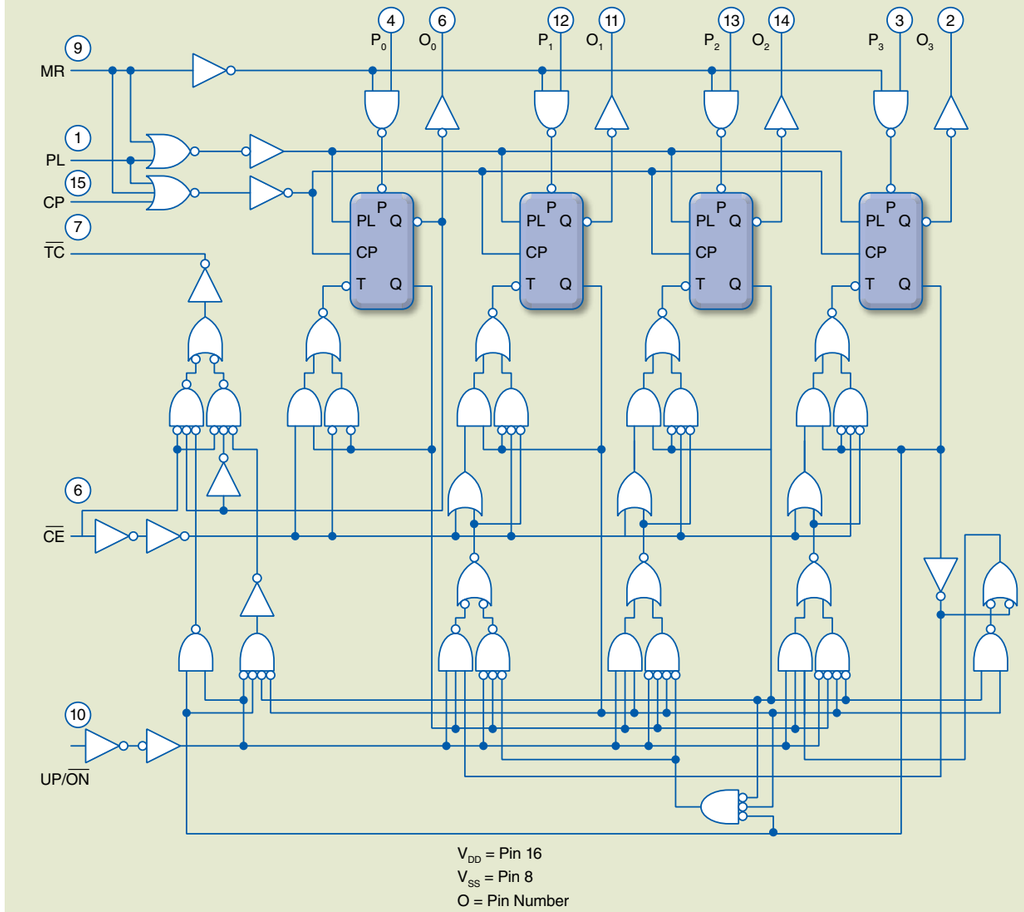
Exemplo de circuito integrado CMOS 4510 BCD counter

Pin Description		
Pin nº	Symbol	Name and Function
1	PL	parallel load input (active HIGH)
4, 12, 13, 3	D0 to D3	parallel inputs
5	CE	count enable input (active LOW)
6, 11, 14, 2	Q0 to Q3	parallel outputs
7	TC	terminal count output (active LOW)
8	GND	ground (0V)
9	MR	asynchronous master reset input (active HIGH)
10	UP/DN	up/down control input
15	CP	clock input (LOW-to-HIGH, edge-triggered)
16	VCC	positive supply voltage

Tabela 4.7
Descrição dos pinos do circuito integrado CMOS 4510 BCD *counter.*

Figura 4.35
Identificação dos pinos do circuito integrado CMOS 4510 BCD *counter.*

Figura 4.36
Detalhe das ligações internas do circuito integrado CMOS 4510 BCD *counter.*



Exemplo de circuito de teste para o contador de décadas 4510.

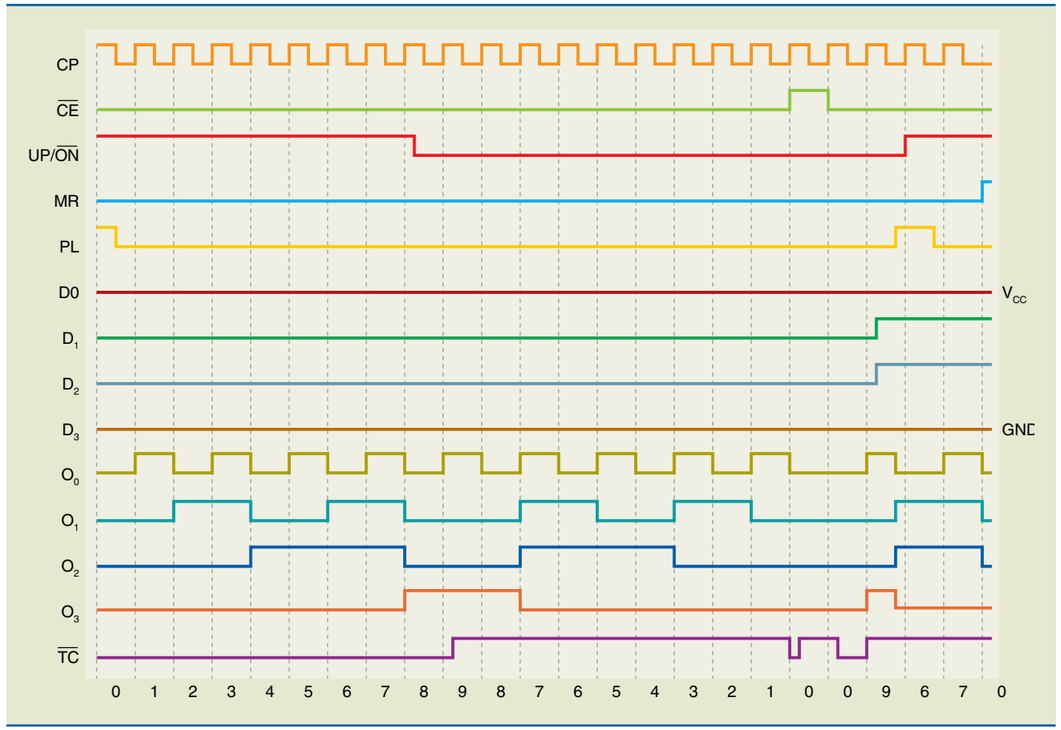


Figura 4.37
Diagrama de tempos do circuito integrado CMOS 4510 BCD *counter.*

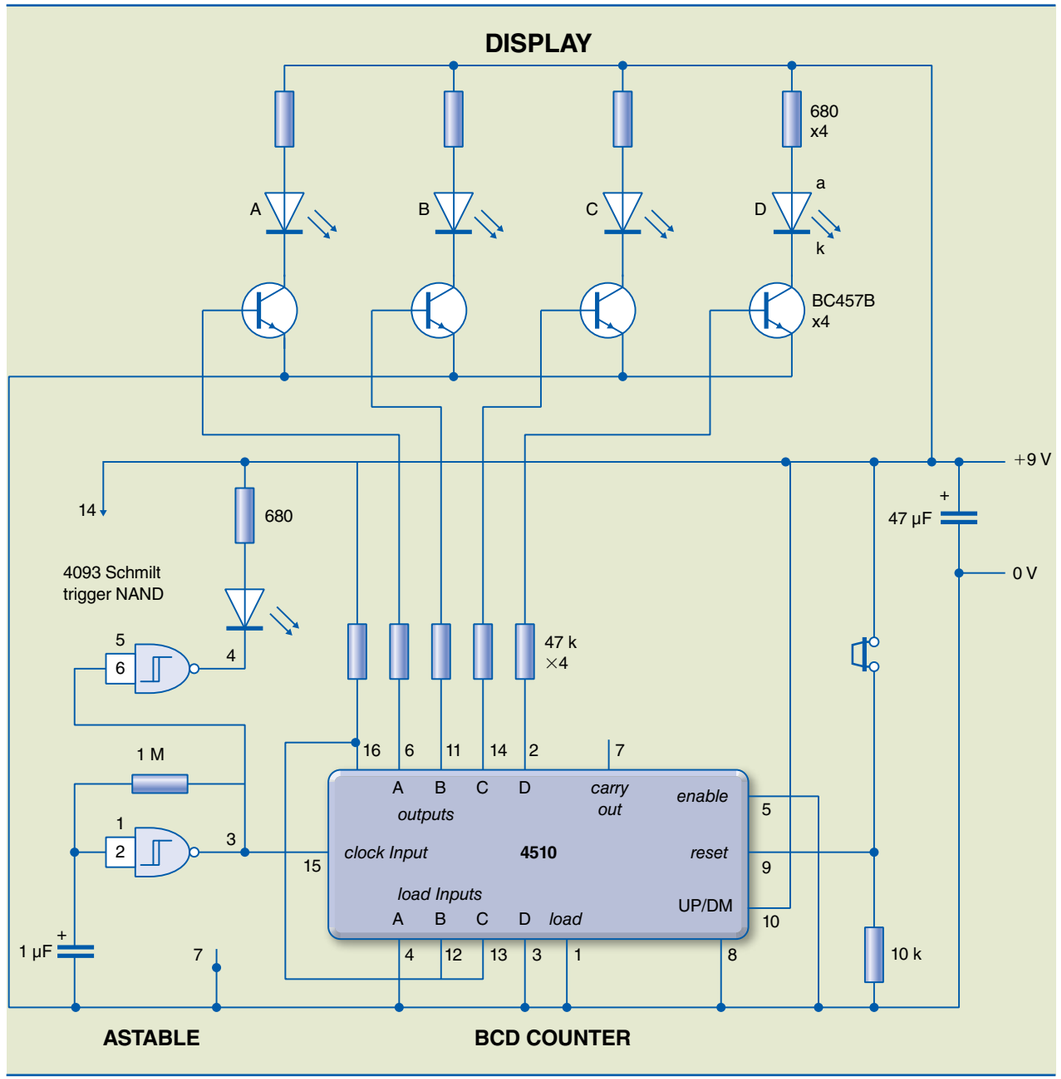


Figura 4.38
Detalhes internos de circuito de teste para o contador de décadas 4510.

Um contador crescente/decrescente tem a lógica interna apresentada na figura 4.39.

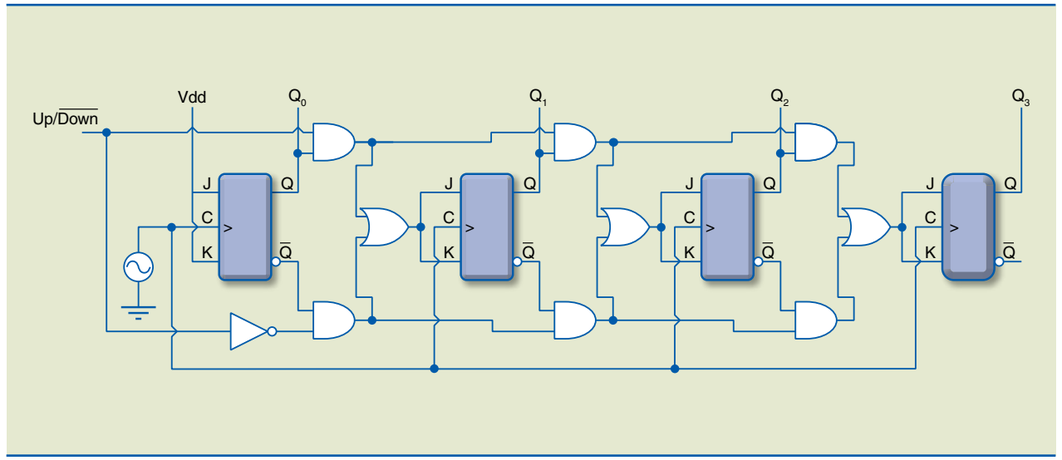


Figura 4.39
Detalhes internos de um contador crescente/decrescente.

Contador em anel

Contador em anel é um conjunto de *flip-flops* conectados em cascata à saída do último estágio conectado à entrada do primeiro, fechando um anel. Um uso comum desse circuito consiste em um único bit = 1, que circula através das saídas. Por exemplo, se forem utilizados quatro *flip-flops*, haverá quatro estados de saída (0001 / 0010 / 0100 / 1000), e cada um deles se repetirá a cada quatro ciclos de *clock*. Nesse caso, ele pode ser usado como um contador cíclico de n estados. O circuito da figura 4.40 mostra um contador em anel módulo 4.

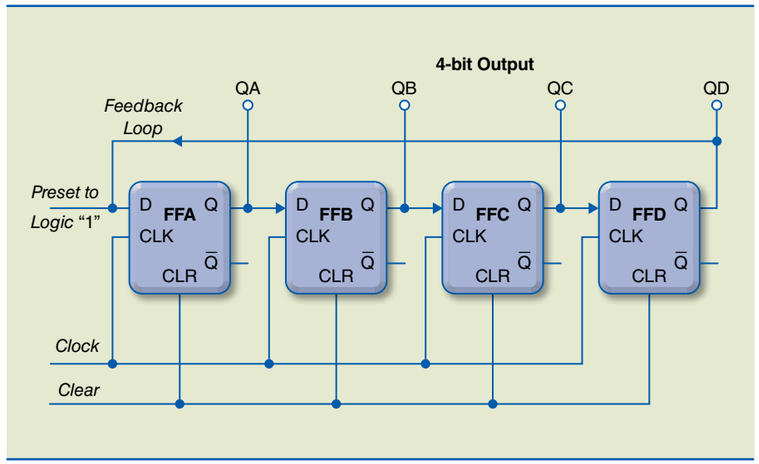


Figura 4.40
Contador em anel módulo 4.

A tabela 4.8 apresenta a sequência de estados do circuito de contador em anel módulo 4.

Tabela 4.8

Pulso de Clock	Q3	Q2	Q1	Q0	
0	0	0	0	1	↑
1	0	0	1	0	
2	0	1	0	0	
3	1	0	0	0	

Contador Johnson

Contador Johnson (ou contador de anel torcido) é um contador em anel modificado, no qual a saída do último estágio invertida é realimentada para a entrada do primeiro estágio. O circuito da figura 4.41 mostra um contador Johnson módulo 4.

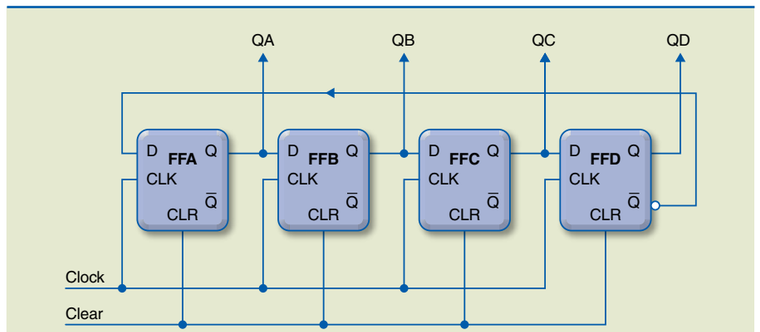


Figura 4.41
Contador Johnson módulo 4.

A tabela 4.9 apresenta a sequência de estados gerada pelo circuito do contador Johnson módulo 4.

Tabela 4.9

Pulso de Clock	Q3	Q2	Q1	Q0	
0	0	0	0	0	↑
1	0	0	0	1	
2	0	0	1	1	
3	1	1	1	1	
4	1	1	1	1	
5	1	1	1	0	
6	1	1	0	0	
7	1	0	0	0	

4.3 Registradores de deslocamento

Registrador é um circuito formado por interligações de *flip-flops* com a finalidade de armazenar informação binária (número binário) pelo tempo que for necessário.

Os registradores são utilizados em operações aritméticas de complementação, multiplicação e divisão, em conversão de uma informação série em paralela e também em vários outros tipos de circuitos digitais.

4.3.1 Informação série e informação paralela

Uma informação é chamada **informação série** quando os bits são apresentados sequencialmente, um após o outro, necessitando somente de uma via para o transporte dos bits. Esse modo de transferir informação é conhecido como **deslocamento em série**.

Uma informação é chamada **informação paralela** quando os bits são apresentados simultaneamente; assim, a transferência da informação acontece em um único instante. É necessária uma quantidade de vias para transmissão igual ao número de bits da informação. Esse modo é conhecido como **deslocamento paralelo**.

A entrada e a saída de um registrador podem ser configuradas nesses dois modos, resultando em quatro possibilidades: entrada série – saída série, entrada série – saída paralela, entrada paralela – saída série, entrada paralela – saída paralela (figura 4.42).

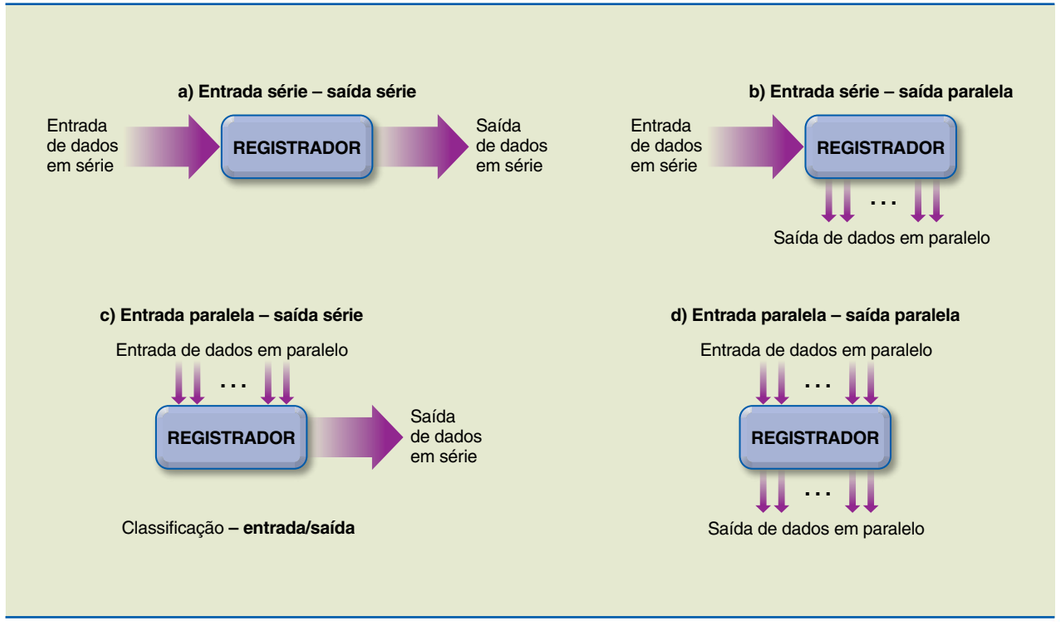
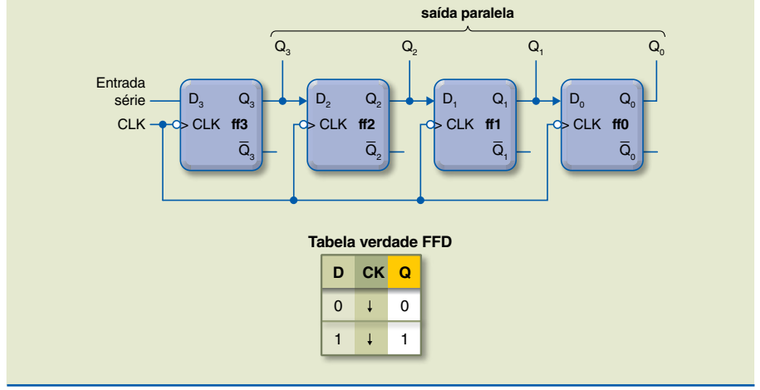


Figura 4.42
Configurações da entrada e da saída de um registrador: (a) entrada série – saída série, (b) entrada série – saída paralela, (c) entrada paralela – saída série e (d) entrada paralela – saída paralela.

4.3.2 Registrador de deslocamento para a direita

O circuito da figura 4.43 mostra como o registrador de deslocamento pode ser montado usando *flip-flops* tipo D.

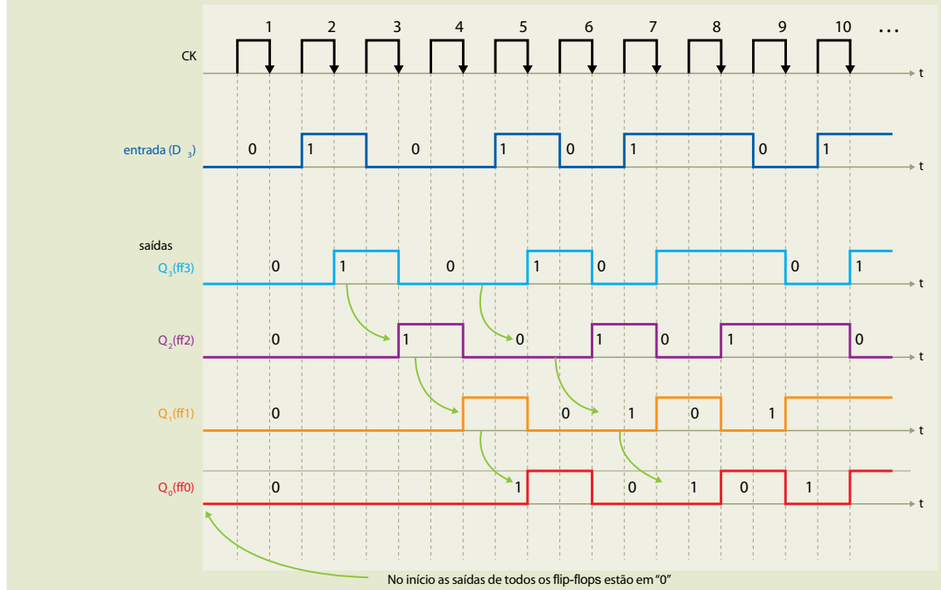
Figura 4.43
Registrador de deslocamento para a direita usando *flip-flop* tipo D e tabela verdade.



No símbolo dos *flip-flops (ffs)* da figura 4.43, a "bolinha" na entrada do *clock* indica sensibilidade à borda negativa. Os *flip-flops* desse circuito são do tipo D, sensível à borda negativa, como podemos observar pelos símbolos dos *ffs*. Na tabela verdade, a seta apontando para baixo indica sensibilidade à borda negativa.

Figura 4.44
Formas de onda do registrador de quatro bits.

As formas de onda de entrada e saída do registrador de quatro bits são apresentadas na figura 4.44.



Em um *flip-flop* mestre-escravo, a atualização da saída devido à transição do *clock* só ocorre imediatamente após o fim da transição do *clock*.

No circuito figura 4.43, observamos que o *clock* ocorre simultaneamente em todos os *flip-flops*. No momento da transição negativa do *clock*, D_2 tem como entrada o valor de Q_3 anterior à transição do *clock*, pois Q_3 somente terá seu valor atualizado após o fim da transição.

Vamos verificar como progride o primeiro bit "1" de entrada nos *flip-flops* do registrador em análise. Acompanhe pelas formas de onda na figura 4.44.

- Após o segundo pulso, o bit "1" é colocado na saída Q_3 do ff3.
- Após o terceiro pulso, o bit "1" é colocado na saída Q_2 do ff2.
- Após o quarto pulso, o bit "1" é colocado na saída Q_1 do ff1.
- Após o quinto pulso, o bit "1" é colocado na saída Q_0 do ff0.
- Após o sexto pulso, o bit "1" é perdido, ou seja, não está na saída de nenhum *flip-flop* do circuito.

Como podemos observar, o primeiro bit "1" deslocou-se para a direita a cada pulso de *clock*. O deslocamento que ocorreu com o bit "1" ocorre com os demais bits. Esse deslocamento que os bits de entrada apresentam a cada pulso de *clock* deu origem ao nome registrador de deslocamento.

4.4 Registrador de deslocamento para a esquerda

Para obter o registrador de deslocamento para a esquerda, basta mudar a ordem dos *flip-flops* e a entrada do registrador passará a ser no primeiro *flip-flop* da direita (figura 4.45).

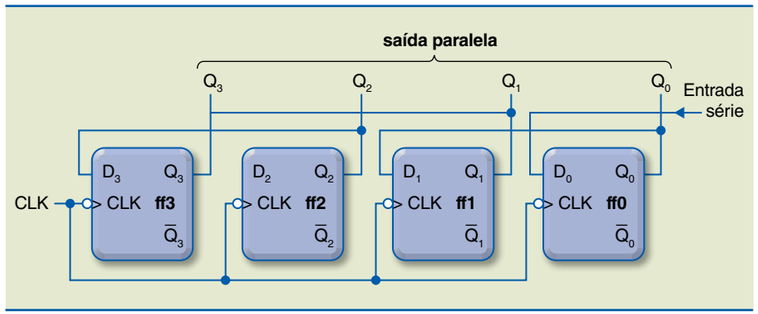


Figura 4.45
Registrador de deslocamento para a esquerda.

CI 7491 – Registrador de deslocamento de oito bits – entrada série e saída série (figura 4.46)

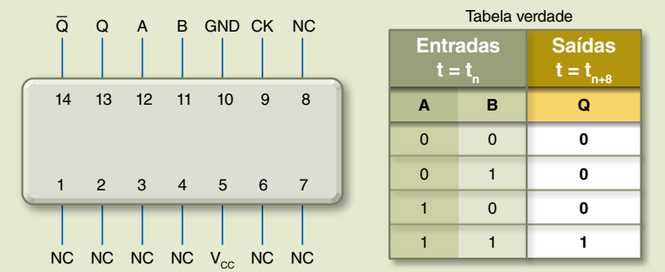


Figura 4.46
Registrador de deslocamento de oito bits: (a) identificação dos pinos do CI 7491, (b) tabela verdade e (c) detalhe do circuito interno do CI.

Se uma das entradas for "0", será transferido "0" para a saída do registrador após oito pulsos de *clock*, independentemente da outra entrada. Se uma das entradas for "1", o valor da outra entrada ("0" ou "1") será transferido para a saída do registrador após oito pulsos de *clock*. Podemos usar uma entrada como controle e a outra como entrada de dados.

CI 74164 – Registrador de deslocamento de oito bits com entrada de *reset* (figura 4.47)

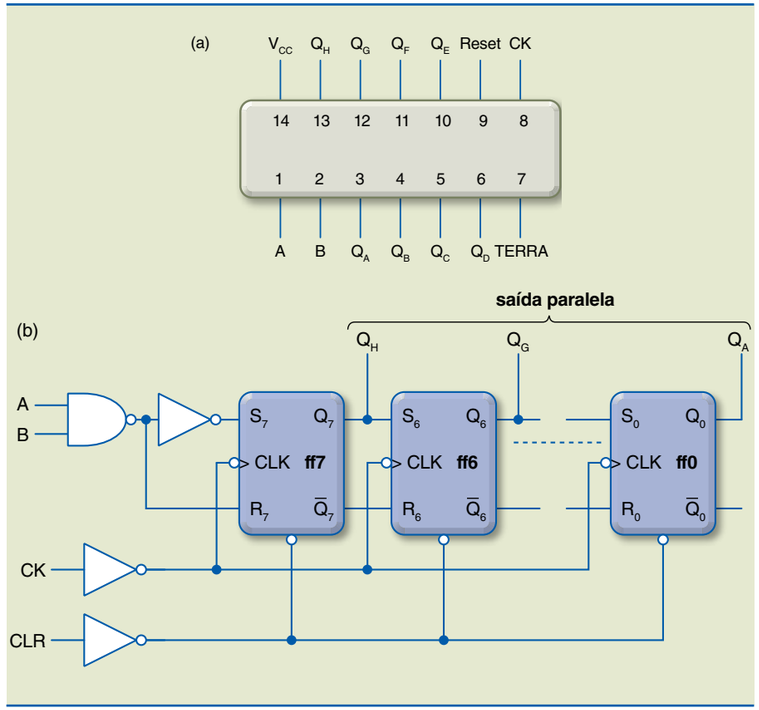


Figura 4.47
Registrador de deslocamento de oito bits: (a) identificação dos pinos do CI 74164 e (b) detalhe do circuito interno do CI 74164.

O CI 74164 é um registrador de deslocamento só para a direita, podendo ser utilizado como entrada série e saída série ou paralela. É sensível à transição negativa do *clock* e normalmente é usado com uma das entradas séries (A ou B) alta e os dados são enviados para outra entrada.

4.4.1 Circuito registrador de deslocamento – entrada série ou paralela

Os dados em paralelo transferidos para o registrador não devem ser colocados diretamente nas saídas dos *flip-flops*, pois com a ação do *clock* eles se deslocam, ocorrendo conflito. Assim, os dados em paralelo podem ser carregados no registrador pelo terminal *preset*.

Para obtermos um registrador com entrada paralela, necessitamos de *flip-flops* com *clear* e *preset*. Como sabemos, o terminal *clear* serve para colocar todas as saídas dos *flip-flops* internos em "0", ou seja, zerar ("setar") as saídas, e o *preset*, para colocar todas as saídas em "1" , ou seja, "setar" todas as saídas. O *clear* e o *preset* não podem estar ativos ao mesmo tempo, pois haveria conflito nas saídas.

Vamos avaliar como atuam o *clear* e o *preset* para que possamos obter um registrador com entrada paralela. Para isso, admitamos que o *clear* e o *preset* sejam ativos em "0".

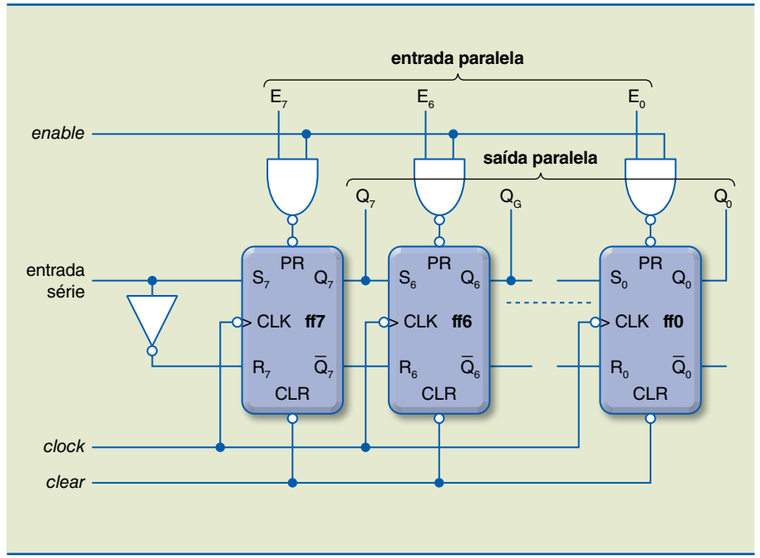
Primeiro, ativamos o *clear* (CLR = 0) zerando as saídas. Uma vez zeradas as saídas, desativamos essa função, dando condição de funcionamento normal ao registrador.

Colocamos bit a bit nos *presets* dos *flip-flops* a informação que corresponde à entrada paralela, ou seja, os terminais de *preset* estão sendo usados como entradas paralelas. Nos *presets* em que o valor colocado é "1", o *flip-flop* correspondente mudará a saída de "0" para "1" e será possível, portanto, transferir para o registrador os dados da entrada através dos *presets*. Isso feito, desativamos os *presets*, dando condição de funcionamento normal ao registrador (ver figura 4.48 – terminal *enable*).

Assim preparado, o registrador deslocará normalmente, com a ação do *clock*, os dados nele inseridos.

Vamos considerar o registrador de deslocamento para a direita da figura 4.43 e nele acrescentar *preset* e *clear* ativados em "0", com acesso possível ao *preset* de cada *flip-flop* interno. Os *clears* são interligados zerando simultaneamente, quando ativados, todos os *flip-flops* (figura 4.48).

Figura 4.48
Registrador de deslocamento – entrada série ou paralela; saída série ou paralela.



O terminal *enable* controla a função do PR, selecionando-o para ser entrada ou funcionamento normal de PR.

O terminal Q_0 ou outro da saída paralela pode ser considerado saída série, dependendo do atraso desejado na transferência do sinal ou outra condição específica do caso em questão. O referido atraso é aquele ocasionado na passagem do sinal da entrada do *flip-flop* interno para sua saída que leva um período de relógio.

Assim, um trem de pulsos na entrada se reproduzirá em Q_0 com atraso igual a (n 1)T, em que T é o período de relógio (*clock*).

4.4.2 Associação de registradores – registrador de maior capacidade

A figura 4.49 apresenta dois registradores entrada série, A_1 A_2 - - - - - D_3 D_4 com quatro bits de saída colocados em cascata para a obtenção de um registrador saída com oito bits.

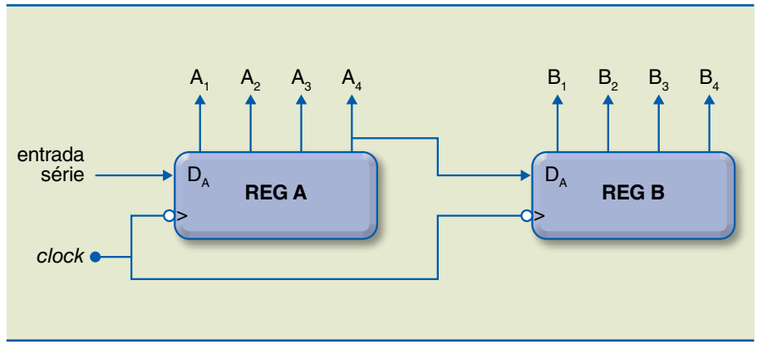


Figura 4.49
Dois registradores entrada série.

A saída série de REG A (A_4) é direcionada para a entrada série de REG B (D_4). O arranjo equivale a um registrador entrada série com saída paralela de oito bits. Podemos usar qualquer A_i ou B_i como saída série; a escolha dependerá do atraso desejado.

4.4.3 Registrador como multiplicador ou divisor por 2

Consideremos um número natural binário de oito bits, por exemplo: 1 1 0 0 0 1 1 0.

Vamos supor que esse número esteja carregado em um registrador de deslocamento e sofra deslocamento para a direita. Perde-se o bit "0" e fica indefinido o bit 7, que consideramos "0" (figura 4.50).

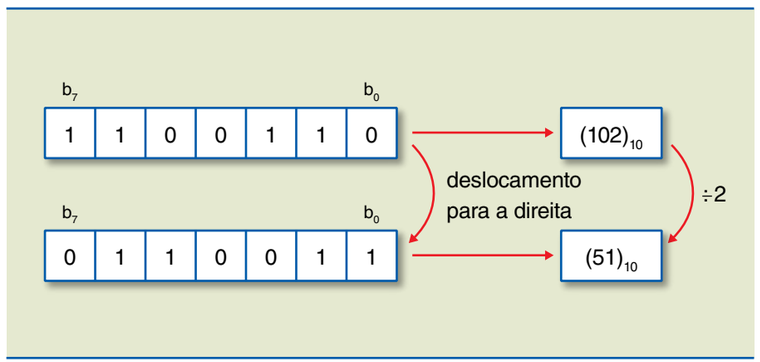


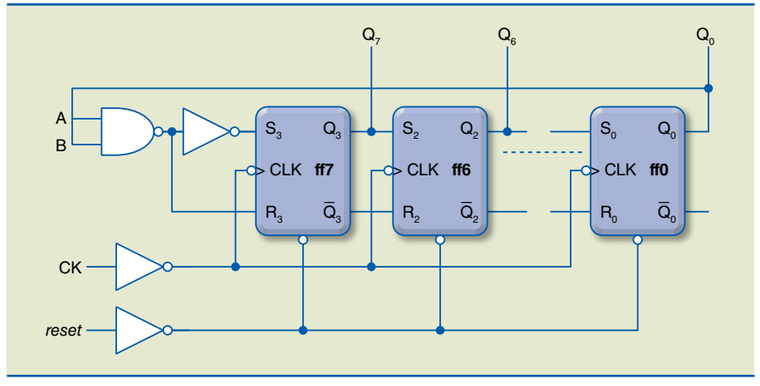
Figura 4.50
Registrador como multiplicador ou divisor por 2.

A operação de deslocamento para a direita, como vimos, pode ser associada à divisão por 2. Inversamente, um deslocamento para a esquerda pode ser associado à multiplicação por 2.

4.4.4 Registrador de deslocamento em anel

No circuito da figura 4.51, vamos conectar Q_0 à entrada série (A e B interligadas).

Figura 4.51
Registrador de deslocamento em anel de quatro bits.

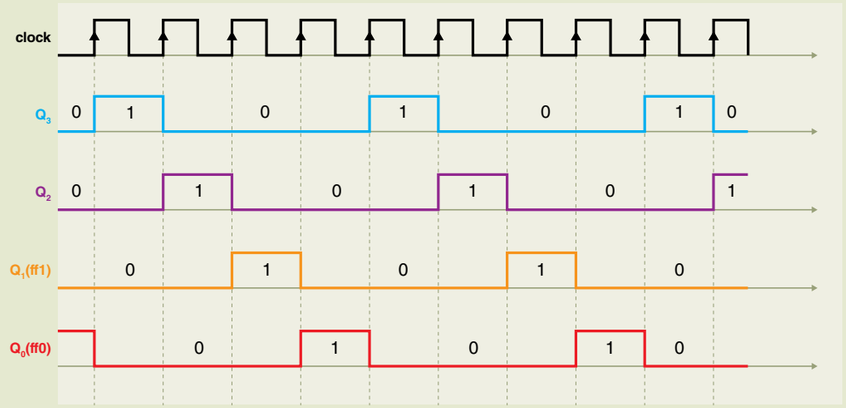


Vamos avaliar o circuito da figura 4.51, admitindo a condição inicial $Q_3 = Q_2 = Q_1 = 0$ e $Q_0 = 1$.

Na primeira transição positiva do *clock* (CK), Q_3 vai para "1", e Q_0, para "0"; as demais saídas permanecem em "0". Na segunda transição positiva, Q_2 vai para "1", e Q_3, para "0"; portanto, $Q_0 = Q_1 = 0$. Assim, "1" vai deslocando-se para a direita até $Q_0 = 1$ e as demais saídas = 0. Eventos distintos podem ser comandados (controlados) pelas saídas, obedecendo a uma sequência bem definida e em intervalos de tempo determinados pelo *clock*. Cada evento será comandado pela saída que estiver com valor "1".

Figura 4.52
Formas de onda de dois ciclos completos.

Na figura 4.52 estão registradas as formas de onda de dois ciclos completos.



Para concluir, observamos que, embora os *flip-flops* internos sejam sensíveis à transição negativa, devido ao inversor, é na transição positiva do *clock* que as mudanças ocorrem, lembrando também que, a partir do segundo *flip-flop*, o valor efetivo da entrada é o anterior à transição do *clock*, pois os *flip-flops* têm como base o *flip-flop* J-K mestre-escravo.

Capítulo 5

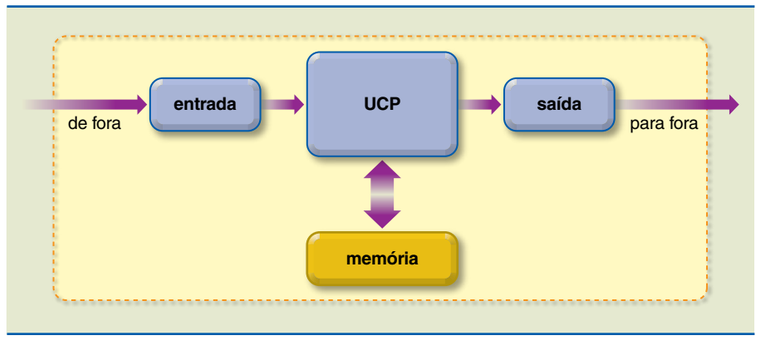
Sistemas microprocessados

Inicialmente, vejamos o conceito de computador, microprocessador e microcontrolador.

Computador

Computador é uma máquina que pode armazenar e processar informações. Hoje consideram-se computadores as máquinas digitais que operam com informações binárias. Basicamente, um computador é constituído de entrada, UCP (unidade central de processamento), memória e saída (figura 5.1).

Figura 5.1
Organização básica de um computador.



Registrador é um conjunto de posições (circuitos *flip-flops*) formando uma unidade que armazena uma informação binária. Os locais de memória contidos na UCP chamam-se registradores. Eles podem ser constituídos de diferentes números de bits, dependendo da quantidade de bits que armazenam.

UCP – Coordena todas as tarefas e executa os cálculos. É composta de três partes: unidade de controle, unidade lógico-aritmética (ULA) e um conjunto de registradores.

a) **Unidade de controle** – É de onde partem os sinais de controle de todo o sistema, estabelecendo a sincronização correta das tarefas que estão sendo realizadas.

b) **ULA** – É onde se realizam as operações lógicas e aritméticas determinadas pela unidade de controle. São operações aritméticas: subtrair, incrementar, setar bit etc. São operações lógicas: lógica "E", lógica "OU", comparação etc.

c) **Conjunto de registradores** – É constituído de registradores com várias finalidades, entre elas: contador de programa, armazenamento de dados processados pela UCP, armazenamento de endereços etc.

Entrada e saída – São compostas de todos os dispositivos que interligam as informações externas ao computador. É por meio desses dispositivos que podemos inserir informações no computador (entrada) ou receber informações dele (saída). São dispositivos de entrada: teclados, sensores, chaves etc. São dispositivos de saída: impressoras, motores, painéis etc. Os dispositivos de entrada e saída são ligados à UCP por interfaces apropriadas a cada dispositivo.

Memória – O conjunto de memórias é basicamente constituído de memórias RAM e ROM. As memórias RAM, por serem de leitura e escrita, armazenam dados que podem variar no decorrer do programa. As memórias ROM, apenas de leitura, armazenam dados fixos ou programas, ou seja, dados que não podem mudar durante toda a execução do programa.

5.1 Processadores

Os dois tipos de arquitetura mais comuns utilizados em microprocessadores e microcontroladores são Von-Neumann e Harvard.

A arquitetura Von-Neumann tem um único barramento por onde circulam os dados e instruções, tornando necessária maior quantidade de ciclos de máquina para executar uma instrução – uma simples soma de dois números, por exemplo, gasta três ciclos de máquina.

Essa arquitetura é utilizada na família de microcontroladores 8051. Nela, as instruções são estruturadas com base na tecnologia CISC, a qual envolve maior complexidade na execução da instrução. Comparada com a tecnologia RISC, usada na arquitetura Harvard, gasta mais ciclos de máquina.

Sigla em inglês de *complex instruction set computer*, computador com conjunto de instruções complexas.

Em geral, para uma mesma capacidade de processamento, o microprocessador com tecnologia CISC tem um *set* de instruções bem maior; assim, é possível executar programas menores com CISC. O mesmo programa em tecnologia RISC ficará bem maior, pois as instruções disponíveis em CISC não estão em RISC, sendo necessário criar as instruções faltantes com as RISC existentes.

Sigla em inglês de *reduced instruction set computer*, computador com conjunto de instruções reduzidas.

A arquitetura Harvard tem dois barramentos distintos, um para dados e outro para instruções, possibilitando maior rapidez no processamento – para fazer uma operação de soma de dois números, por exemplo, é necessário apenas um ciclo de máquina. Uma grande vantagem dessa arquitetura é o fato de que, enquanto uma instrução é processada, outra já pode estar executando seu ciclo de busca, carregando a próxima instrução. Tal processo de busca/execução é conhecido como *pipeline*. Essa é a arquitetura da família PIC, cujas instruções são estruturadas com base na tecnologia RISC, conforme já comentado.

Sigla em inglês de *programmable interface controller*, controlador de interface programável.

Microprocessador

O microprocessador MPU é um *chip* que substitui a UCP de um computador. As características técnicas do MPU basicamente definem as características do computador.

Sigla em inglês de *microprocessor unit*.

Antes dos microprocessadores, a UCP era construída com transistores e CI digitais discretos, o que tornava os computadores maiores e mais caros. Os microprocessadores de oito bits começaram a ser desenvolvidos em 1972. O pioneiro foi o 8008, que era capaz de endereçar 16 kB de memória e possuía 45 instruções e velocidade de 300 000 operações por segundo.

Com o passar dos anos, surgiram o 8080 (Intel), o 6800 (Motorola) e, em 1976, o "famoso" Z80 (Zilog), considerado na época um microprocessador com grande capacidade de processamento. O Z80 podia endereçar 64 kB de memória, possuía 176 instruções, além de executar todos os programas escritos para o 8080. Com essas vantagens, ganhou a preferência do mercado.

Microcontrolador

O microcontrolador (também designado por MCU) é um *chip* que contém, além da UCP, memórias RAM e ROM, oscilador interno de *clock*, I/O e outros recursos, o que o torna um verdadeiro computador em uma única pastilha. Atualmente, existe grande variedade de MCUs, como os das famílias 8051, PIC, COP, AVR etc.

O poder de processamento dos microprocessadores é maior que o dos microcontroladores. Por isso, os microprocessadores são usados em sistemas que necessitam de UCP mais sofisticada e com funções mais complexas (figuras 5.2 e 5.3).

Figura 5.2
Bloco microprocessador.

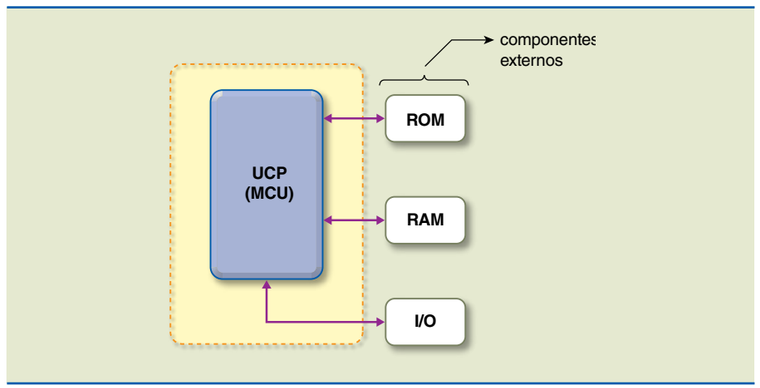
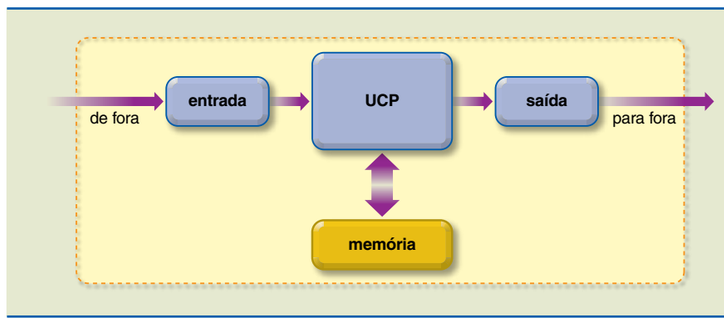


Figura 5.3
Bloco microcontrolador mínimo simplificado.



A figura 5.4 representa a estrutura interna de um microcontrolador constituído dos seguintes blocos internos:

- UCP
- Memória
- Entrada e saída
- Comunicação serial
- Temporizador
- *Watchdog*

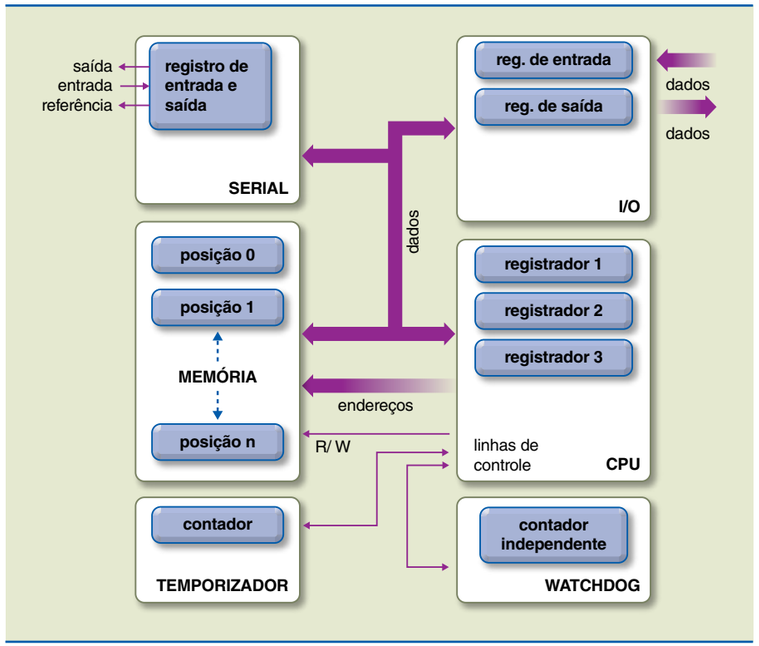


Figura 5.4
Microcontrolador: estrutura básica.

Comunicação serial – Nesse tipo de comunicação, são necessárias somente três linhas para enviar e/ou receber os dados: uma apenas para enviar dados, uma só para receber dados e uma para estabelecer o protocolo. Protocolo é a regra estabelecida para que receptor e emissor se entendam; assim, na terceira linha é colocado um sinal informando "quem" deve estar preparado para receber e "quem" deve estar preparado para enviar.

Temporizador – Permite usar intervalos de tempo para controlar eventos, informações de tempo gasto etc. A base dessa unidade é o contador, que é um registrador que aumenta em uma unidade seu valor em determinado tempo.

***Watchdog* (WDT)** – É um contador incrementado automaticamente com um *clock* independente. O *clock* do WDT provém de um temporizador RC só para ele; portanto, seu incremento independe do *clock* da máquina. Essa característica do WDT possibilita usá-lo para evitar o travamento do programa, da seguinte maneira: o tempo normal de "estouro" do WDT é de 18 ms; quando ocorre o estouro, é gerado um *reset* forçado e esse *reset* é usado para evitar o travamento. O programador, ao longo do programa, zera o WDT, evitando seu estouro e mantendo, assim, o desenvolvimento normal do programa. Nessa condição, caso ocorra

O estouro (*overflow*, em inglês) corresponde à situação em que o contador WDT atingiu o último estado de sua sequência de contagem. Ao fazer uso de um WDT, é necessário escrever o programa de maneira a redefinir o WDT com frequência suficiente (menor que 18 ms) para impedi-lo de atingir o último estado (estouro). Se isso acontecer é porque, por algum motivo, o programa travou.

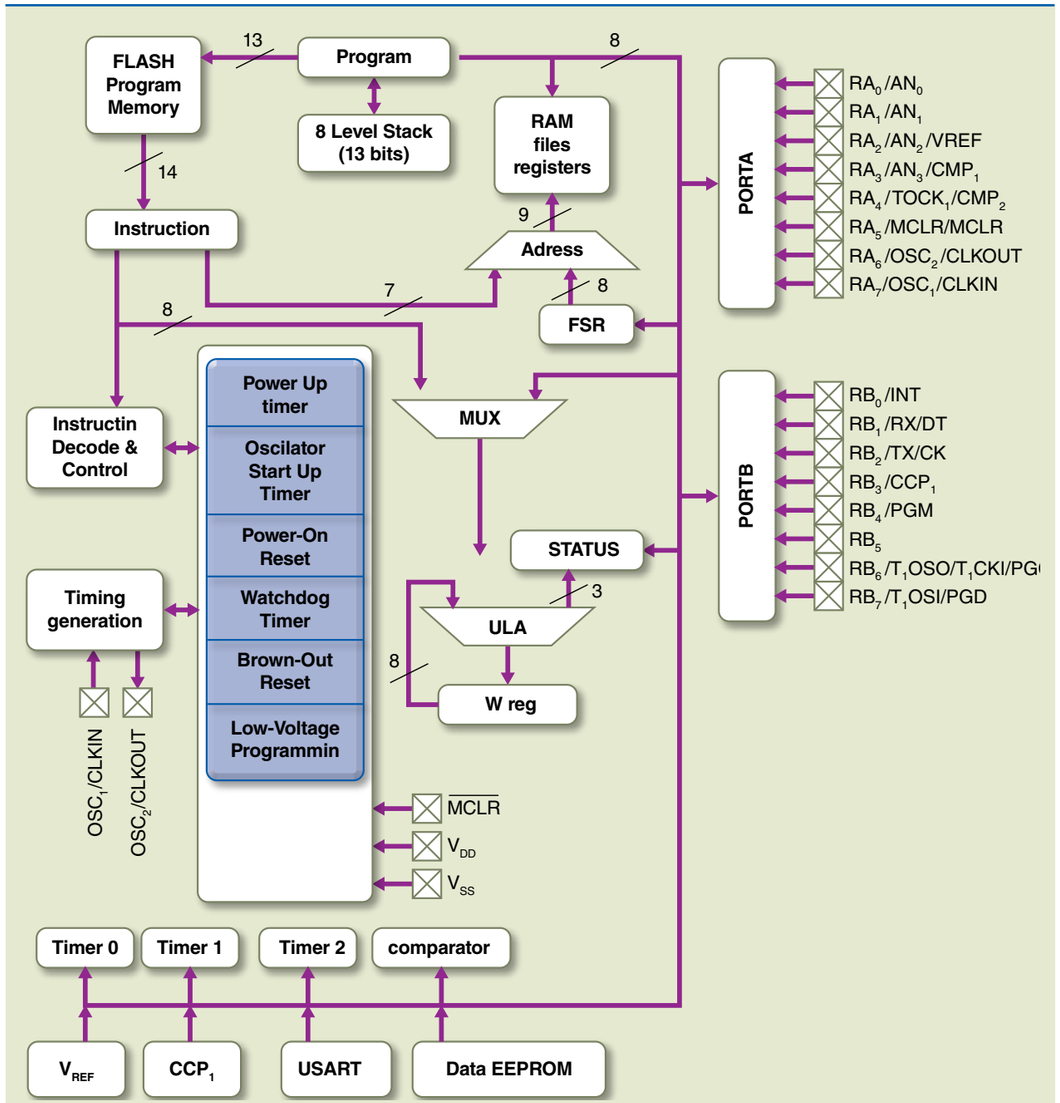
travamento, o WDT não é zerado e o WDT estoura, gerando um *reset* e saindo do travamento. O WDT pode ser desligado como opção de configuração.

Os demais blocos da figura 5.4 (UCP, memória, entrada e saída) foram explicados anteriormente. É importante ressaltar a ação das linhas de controle gerenciadas pela UCP existente entre ela e as unidades de memória, temporização e WDT.

5.1.1 Estrutura interna do PIC16F628A

A figura 5.5 apresenta a estrutura interna do microcontrolador PIC16F628A.

Figura 5.5
PIC16F628A: estrutura interna.



A estrutura interna apresentada na figura 5.5 encontra-se no *datasheet* da Microchip, que é sua fabricante. Observe que o registrador W (*work*) aparece em destaque e está diretamente ligado à ULA. A mais importante função desse registrador é sua utilização na troca de dados entre registradores, pois não é possível trocar diretamente dados entre registradores SFR e GPR, como veremos mais adiante.

A memória de dados é conectada a um barramento de oito bits, enquanto a memória de programa, devido à arquitetura Harvard, pode ser conectada a um barramento de 14 bits, possibilitando o processo de execução *pipeline*.

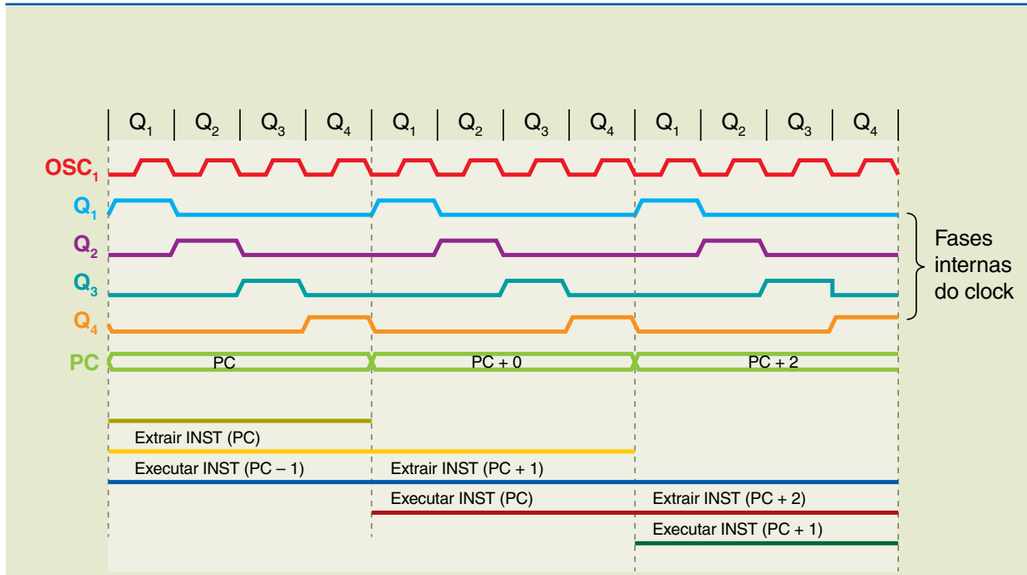
Na figura 5.5, notamos também blocos que representam circuitos (memórias, *timers*, comparadores, porta serial) que não são encontrados em microprocessadores, porém por ser um microcontrolador o modelo 16F628A inclui esses circuitos.

O PIC16F628A pode ter seu *clock* gerado por oscilador externo de no máximo 20 MHz ou por oscilador interno programado para 4 MHz ou 37 kHz. Em todos os casos, o *clock* gerado é dividido internamente por 4.

Vamos considerar um *clock* de 4 MHz. Após a divisão interna por 4, temos um *clock* interno de 1 MHz e, portanto, um período de 1 µs, que é o tempo denominado ciclo de máquina, no qual cada etapa da instrução é executada.

O diagrama da figura 5.6 consta do manual da Microchip.

Figura 5.6
Diagrama das fases internas do *clock* para o PIC16F628A.



Analisando a figura 5.6, podemos notar que durante um ciclo (composto de quatro fases: Q_1, Q_2, Q_3, Q_4), enquanto uma instrução é executada, a próxima é buscada para ser executada no ciclo seguinte; assim, cada instrução é executada em um único ciclo. Essa execução em *pipeline* é facilmente realizada pelo microcontrolador devido à arquitetura Harvard.

Em geral, as instruções são executadas em apenas um ciclo de máquina, exceto as que geram salto, como as chamadas de rotina e os saltos para outros endereços que não os da sequência normal do programa. Dessa maneira, para um *clock* externo de 4 MHz, temos uma instrução simples que leva 1 µs para ser executada e aquelas que produzem salto levam 2 µs.

5.2 Programação

Para que o processador execute tarefas, é necessário fazer a programação, especificando os passos que devem ser seguidos. Essa sequência é inicialmente descrita em um fluxograma e, depois, transformada em comandos de uma linguagem de programação.

5.2.1 Fluxograma

O fluxograma descreve a lógica do programa de acordo com a sequência em que as tarefas ocorrem; por isso, é uma ferramenta valiosa na execução do programa, qualquer que seja a linguagem utilizada. A figura 5.7 apresenta os blocos que usaremos para fazer o fluxograma de um evento.

Figura 5.7
Blocos do fluxograma.

	terminal	início ou fim do programa
	processo	o que deve ser executado (operações, cálculos etc.)
	sub-rotina	processo pré-definido
	decisão	testa determinada condição, dependendo do resultado segue um dos caminhos.
	conector	indica que o fluxograma continua em outro ponto.
	setas	indica o sentido do fluxo em que o evento está ocorrendo.

A figura 5.8 exemplifica a elaboração de um fluxograma para um evento simples: acender uma lâmpada sempre que um botão estiver pressionado.

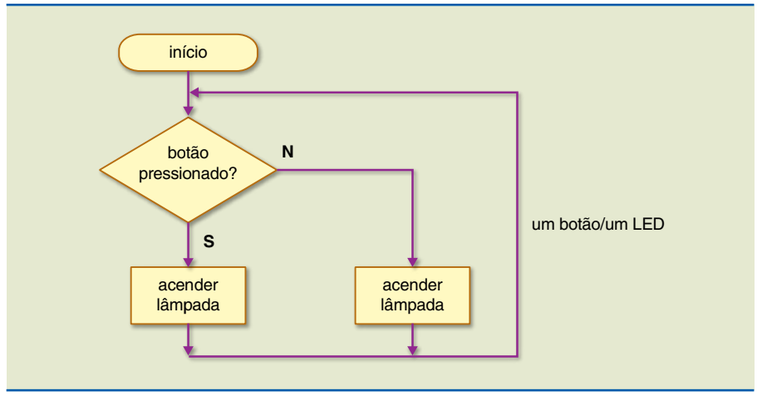


Figura 5.8
Fluxograma simples para acender uma lâmpada sempre que o botão estiver pressionado.

Exemplo

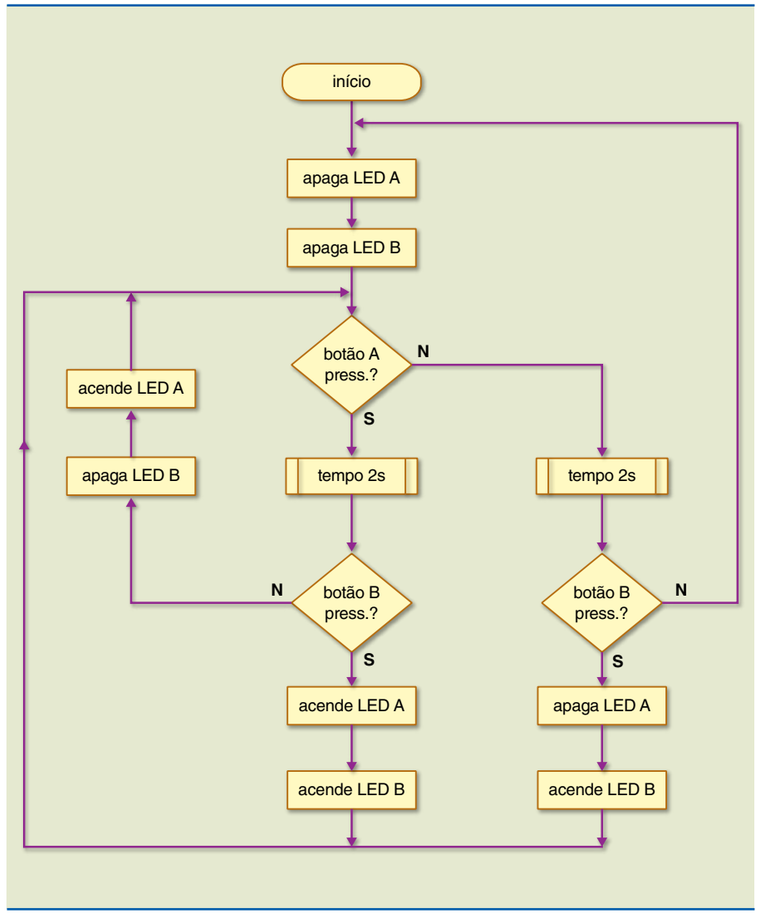
Faça o fluxograma do seguinte evento:

a) Inicialmente dois LEDs estão apagados.
b) Ao pressionar o botão A, mantendo o botão B solto, o LED A acende e o LED B permanece apagado.
c) Ao pressionar o botão B, mantendo o botão A solto, o LED B acende e o LED A permanece apagado.
d) Os dois botões pressionados acendem os dois LEDs.
e) Os dois botões não pressionados mantêm os dois LEDs apagados.
f) O segundo botão deve somente ser considerado pressionado se for pressionado no máximo 2 segundos após o primeiro.

Solução:

A figura 5.9 mostra o fluxograma solicitado.

Figura 5.9
Fluxograma dois botões/dois LEDs.



O bloco "tempo 2 s" é representado como sub-rotina, pois é um programa predefinido, ou seja, quando o programa principal necessita de um intervalo (*delay*) de 2 segundos, chama essa sub-rotina. A sub-rotina é executada e retorna à posição imediatamente seguinte no programa principal; assim, o programa da sub-rotina é escrito uma única vez.

5.2.2 Linguagens de programação

Linguagem de máquina ou **código objeto** é um conjunto de informações na forma binária dispostas em uma condição previamente definida de maneira que o microcontrolador possa processá-las.

Elaborar um programa dessa maneira é trabalhoso e cansativo, pois as instruções, os dados e todas as demais informações necessárias para a máquina devem estar em binário. A linguagem de máquina é própria de cada microcontrolador, sendo definida pelo fabricante; portanto, em geral, o código objeto de uma máquina não é compatível com o de outra.

A figura 5.10 mostra a aparência de um programa em linguagem de máquina.

endereços	dados
0000 0011 0011 0000	0000 1001
0000 0011 1101 0010	1001 0010
0000 0011 0110 0100	0100 1001
0000 0011 0001 1110	1000 0010

Figura 5.10
Aparência dos endereços e dados na linguagem de máquina.

Como podemos observar, um programa escrito desse modo pode se tornar confuso, de difícil compreensão, pois não indica o que o microcontrolador está executando, e a correção de erros é trabalhosa. Enfim, é uma linguagem somente para a máquina. Esse tipo de linguagem é chamado de linguagem de baixo nível.

Para facilitar o trabalho de programação e eliminar uma série de inconvenientes para o programador, foi desenvolvida a **linguagem *assembly***. Embora seja considerada linguagem de baixo nível, facilitou muito esse trabalho.

A linguagem *assembly* substitui códigos binários por mnemônicos, isto é, nomeia os códigos, permitindo fazer fácil associação com a função do código. O microcontrolador não entende diretamente um programa escrito em linguagem *assembly*; é necessário convertê-la em linguagem de máquina. O compilador *assembler* ou programa *assembler* faz essa transformação (figura 5.11).

Com o passar dos anos, surgiram linguagens classificadas como de alto nível. Elas apresentavam um conceito mais geral, podendo ser usadas para programar praticamente qualquer microcontrolador, independentemente de sua linguagem de baixo nível específica.

Para converter um programa fonte escrito em linguagem de médio ou alto nível em linguagem de máquina, utiliza-se um programa compilador (figura 5.12).

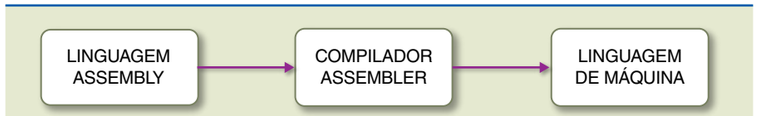


Figura 5.11
Compilador ***assembler***.

Figura 5.12
Programa compilador.

Existem diversos tipos de compiladores, dependendo da linguagem: compilador C, compilador Pascal etc.

O programa compilador transforma primeiro todo o programa em linguagem de máquina e depois o executa.

Existe também o programa interpretador, no qual cada instrução é transformada em linguagem de máquina e em seguida executada antes de a próxima instrução ser transformada. Portanto, a execução de um programa compilado é mais rápida que a de um programa interpretado.

Podemos testar e corrigir um programa por meio de programas que simulam sua execução; esse procedimento é chamado de depuração.

5.2.3 Linguagem *assembly*

A linguagem que utilizamos para nos comunicar com o microcontrolador é a linguagem *assembly*. O *assembly* é composto de um conjunto de instruções específico para cada processador. Ocorre, porém, que o sistema digital somente entende uma linguagem composta por "0" e "1", sendo necessário, portanto, que o código *assembly* seja traduzido para o código binário.

A linguagem *assembly* corresponde a um conjunto de regras e instruções necessárias para escrever um programa que será utilizado em determinada CPU, microprocessador ou microcontrolador.

Assembler é um programa que, executado em um sistema digital microprocessado ou microcontrolado, traduz o código escrito em linguagem *assembly* para um código equivalente de "0" e "1", ou seja, em linguagem de máquina.

O programa fonte é uma sequência de instruções escritas conforme as regras do *assembly* do processador ou qualquer outra linguagem de programação de microcontroladores (por exemplo: linguagem C para PICmicros), que normalmente é gravado em disco para ser carregado na RAM. Vamos considerar, por exemplo, a instrução *return* que um microcontrolador utiliza para retornar de uma sub-rotina. Quando o programa *assembler* faz a tradução, obtemos uma série de "0" e "1" correspondente a 14 bits que o microcontrolador PIC pode reconhecer, nesse caso: 00 0000 0000 1000.

Quando o programa *assembler* realiza a compilação do código fonte, também é gerado um arquivo com extensão "HEX". Esse nome provém de uma representação hexadecimal do programa em linguagem de máquina. Uma vez concluído o processo de montagem e compilação, o arquivo em código de máquina gerado é inserido no microcontrolador por um programador.

Um programa em linguagem *assembly* pode ser escrito originalmente em qualquer editor de textos e depois copiado na tela do editor do ambiente de programação, ou, então, editado diretamente no próprio ambiente de programação, como o MPLAB (ver Apêndice C).

O conjunto de instruções do microcontrolador PIC

Vejamos alguns termos utilizados no conjunto de instruções do microcontrolador PIC:

Work – Registrador acumulador temporário, representado pela letra **W**.

File – Referência a um registrador (posição de memória) representado pela letra **F**.

Literal – Um número qualquer em binário, decimal ou hexadecimal, representado por **L** nas instruções e por **K** nos argumentos.

Destino – Onde o resultado de uma operação será armazenado. Só existem duas possibilidades para o destino: em **F** – no registrador passado como argumento – ou em **W** – no registrador *work*. Também podemos representar "**0**" para **W** e "**1**" para **F** no destino.

Exemplo de como fazer a construção do nome das instruções

Somar uma unidade ao valor armazenado em um registrador chamado REGX.

Solução:

Incrementar (**INC**) registrador (**F**) REGX = **INCF REGX**

Tabela 5.1
Conjunto de instruções retirado do original em inglês.

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word				Status Affected	Notes
				Msb			Lsb		
Byte-Oriented File Register Operations									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word				Status Affected	Notes
				Msb			Lsb		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nobbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
Bit-Oriented File Register Operations									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1(2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit test f, Skip if Set	1(2)	01	11bb	bfff	ffff		3
Literal and Control Operations									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0xxx	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0100	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., MOVF, PORTB, 1) the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMRO register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

O conjunto de instruções dos microcontroladores PIC da Microchip que utilizaremos em nosso curso consiste em um pequeno repertório de apenas 35 instruções de 12 bits, que podem ser agrupadas em cinco grupos ou categorias. Essas cinco categorias são definidas de acordo com a função e o tipo de operandos envolvidos:

- Instruções que operam com bytes e envolvem algum registrador da memória interna.
- Instruções que operam apenas sobre o registrador W e que permitem carregar uma constante implícita ou incluída literalmente na instrução (literal).
- Instruções que operam sobre bits individuais dos registradores da memória interna.
- Instruções de controle de fluxo do programa, ou seja, aquelas que permitem alterar a sequência de execução das instruções.
- Instruções especiais cujas funções ou tipos de operandos são muito específicos e não se encaixam nas descrições anteriores.

Instruções que operam com registradores

Essas instruções podem ser de operando de origem simples ou duplo. O primeiro operando de origem será o registrador selecionado na instrução e o segundo, se existir, o registrador W. O destino, onde ficará armazenado o resultado, será o registrador selecionado ou W.

As instruções a seguir são operações lógicas de duplo operando:

- **ANDWF** **F,d:** operação lógica AND entre F e W, destino = W ou f
- **IORWF** **F,d:** operação lógica OR entre F e W, destino = W ou f
- **XORWF** **F,d:** operação lógica XOR entre W e F, destino = W ou f

Essas instruções correspondem a operações de simples operando:

- **MOVF** **F,d:** movimento de dados, destino = F ou W
- **COMF** **F,d:** complemento lógico, destino = F ou W
- **INCF** **F,d:** incremento aritmético, destino = F ou W
- **DECF** **F,d:** decremento aritmético, destino = F ou W

Nas sete instruções anteriores, o único bit afetado do registrador de *status* é o Z (bit de *zero*), que assumirá nível lógico "1" se o resultado da operação for "00000000" e nível lógico "0" se o resultado for qualquer outro valor.

As instruções de rotação de bits através do bit C (*carry*) do registrador de *status* são:

- **RLF** **F,d:** rotação de bits à esquerda, destino = F ou W
- **RRF** **F,d:** rotação de bits à direita, destino = F ou W

Nas operações Rotate Left File e Rotate Right File, os bits são deslocados de cada posição à seguinte, para a esquerda ou para a direita. O deslocamento é fechado, formando um anel com o bit C do registrador de *status*.

Nessas duas instruções, o único bit afetado do registrador de *status* é o bit C, que assumirá o valor que estava no bit "7" ou no bit "0", de acordo com o sentido de deslocamento.

A instrução seguinte realiza a troca de posição entre os quatro bits menos significativos e os quatro mais significativos (*nibble* baixo e *nibble* alto):

- **SWAPF F, d:** troca *nibbles*, destino = F ou W

A instrução SWAP File não afeta nenhum bit do registrador de *status*.

As duas operações que se seguem são a soma e a subtração aritméticas:

- **ADDWF F, d:** soma aritmética, destino = F ou D
- **SUBWF F, d:** subtração aritmética, destino = F ou D

As operações ADD WF e SUBtract W de F afetam três bits do registrador de *status*: C, DC e Z.

O bit Z assumirá nível lógico "1" se o resultado da operação for "00000000" e nível lógico "0" se o resultado for qualquer outro valor.

Os bits do registrador de *status* C e DC assumem o valor normal correspondente à soma de F com o complemento 2 de W. Dessa maneira, o significado para a operação de subtração resulta invertido, ou seja, C (*carry*) será "1" se não houver estouro na subtração (se o conteúdo de W for menor que o de F). O bit DC se comporta de modo similar, isto é, DC será "1" se não houver estouro na metade menos significativa, o que equivale a dizer que o *nibble* baixo do conteúdo de W é menor que o *nibble* baixo do conteúdo do registrador F.

As instruções a seguir são de simples operando, mas trata-se de casos especiais, pois o destino será sempre o próprio registro selecionado:

- **CLRF F:** zera todos os bits de F
- **MOVWF F:** copia conteúdo de W em F, destino = F

A instrução CLRF (CLeaR File) afeta somente o bit Z, que resulta sempre "0". A instrução MOVWF (MOVe W para F) não afeta nenhum bit do registrador de *status*.

Registrador de *status*

Esse registrador armazena o estado atual da unidade lógico-aritmética, do *reset* e do banco de memória utilizado.

Bit de escrita/ leitura	Bit de escrita/ leitura	Bit de escrita/ leitura	Bit de leitura	Bit de leitura	Bit de escrita/ leitura	Bit de escrita/ leitura	Bit de escrita/ leitura
IRP	RP1	RP0	TO	PD	Z	DC	C
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Bit 7 – IRP – Esse registro não é utilizado pelo PIC16F84 ou PIC16F628A, lido sempre como "0".

Bits 6, 5 – RP1, RP0 – Registros de seleção do banco de registros:

- 00 = banco 0 (00H – 7FH)
- 01 = banco 1 (80H – FFH)

Bit 4 – TO (*Time-out)*

- 1 = Após *power-up* (ligar), instrução CLRWDT (limpar cão de guarda) ou instrução SLEEP (dormir).
- 0 = Um *time-out* do WDT ocorreu.

Bit 3 – PD (*Power-down)*

- 1 = Após *power-up* (ligar) ou instrução CLRWDT (limpar cão de guarda).
- 0 = Execução da instrução SLEEP (dormir).

Bit *2* – Z (*Zero)*

- 1 = O resultado da operação aritmética ou lógica é igual a zero.
- 0 = O resultado da operação aritmética ou lógica não é zero.

Bit 1 – DC (Digito *Carry/borrow)*

- 1 = Ocorreu transbordamento do quarto bit menos significativo.
- 0 = Não ocorreu transbordamento do quarto bit menos significativo.

Bit 0 – C (*Carry)*

- 0 = Não ocorreu transbordamento do sétimo bit mais significativo.

Instruções que operam com registrador W e literais

Essas instruções se referem ao registrador W, isto é, um dos operandos de origem e o operando de destino serão sempre o registrador W. Nas instruções desse grupo que têm um segundo operando de origem, este será sempre uma constante incluída na instrução, chamada literal.

As instruções a seguir são as operações lógicas tradicionais, similares às que vimos anteriormente, porém realizadas entre a constante de programa e o registrador W:

- **IORLW** **K:** operação lógica OR entre W e L, destino = W
- **ANDLW** **K:** operação lógica AND entre W e L, destino = W
- **XORLW** **K:** operação lógica XOR entre W e L, destino = W

Nessas três instruções (Inclusive OR Literal W, AND Literal W e XOR Literal W), o único bit afetado do registrador de *status* é Z, que assumirá nível lógico "1" se o resultado da operação for "00000000" e nível lógico "0" se o resultado for qualquer outro valor.

A instrução que se segue é utilizada para armazenar uma constante no registrador W:

- **MOVLW** **K:** armazena constante em W, destino = W

A instrução MOVe Literal W não afeta nenhum bit do registrador de *status*.

A instrução seguinte (CLeaR W) é um caso especial da instrução CLRF, com destino = W:

- **CLRW:** zera todos os bits de W

Como na instrução CLRF, o único bit do registrador de *status* afetado é Z, que assumirá nível lógico "1".

As instruções são operações aritméticas entre W e uma constante.

- **ADDLW** **K:** soma W e K, destino = W
- **SUBLW** **K:** subtrai W de K

As operações ADD LW e SUBtract W de K afetam três bits do registrador de *status*: C, DC e Z.

Instruções que operam com bits

Essas instruções operam somente sobre o bit especificado; todos os outros bits do registrador não são alterados. Elas não têm especificação de destino, já que este será sempre o próprio registrador selecionado.

- **BCF** **F, b:** zera o bit b de F
- **BSF** **F, b:** "seta" o bit b de F

As instruções Bit Clear File e Bit Set File não afetam nenhum bit do registrador de status.

Instruções de controle

- **GOTO** **K:** salta para a posição K do programa

Essa é uma instrução típica de salto incondicional para qualquer posição de memória de programa.

A constante literal K corresponde ao endereço de destino do salto, isto é, o novo endereço de memória de programa a partir do qual serão executadas as instruções após a execução de uma instrução GOTO. Essa instrução simplesmente armazena a constante K no registrador PC (contador de programa).

A instrução seguinte é chamada de sub-rotina:

- **CALL** **K:** salta para a sub-rotina na posição K

O comportamento dessa instrução é muito similar ao da instrução GOTO, salvo que, além de saltar, armazena no *stack* o endereço de retorno da sub-rotina (que será usado pela instrução RETURN e RETLW). Isso é realizado armazenando no *stack* uma cópia do PC incrementado, antes que ele seja carregado com o novo endereço K.

As instruções a seguir são de retorno de sub-rotina:

- **RETURN:** retorno de sub-rotina

Retorna para o endereço de programa imediatamente posterior ao de chamada da referida sub-rotina.

- **RETLW** **K:** retorno de sub-rotina com constante K armazenada em W

Essa instrução (RETurn with Literal in W) permite o retorno de sub-rotina com uma constante literal K armazenada no registrador W. A operação que essa instrução realiza consiste simplesmente em retirar do *stack* um valor e carregá-lo no PC. Esse valor é o PC antes de realizar o salto incrementado, proveniente da última instrução CALL executada; portanto, será o endereço da instrução seguinte ao CALL.

- **RETFIE:** retorno de interrupção

Retorna de uma interrupção, recuperando o último endereço colocado no *stack*.

A seguir encontram-se as únicas instruções de salto (*skip*) condicional, os únicos meios para implementar desvios condicionais em um programa. Elas são genéricas e muito poderosas, pois permitem ao programa tomar decisões em função

de qualquer bit de qualquer posição de memória interna de dados, incluindo os registradores de periféricos, as portas de entrada/saída e o próprio registrador de *status* do processador. Essas duas instruções substituem e superam todas as instruções de saltos condicionais dos processadores convencionais (salto por zero, por não zero, por *carry* etc.).

- **BTFSC F, b:** salta se bit = 0
- **BTFSS F, b:** salta se bit = 1

A instrução BTFSC (Bit Test File and Skip if Clear) salta a próxima instrução se o bit *b* do registrador F for "0", e a instrução BTFSS (Bit Test File and Skip if Set), se o bit *b* do registrador F for "1".

Essas instruções podem ser usadas para realizar ou não uma ação conforme o estado de um bit, ou, em combinação com a instrução GOTO, para realizar um desvio condicional.

As instruções a seguir são casos especiais de incremento e decremento vistos anteriormente, mas estas afetam o fluxo do programa:

- **DECFSZ F, d:** decrementa F e salta se 0, destino = F ou W
- **INCFSZ F, d:** incrementa F e salta se 0, destino = F ou W

Essas duas instruções (DECrement File and Skip if Zero e INCrement File and Skip if Zero) se comportam de maneira similar a DECF e INCF, salvo que não afetam nenhum bit do registrador de *status*. Uma vez realizado o incremento ou decremento, se o resultado for "00000000", o processador saltará a próxima instrução do programa.

Essas instruções são utilizadas geralmente em combinação com uma instrução de salto (GOTO), para o projeto de ciclos ou laços (*loops*) de instruções que devem ser repetidas determinado número de vezes.

Instruções especiais

Nesse grupo estão as instruções que controlam funções específicas do microcontrolador ou que atuam sobre registradores especiais não endereçados como memória interna normal.

A instrução a seguir é a típica NO OPERATION, existente em quase todos os processadores:

- **NOP:** não faz nada, consome tempo

É utilizada apenas para introduzir um atraso (*delay*) no programa, equivalente ao tempo de execução de uma instrução. Não afeta nenhum bit do registrador de *status*.

A instrução seguinte "resseta" o contador do watch dog timer. Esse registrador não está acessível como memória e essa é a única instrução que o modifica.

- **CLRWDT:** "resseta" o *watch dog timer*

Essa instrução também afeta os bits PD (*power-down*) e TO (*time-out*) do registrador de *status*.

A instrução seguinte é um controle especial do microcontrolador que o coloca no modo *power down*, no qual: o microcontrolador suspende a execução do programa; o oscilador fica em estado constante; os registradores e portos conservam seu estado; e o consumo se reduz ao mínimo. A única maneira de sair desse estado é por meio de um *reset* ou por *time-out* do *watch dog timer*.

- **SLEEP:** coloca o MC em modo *sleep*

Essa instrução zera o bit PD (*power-down*) e "seta" o bit TO (*time-out*) do registrador de status.

Apêndice A

Famílias de circuitos integrados

Os circuitos integrados (CIs) que implementam as funções lógicas são construídos com pastilhas de silício. A maneira como a função é implementada fisicamente em cada CI define o que é denominado "família" de CIs.

As duas famílias principais são: TTL e CMOS.

A.1 Família TTL (transistor – *transistor logic*)

Os circuitos integrados TTL apresentam as seguintes vantagens:

a) baixo custo;
b) relativamente alta velocidade de operação (20 MHz típico);
c) disponibilidade comercial de centenas de tipos diferentes.

Existe uma linha comercial (74XXX) utilizável de 0 a 70 °C e uma linha militar (54XXX) utilizável de –55 a +125 °C. Os dígitos XXX determinam qual o tipo de TTL, por exemplo:

7400 → quatro portas NAND de duas entradas
74121→ multivibrador monoestável

Níveis de tensão e de corrente (74XXX)
Tensão de alimentação 5 V ± 5%
Máxima corrente de saída em nível baixo.....16 mA
Máxima corrente de saída em nível alto .....1,6 mA
Máxima tensão de entrada garantindo nível baixo .....0,8 V
Mínima tensão de saída garantindo nível alto .....2,4 V

As saídas TTL são maiores que 2,4 V e por volta de 3,3 V para alimentação de +5 V. Para termos uma saída TTL mais alta do que 3,3 V, conectamos um resistor (*pull up*) de 2,2 kΩ da saída para a alimentação de +5V.

Uma entrada TTL não conectada (flutuando) é reconhecida pelo CI como "1", mas esse procedimento não deve ser praticado devido a ruídos. Para que uma entrada permaneça em "1", temos de conectá-la diretamente à alimentação +5 V; para que permaneça em "0", ligamos ao terra.

Tipos de TTL

TTL padrão (normal)

Potência por porta: 10 mW
Nomenclatura: 74XXX (normal)
Velocidade de operação: 20 MHz (típico)

Fan-out (quantidade de entradas que podemos ligar na saída):

- 10 entradas TTL padrão
- 40 entradas TTL de baixa potência
- 6 entradas TTL de alta potência
- 6 entradas TTL Schottky
- 20 entradas TTL Schottky de baixa potência

TTL de baixa potência

Potência por porta: 1 mW
Nomenclatura: 74LXXX
Velocidade de operação: 3 MHz

Fan-out:

- 2 entradas TTL normais
- 10 entradas TTL de baixa potência
- 1 entrada TTL de alta potência
- 1 entrada TTL Schottky
- 5 entradas TTL Schottky de baixa potência

TTL de alta potência

Potência por porta: 22 mW
Nomenclatura: 74HXXX
Velocidade de operação: 50 MHz

Fan-out:

- 12 entradas TTL normais
- 40 entradas TTL de baixa potência
- 10 entradas TTL de alta potência
- 10 entradas TTL Schottky
- 40 entradas TTL Schottky de baixa potência

TTL Schottky

Potência por porta: 19 mW
Nomenclatura: 74SXXX
Velocidade de operação: 125 MHz

Fan-out:

- 12 entradas TTL normais
- 40 entradas TTL de baixa potência
- 10 entradas TTL de alta potência
- 10 entradas TTL Schottky
- 40 entradas TTL Schottky de baixa potência

TTL Schottky de baixa potência

Potência por porta: 2 mW
Nomenclatura: 74LSXXX
Velocidade de operação: 45 MHz

Fan-out:

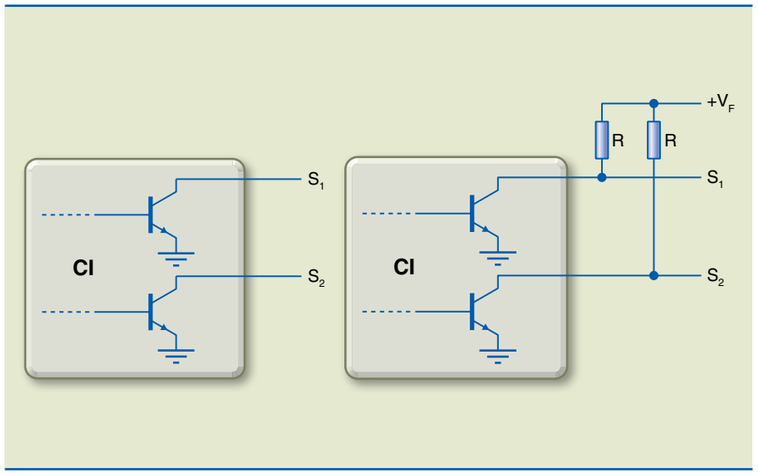
- 5 entradas TTL normais
- 20 entradas TTL de baixa potência
- 4 entradas TTL de alta potência
- 4 entradas TTL Schottky
- 10 entradas TTL Schottky de baixa potência

TTL – saída coletor aberto

Alguns CIs da família TTL têm suas saídas em "coletor aberto", isto é, o fabricante não completa o circuito internamente, deixando para que o projetista o complete externamente.

No circuito da figura A.1 estão representadas duas das saídas em coletor aberto de um TTL. Os transistores trabalham como chave (corte/saturação). Os resistores de coletor devem ser ligados externamente a uma fonte de tensão, que poderá ser a de alimentação do CI ou outra de valor diferente.

Figura A.1
Circuito com duas saídas coletor aberto de um TTL.



Se V_F = 5 V (alimentação do CI), o valor usual para R é 2,2 kΩ. No circuito da figura A.1, os resistores R podem ser substituídos por um único resistor R_1, conforme figura A.2.

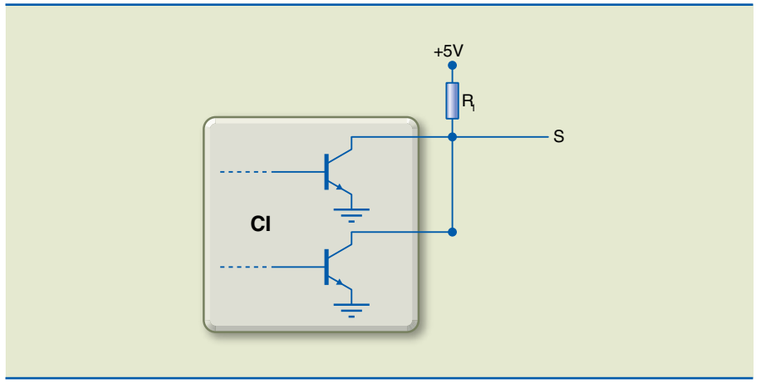


Figura A.2
Circuito com um resistor R_1.

Se V_F = 5 V, o valor usual para R_1 é 2,22 kΩ. Esse procedimento coloca em curto as duas saídas sem gerar conflito, pois, se ambos os transistores estiverem cortados ou saturados, seus coletores estarão no mesmo potencial e, portanto, o curto não trará problemas.

A saturação de um deles leva o coletor de ambos para "0" sem problema, devido ao resistor R_1. Se as saídas não fossem coletor aberto, não poderiam ser conectadas, pois haveria conflito de tensões. Analise o circuito da figura A.2 e conclua que a saída S é a saída de uma porta E, que tem como entradas as saídas interligadas do CI.

TTL – saída em alta impedância (*three-state*)

Alguns TTL têm um terminal para colocar a saída em alta impedância, o que, na prática, equivale a desconectar a saída do CI do circuito externo em que ela está conectada. Esse recurso é interessante nos casos em que vários CIs compartilham o mesmo barramento, evitando conflito no envio dos dados.

TTL – entrada em Schmitt trigger

O símbolo da figura A.3 é de uma porta inversora com entrada Schmitt trigger.

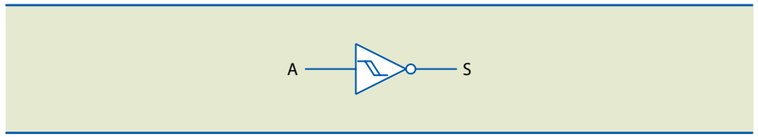


Figura A.3
Símbolo de porta inversora com entrada Schmitt trigger.

Schmitt trigger significa que a entrada tem dois valores diferentes para a transição lógica entre "0" e "1": um para valores ascendentes da tensão de entrada

e outro para valores descendentes da tensão de entrada. O valor que resulta na transição de “0” para “1”, tensão de entrada ascendente, é maior que o valor que resulta na transição de “1” para “0”, tensão de entrada descendente. Assim, uma entrada de tensão ascendente com ruído dentro de certos valores V_A, V_S limites não apresentará mudança indesejável na saída, desde que o valor de ruído não faça a entrada diminuir para o valor de transição de comutação para tensão descendente (figura A.4).

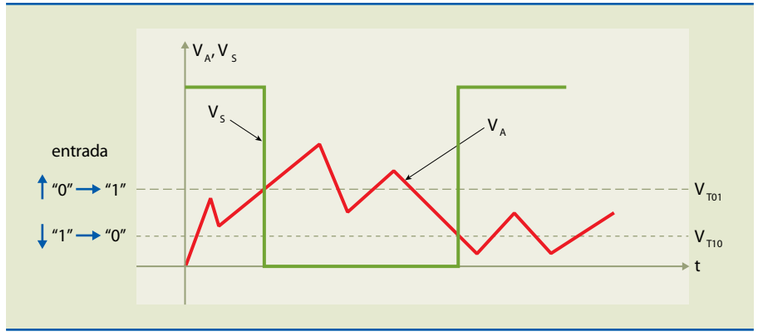


Figura A.4
Porta inversora com entrada Schmitt trigger – tensão de entrada com ruído.

Se V_A for uma tensão senoidal, obtemos uma onda quadrada com a mesma frequência da senoidal.

A.2 Família CMOS (complementary metal-oxide-semiconductor)

Em geral, a série CMOS normal (série 4000) tem velocidade menor que a dos TTLs, e a série H-CMOS apresenta velocidade equivalente à da série TTL normal. A tensão de alimentação da série 4000 e 74C é de 3 V a 15 V e faixa de temperatura de –40 a +85 °C.

Para a série 4000B, temos:

- corrente máxima na entrada em nível “0” → 1 µA
- corrente máxima na saída em nível “0” → 0,4 mA
- corrente máxima na entrada em nível “1” → 1 µA
- corrente máxima de saída em nível “1” → 0,4 mA

Para os CMOS, temos, em geral:

- “0” lógico → entre 0 V e 30% de V_{DD}
- “1” lógico → entre 70% de V_{DD} e V_{DD}

As versões mais recentes dessa família possuem internamente diodos de proteção para evitar a ação destrutiva da eletricidade estática. A potência dissipada é muito baixa, caracterizando uma grande vantagem da família CMOS.

Apêndice B

Conversores A/D e D/A

Os sinais que nos interessam são grandezas elétricas, em geral tensão em função do tempo. Os sinais podem ser analógicos ou digitais (figura B.1). Sinais analógicos são aqueles que variam continuamente com o tempo; portanto, entre dois valores distintos do sinal existem infinitos valores. Em um sinal digital, a variação do valor do sinal com o tempo não é contínua; entre dois valores distintos do sinal, o total de valores no intervalo é finito.

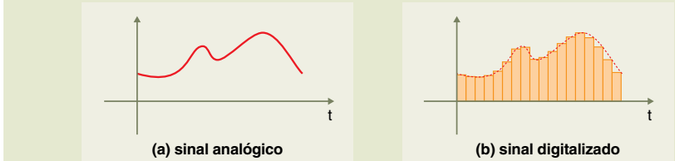


Figura B.1
(a) Sinal analógico e (b) sinal digitalizado.

Processar um sinal de modo totalmente analógico, dependendo do nível de qualidade exigido nesse processo, implica utilizar quantidade de componentes interligados de maneira complexa e muitas vezes apresenta resultado final insatisfatório.

Atualmente, devido à evolução da eletrônica, o processamento de um sinal é feito em sua forma digital. Assim, um sinal analógico é digitalizado e depois sofre a transformação necessária. A transformação desse sinal em sua forma analógica poderá ser feita ou não, dependendo do objetivo com que esse sinal foi modificado.

São exemplos de aplicação de processamento digital a compactação de uma informação analógica, a produção de eco em áudio (provocado pela defasagem do sinal e a soma do sinal defasado ao próprio sinal), a transmissão do sinal digitalizado em velocidade muito maior que a original e o uso de sensores com saída digital, minimizando as distorções da informação por ruídos.

A conversão de um sinal digital em analógico (D/A) e a de analógico em digital (A/D) é de fundamental importância no processamento de sinais, e é esse assunto que estudaremos neste apêndice.

B.1 Conversor digital-analógico

Quando necessitamos converter um sinal digital em analógico, usamos um circuito chamado conversor digital/analógico ou simplesmente D/A. Esse circuito recebe como entrada o sinal na forma digital codificado, em geral em binário comum ou no código BCD 8421, e o converte para um valor proporcional ao valor binário da entrada, (como mostra a figura B.2), em que **k** é a constante de proporcionalidade que está associada ao ganho do circuito conversor D/A. V_S é chamada de saída analógica do valor da entrada.

Figura B.2
Conversor digital/analógico.

No exemplo a seguir, consideramos **k = 0,4**.

A	B	C	D	V_S = k (ABCD)		A	B	C	D	V_S = k (ABCD)	
0	0	0	0	0V	(0,4 · 0)	0	1	0	1	2,0V	(0,4 · 5)
0	0	0	1	0,4V	(0,4 · 1)	0	1	1	0	2,4V	(0,4 · 6)
0	0	1	0	0,8V	(0,4 · 2)	0	1	1	1	2,8V	(0,4 · 7)
0	0	1	1	1,2V	(0,4 · 3)	1	0	0	0	3,2V	(0,4 · 8)
0	1	0	0	1,6V	(0,4 · 4)	1	0	0	1	3,6V	(0,4 · 9)

Nesse exemplo consideramos k = 0,4

Em um conversor D/A, a sequência de valores de saída resultante de uma sequência de valores digitais na entrada não é um sinal analógico, pois este não tem variação contínua com o tempo. Para obtermos uma saída analógica, devemos filtrar a saída, transformando-a em um sinal de variação contínua.

Quando há interesse em modificar um sinal analógico, muitas vezes é necessário convertê-lo para a forma digital, modificá-lo na forma digital e por fim convertê-lo para um sinal analógico. Esse processo é representado na figura B.3.

Consideremos um sinal analógico de frequência de 1 Hz. Um período desse sinal, portanto, é de 1 segundo. Se digitalizarmos esse sinal, o trem de bits que o representa poderá ser transmitido em muito menos tempo – por exemplo, em 1 ms. O sinal será transmitido com uma velocidade mil vezes maior que a transmissão em tempo real. Essa técnica é usada em telefonia digital, na TV digital e em muitas outras aplicações.

Em uma visão simplificada, podemos dizer que o conversor A/D prepara o sinal analógico por meio da conversão para digital a fim de que seja processado convenientemente e então entregue a um conversor D/A, possibilitando o retorno à forma analógica nas novas condições.

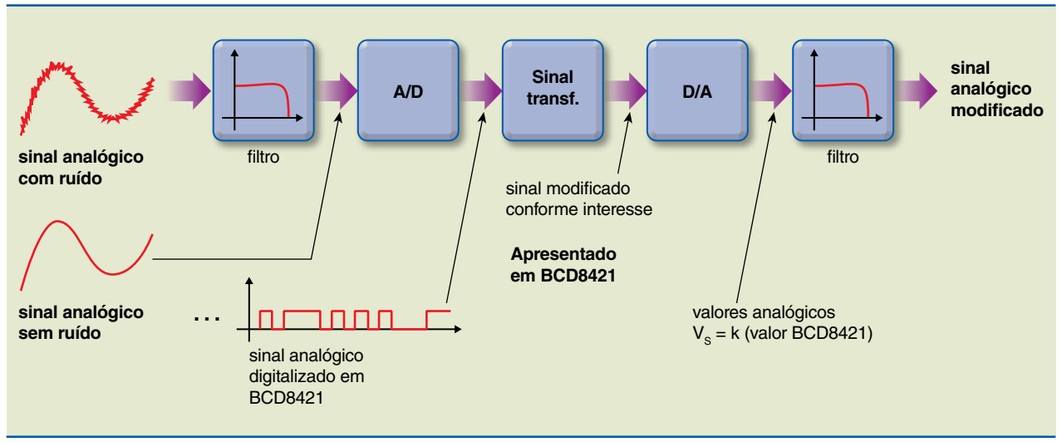


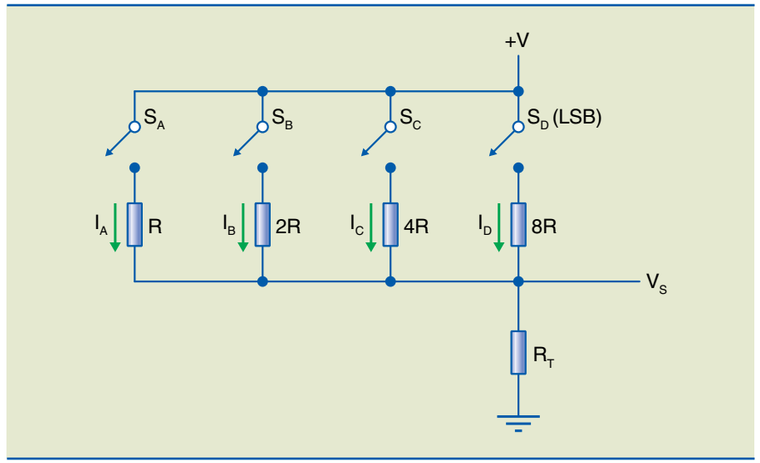
Figura B.3
Conversões em um sinal analógico com a finalidade de alterá-lo.

Processar o sinal digital, ou seja, a operação intermediária entre os conversores A/D e D/A, tornou-se de grande importância nos diversos ramos da eletrônica, e hoje existem componentes programáveis para essa finalidade, os DSPs (sigla em inglês de processadores digitais de sinais).

B.1.1 Conversor D/A com resistores de peso binário

Observe a figura B.4. As entradas digitais são definidas pelas chaves S_A até S_D. O valor "1" é o valor de V e o valor "0" é representado pela tensão 0 V.

Figura B.4
Conversor D/A com resistores de peso binário.



Usando a fórmula para cálculo do resistor equivalente de resistores em paralelo, temos:

$$\frac{1}{Req} = \frac{1}{R} + \frac{1}{2R} + \frac{1}{4R} + \frac{1}{8R} + \frac{1}{16R} + \ldots$$

Note que a fração seguinte é a anterior dividida por dois. Utilizando matemática para cálculo de progressões geométricas, temos:

$$\frac{1}{Req} = \frac{2}{R}$$

Portanto, $Req = \frac{R}{2}$ para o número de resistores tendendo a infinito.

Dessa maneira, para uma quantidade finita de resistores em paralelo nas condições estabelecidas, temos sempre a seguinte expressão:

$$\frac{R}{2} < R_{eq} < R$$

A conclusão $R_{eq} > R/2$ permite dimensionar o resistor R_T de modo que ele não influa no circuito. Como o resistor R_T está em série com o resistor equivalente Req, o valor de R_T deve ser muito menor que o de R_{eq}. Em geral, em eletrônica, dez vezes menor é bem menor e cem vezes menor é muito menor. Em nosso caso, a condição mais forte, de cem vezes, é importante. Assim, devemos garantir o correto funcionamento do circuito da figura B.4, independentemente do número de resistores $R_T << R/2$.

Exemplo

Calcule V_S no circuito da figura B.4.

Solução:

$$V_S = I_{RT} \cdot R_T$$

Como $R_T << (R/2)$, R_T pode ser desconsiderado no cálculo da corrente de cada resistor, estando a chave correspondente fechada ou aberta (considerando fechada para cálculo).

$$I_{RT} = (I_A + I_B + I_C + I_D) = \left(\frac{V}{R} + \frac{V}{2R} + \frac{V}{4R} + \frac{V}{8R}\right) \therefore V_s = RT$$

$$\left(\frac{V}{R} + \frac{V}{2R} + \frac{V}{4R} + \frac{V}{8R}\right)$$

$$V_s = \frac{R_T}{R}\left(V + \frac{V}{2} + \frac{V}{4} + \frac{V}{8}\right) \quad \therefore \quad V_S = \frac{R_T V}{R}(2^0 + 2^{-1} + 2^{-2} + 2^{-3})$$

A soma cujas parcelas são potências de 2 representa um número binário. Cada parcela existirá somente se a chave S_i correspondente ao resistor que dá origem a essa parcela estiver fechada. Da fórmula

$$V_S = \frac{R_T V}{R}(2^0 + 2^{-1} + 2^{-2} + 2^{-3}),$$

podemos afirmar que V_S é proporcional ao valor binário determinado pela posição de cada chave (aberta ou fechada).

A constante de proporcionalidade é: $\frac{R_T V}{R}$.

Assim, concluímos que o circuito da figura B.4 é um conversor D/A.

Consideremos os resistores e a tensão V no circuito da figura B.4 com os valores:

$R = 200\ k\Omega$; $R_T = 1\ k\Omega$ e $V = 10\ V$

Vamos antes comparar R_T com seu valor máximo, segundo o critério estabelecido.

$$R_T << \frac{R}{2} \quad \therefore \quad R_{T\ max} = \left(\frac{1}{100}\right)\frac{R}{2} \quad \therefore \quad R_{T\ max} = \frac{200k}{200} \quad \therefore \quad R_{T\ max} = 1\,k\Omega$$

$R_T = 1\ k\Omega$ satisfaz a condição.

$V_S = \frac{R_T V}{R}(2^0 + 2^{-1} + 2^{-2} + 2^{-3})$ chave fechada $0\ V \rightarrow 0$

chave aberta $10\ V \rightarrow 1$

S_A	S_B	S_C	S_D	decimal	V_S	
0	0	0	0	0	0 mV	
0	0	1	0	2	12,5 mV	$\leftarrow V_S = \frac{(1k)10}{200k}(0,25) = 12,5mV$
0	0	1	1	3	8,75 mV	$\leftarrow V_S = \frac{(1k)10}{200k}(0,375) = 18,75mV$
0	1	1	0	6	37,5 mV	$\leftarrow V_S = \frac{(1k)10}{200k}(0,75) = 37,5mV$
1	1	1	1	15	93,75 mV	$\leftarrow V_S = \frac{(1k)10}{200k}(1,875) = 93,75mV$

Ao verificarmos alguns valores, confirmamos que V_S é proporcional ao valor binário determinado pela chave, o que caracteriza o circuito como conversor D/A.

Vamos acrescentar à saída do circuito um amplificador de tensão e, com isso, ter mais liberdade de alterar a constante de proporcionalidade. Usaremos o 741C na configuração de amplificador de tensão inversor, conforme representado nas figuras B.5.

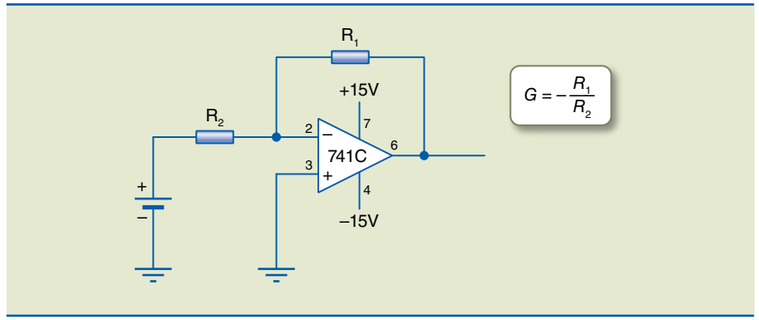


Figura B.5
Amplificador de tensão inversor.

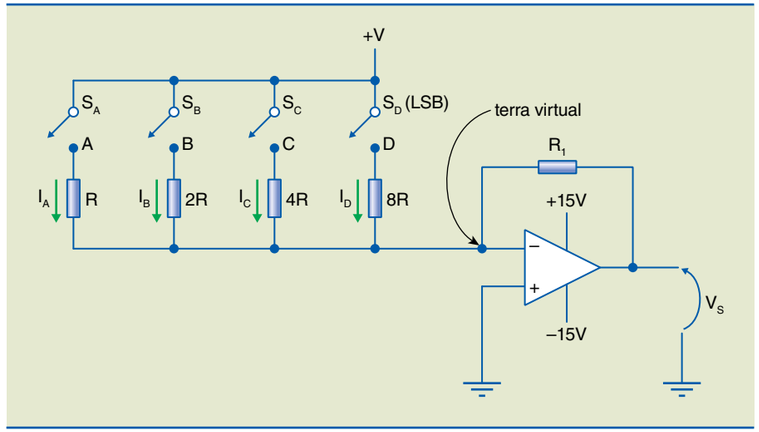


Figura B.6
Conversor D/A com amplificador de tensão inversor na saída.

No circuito da figura B.6

$V_S = -\left(\frac{R_1 V}{R} + \frac{R_1 V}{2R} + \frac{R_1 V}{4R} + \frac{R_1 V}{8R}\right)$, caso todas as chaves estejam ligadas.

Vamos manter os mesmos valores do circuito anterior, $R = 200\ k\Omega$ e $V = 10\ V$, e considerar $R_1 = 160\ k\Omega$, aplicando esses valores na seguinte expressão:

$$V_S = -\frac{R_1 V}{R}(2^0 + 2^{-1} + 2^{-2} + 2^{-3})$$

A fórmula é a mesma, mas o circuito da figura B.4 apresenta condição de valor máximo para R_T, limitando o valor da constante de proporcionalidade do

conversor. Para o circuito da figura B.6, o R_T na fórmula é substituído pelo resistor R_1 = 160 kΩ.

Os valores calculados para o circuito anterior, multiplicados por –160, valem para esse circuito em estudo. Para o valor binário "1", que não foi calculado, pegamos o valor calculado para o binário "2", dividimos por 2 e multiplicamos por –160, obtendo como resultado –1 V. Para o valor binário "15" pegamos o valor 93,75 do circuito anterior e multiplicamos por –160, obtendo –15 V. Como alimentamos o amplificador operacional 741C com fonte simétrica ±15 V, o máximo resultado confiável na saída é –13 V, pois na prática o amplificador operacional vai saturar.

Podemos substituir as chaves mecânicas por um contador de década, obtendo na saída do 741C uma tensão em escada (*staircase*), que varie de 0 a –9.

CI 7490 – contador de década (*decade counter*)

Observe o contador de década da figura B.7.

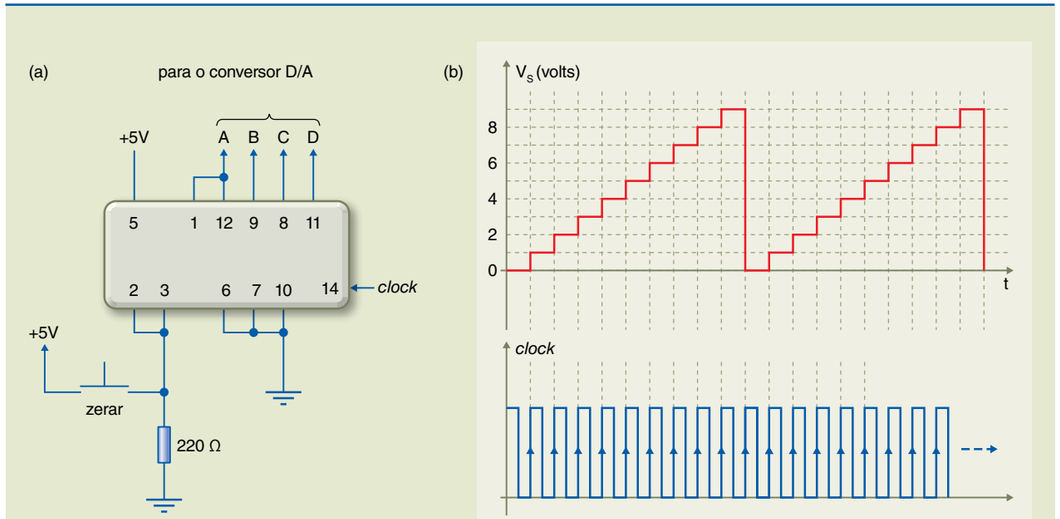


Figura B.7
(a) Circuito que substitui as chaves no circuito da figura B.6 e (b) tensão escada no circuito da figura B.6 com a substituição das chaves S_1 – o contador 7490 é sensível à borda positiva do *clock*.

Note, na figura B.7a, que as ligações no 7490 foram feitas de modo que ele conte até 9 e então zere, repetindo o ciclo indefinidamente. Na saída do conversor D/A, teremos a tensão escada de 0 a –9 V, como mostrado no gráfico da figura B.7b.

Nos conversores D/A com resistores de peso binário, como o que vimos até o momento (figura B.6), o resistor correspondente ao dígito MSB tem de suprir mais corrente que o correspondente ao LSB, e a relação entre essas correntes depende do número de dígitos de saída do conversor. Por exemplo, se tivermos um conversor de 10 bits, o terminal correspondente ao bit MSB deve fornecer 512 vezes mais corrente que o terminal correspondente ao bit LSB. De maneira geral, temos para um conversor de N bits:

$$I_{LSB} = \frac{V}{2^{N-1}R} \quad , \; I_{MSB} = \frac{V}{R} \quad \therefore \; \frac{I_{MSB}}{I_{LSB}} = 2^{N-1} \quad \therefore \; I_{MSB} = 2^{N-1} I_{LSB}$$

(ver circuito da figura B.4)

Essa situação, no conversor D/A com resistores de peso binário, dependendo do número de dígitos, pode inviabilizar o projeto. É bom lembrar que esse tipo de conversor tem a vantagem de usar somente um resistor por dígito.

B.1.2 Conversor D/A tipo escada R-2R

O circuito da figura B.8 utiliza dois resistores por bit, o dobro do circuito conversor D/A visto anteriormente. No entanto, ele apresenta a vantagem de cada terminal correspondente a um bit fornecer o mesmo valor de corrente, independentemente do número de dígitos do conversor.

A figura B.8 apresenta a rede de resistores usada no conversor D/A tipo escada R-2R.

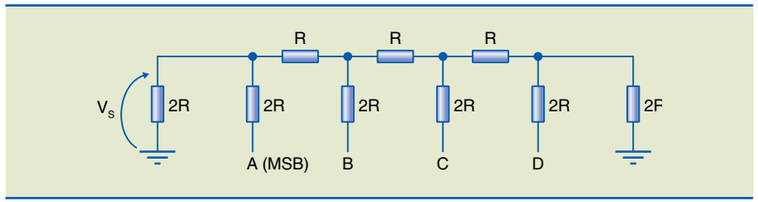


Figura B.8
Rede de resistores no conversor D/A tipo escada R-2R.

Na figura B.9, vamos considerar o binário de entrada 0010, ou seja, somente C = 1 (tem tensão +V).

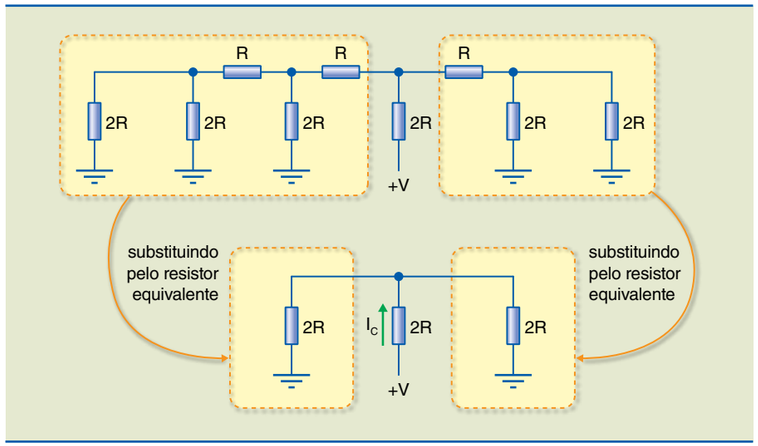
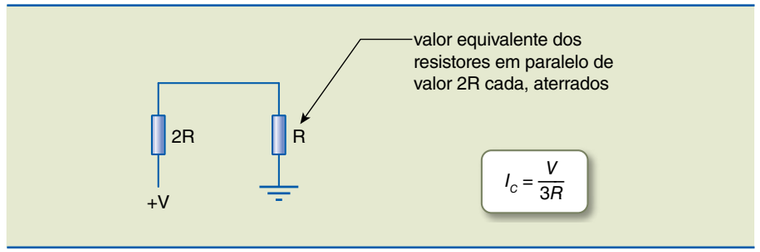


Figura B.9
Circuitos com resistores equivalentes.

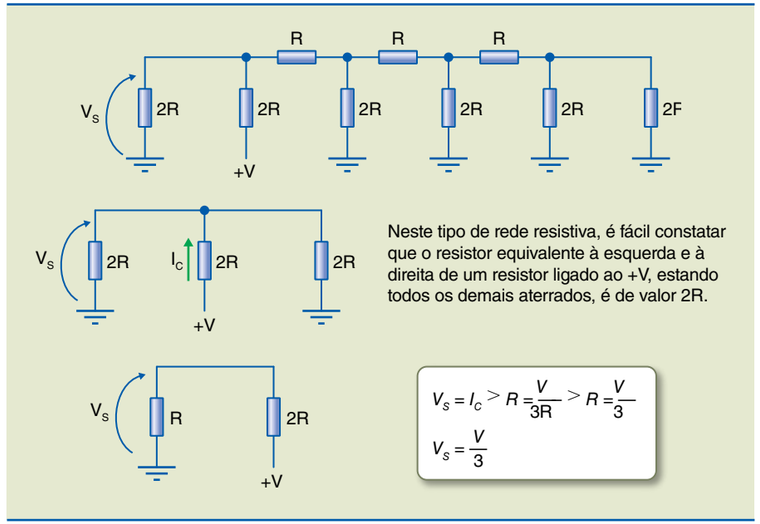


O valor encontrado para I_C equivale ao maior valor de corrente que um terminal correspondente a um bit fornece. Tal condição ocorre quando esse for o único bit com valor "1". Quando um ou mais bits estiverem com valor "1", a corrente fornecida pela terminal será igual para esses terminais e será menor que o valor encontrado para um único terminal ligado ao bit de valor "1".

A condição necessária de fornecimento de corrente pelos terminais correspondentes a bits no conversor D/A tipo escada é bem mais vantajosa que a necessária no conversor D/A com resistores de peso binário, conforme concluímos da análise feita.

O bit MSB de entrada está posicionado ao lado da tensão de saída na rede resistiva. Na figura B.10, vamos verificar o valor analógico de saída do conversor para o binário $1000 = (8)_{10}$ voltando ao circuito da figura B.8.

Figura B.10
Três modelos de redes resistivas.



Esse valor corresponde à saída para $1000 = (8)_{10}$, portanto:

$$\frac{V}{3} = k \cdot 8 \quad \therefore \quad k = \frac{V}{24}$$

em que k é a constante de proporcionalidade do conversor.

Verificamos somente para o binário 1000, mas pode-se afirmar que

$$V_S = \left(\frac{V}{24}\right) \cdot (\text{valor binário de entrada})$$ para todo valor binário de entrada.

Como podemos constatar, os valores das resistências não influem no valor de k, sendo a relação precisa entre elas (R e 2R) o fator mais importante. Se $V = 24$ V, temos $k = 1$; nessa condição, o valor de saída V_S corresponde ao equivalente decimal do binário de entrada.

Vamos acrescentar um amplificador de tensão na saída da rede de resistores do conversor, conforme circuito da figura B.11.

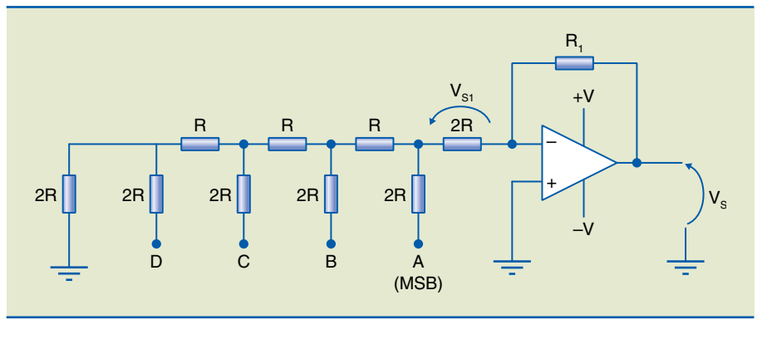


Figura B.11
Amplificador de tensão na saída da rede de resistores do conversor.

Analisando a figura B.11, podemos notar que a rede resistiva é exatamente a mesma, porém o resistor de $2R$ da direita está ligado ao terra virtual do amplificador operacional. Portanto, a tensão de saída da rede é a indicada em cima do $2R$ e o bit MSB é o mais próximo a esse resistor.

$$V_S = -V_{S1} \cdot \frac{R_1}{2R} = -\frac{V}{24} \cdot (\text{valor binário da entrada}) \cdot \frac{R_1}{2R} =$$

$$= -\frac{VR_1}{48R} \cdot (\text{valor binário da entrada})$$

$$V_S = -\frac{VR_1}{48R} \cdot (\text{valor binário da entrada})$$

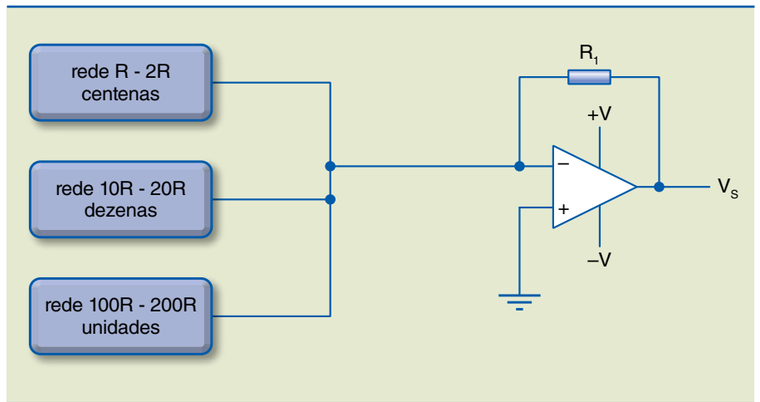
em que V é a tensão aplicada aos resistores correspondentes aos bits de entrada.

Observe na equação anterior que por meio do resistor R_1 podemos ajustar o ganho do amplificador, definindo a constante de proporcionalidade do conversor.

No caso de conversão de binários em código BCD 8421, a conversão da casa das unidades pode ser feita ajustando a constante de proporcionalidade de maneira que as saídas de 0 até –9 V correspondam aos binários de 0 até 9 em decimal. É importante ressaltar que o amplificador de tensão inversor soma os valores de

tensão ligados a sua entrada; assim, podemos construir as redes de resistores de modo a somar os valores da casa das unidades com os da casa das dezenas, centenas e assim por diante, desde que o ganho do amplificador corresponda à casa correta, tendo como referência a casa das unidades (figura B.12).

Figura B.12
Redes de resistores de modo a somar os valores da casa das unidades com os da casa das dezenas e centenas.



Para o bloco das unidades, o amplificador tem ganho

$$G = -\frac{R1}{200R},$$

pois, observando a rede de resistores, o resistor de 200R é aquele conectado ao terra virtual do amplificador operacional, fazendo parte do aumento de tensão deste.

O bloco das dezenas tem seu ganho multiplicado por 10 em relação ao das unidades, pois o resistor ligado ao terra virtual vale 20R, e o bloco das centenas tem seu ganho multiplicado por 100 em relação ao bloco das unidades.

B.2 Conversor analógico-digital

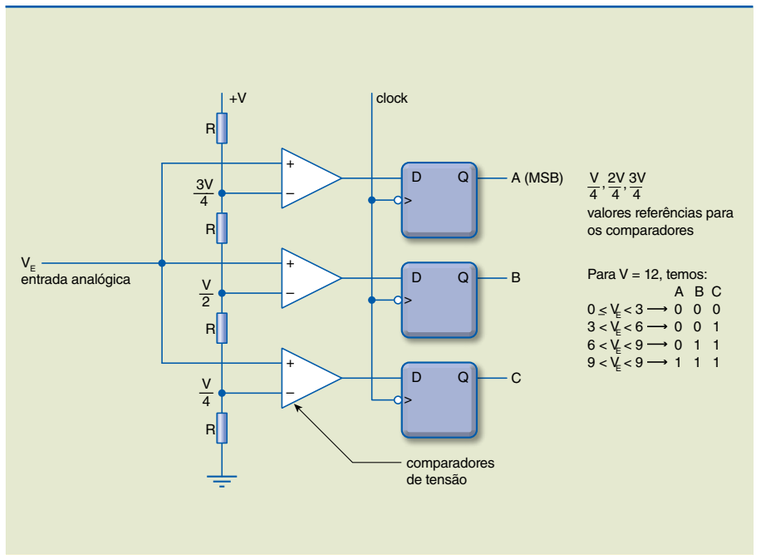
Converter analógico em digital consiste em passar o valor de uma tensão analógica para um valor digital equivalente. Esse processo é basicamente um problema de amostragem do sinal, ou seja, medir periodicamente o sinal que queremos digitalizar e apresentar os valores medidos na forma digital. A taxa com que se repetem as medidas é chamada de frequência de amostragem. É intuitivo que, quanto maior for a frequência de amostragem, mais precisa será a reprodução do sinal em sua forma digital.

A frequência mínima de amostragem é, segundo o teorema de Nyquist, duas vezes maior que a frequência existente no sinal a ser digitalizado. Para melhor digitalização do sinal, devemos amostrá-lo em uma frequência de amostragem dez vezes maior que a citada no teorema de Nyquist.

B.2.1 Conversão A/D – usando comparadores

Nessa conversão, usamos uma série de comparadores, sendo uma das entradas para todos os comparadores o valor analógico que queremos digitalizar (figura B.13).

Figura B.13
Série de comparadores.



Com o aumento do número de bits, aumenta na mesma proporção o número de comparadores, o que pode tornar o processo de conversão inconveniente para uma quantidade de bits maior que quatro. Esse tipo de conversor é bastante rápido.

B.2.2 Conversor A/D – usando contador e conversor D/A

Observe a figura B.14.

O contador de décadas e o conversor analógico geram uma tensão escada (ver figura B.7b) que é a entrada referência para o comparador. O comparador permanece com a saída "1" enquanto a tensão escada não atinge o valor da entrada analógica.

Nessa condição, a saída do comparador permite a passagem do *clock* para o contador, através da porta E, resultando em avanço na tensão escada. Quando o valor da tensão escada atinge o valor analógico de entrada, o comparador vai para "0" e a saída do contador permanece no valor digital correspondente ao valor analógico de entrada, pois o *clock* fica bloqueado na porta E com o "0" do comparador.

Ao ir para "0", a saída do comparador dispara o *clock* dos *flip-flops*, transferindo o valor digital do contador para a saída. A partir desse instante, o sistema não progride, pois não há alteração na saída do comparador para que os *clocks* sejam acionados. A reinicialização do processo é feita zerando o contador.

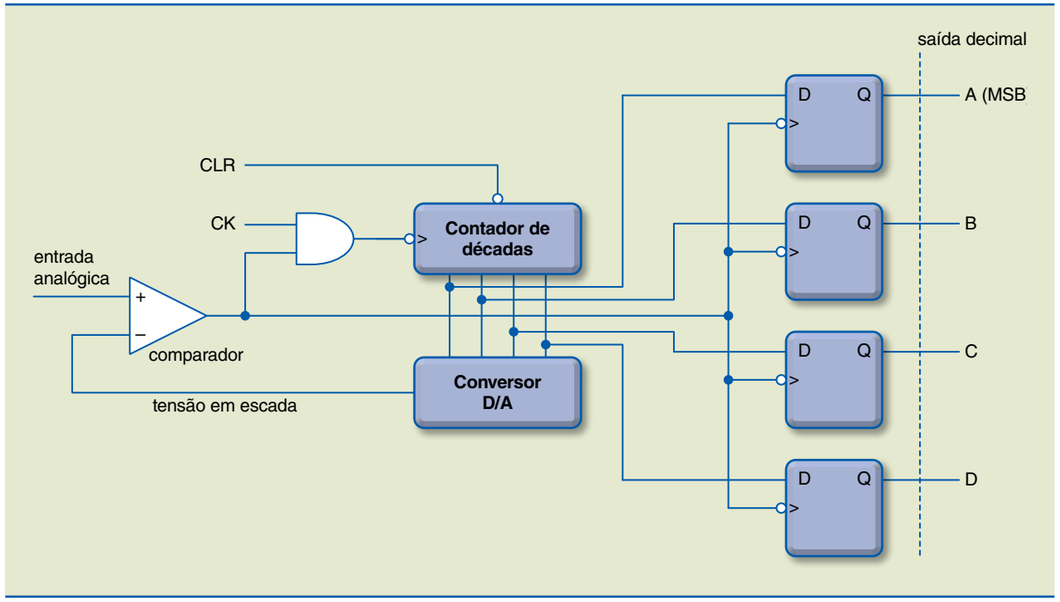


Figura B.14
Conversor A/D – usando contador e conversor D/A.

Podemos transformar o circuito da figura B.14 em um simples voltímetro digital adicionando um decodificador, um *display* e um *clock* conveniente para zerar o contador de década automaticamente. A frequência de *clock* do contador determinará o tempo de atualização do valor da tensão mostrada no *display*.

Apêndice C

MPLAB

O MPLAB é uma importante ferramenta no desenvolvimento de programas com PIC, pois o gerenciamento de projetos, a compilação, a simulação e a gravação são executados em um mesmo ambiente.

Vamos considerar que o MPLAB já esteja instalado, lembrando que ele está disponível gratuitamente no *site* da Microchip: <http://www.microchip.com> (figuras C.1 até C.3).

Figura C.1

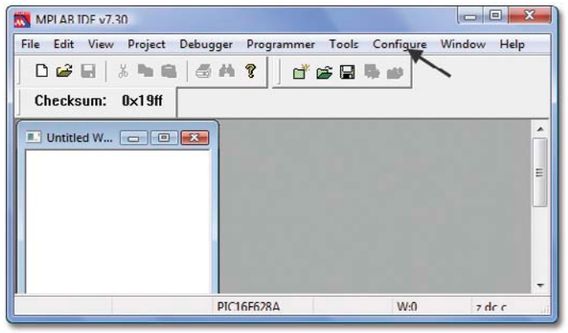


Figura C.2

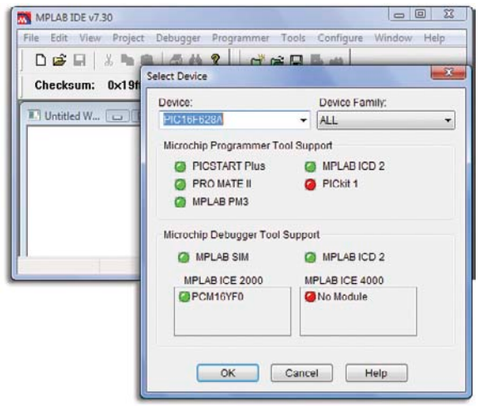
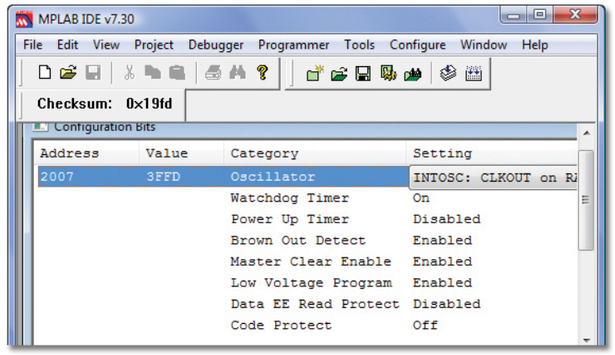


Figura C.3



Assim que abrimos o MPLAB, a tela 1 é exibida (figura C.1). Inicialmente, selecionamos o *chip* com que vamos trabalhar (PIC16F628A) utilizando o comando **Configure > Select Device** (figura C.2). Uma vez selecionado o PIC, no comando **Configure > Configuration Bits** (figura C.3), configuramos os *hardwares* internos na posição de endereço 2007 da memória de programa. Essa configuração é estabelecida conforme as condições do projeto e de acordo com o comportamento do microcontrolador.

C.1 Criação de um projeto

Para criar um projeto, usamos o comando **Project > New**. Na tela **New Project**, em **Project Name**, damos a ele um nome (figura C.4) e, em **Project Directory**, especificamos o diretório (figura C.5). Se o nome dado não é de um diretório existente, o programa solicita permissão para criá-lo.

Figura C.4

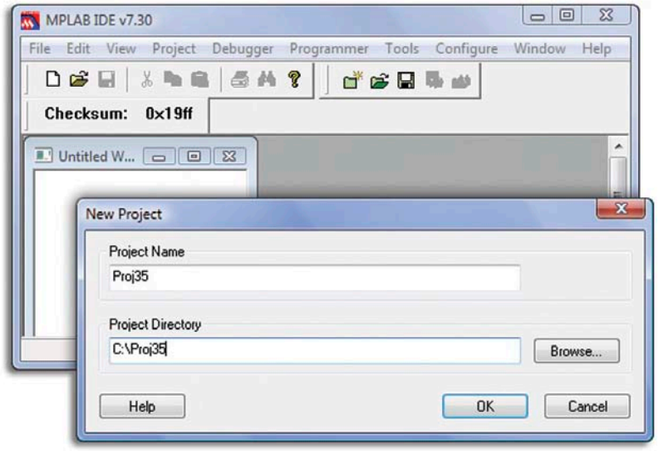
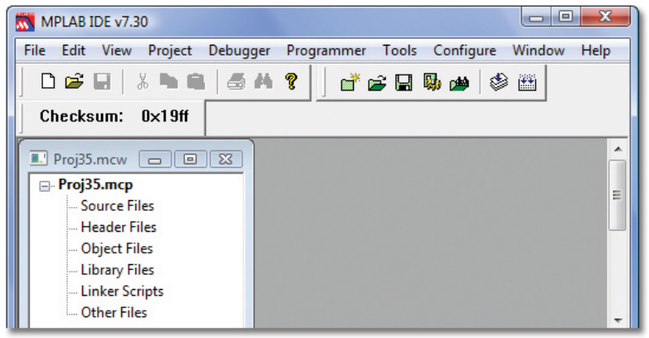
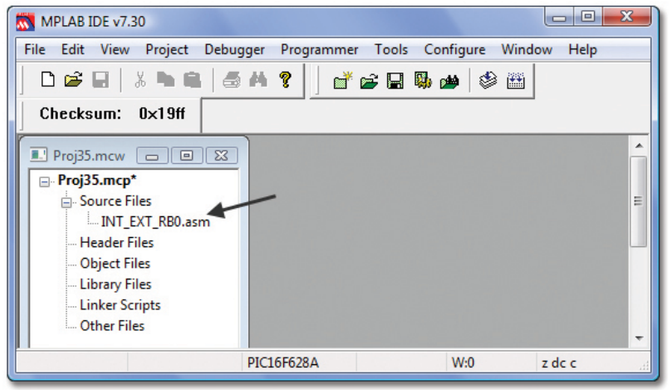


Figura C.5



Agora, podemos criar o **arquivo** de trabalho com extensão .asm. Para isso, adicionamos o arquivo (programa) selecionado ao projeto, que passa a ter extensão .asm. Então, usamos o comando **Project > Add Files Project**, e a tela reproduzida na figura C.6 é exibida.

Figura C.6

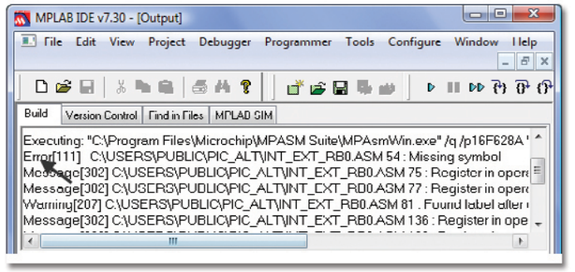


Observando a figura C.6, percebemos que o programa selecionado foi INT_EXT_RBO, que passa a ter no Proj35 extensão .asm.

C.2 Compilação

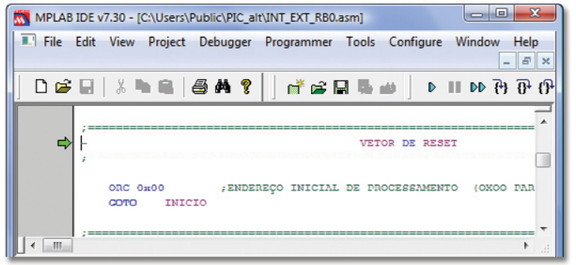
Compilar um programa significa passar as instruções em linguagem *assembly* (asm) para binários, para que possam ser interpretadas pelo microcontrolador. Para compilar um programa, devemos usar o comando **Project > Set Active Project > None (Quickbuild Mode)** e, em seguida, **Project > Quickbuild** (seguido do nome do programa). Caso existam erros no programa, abre-se uma janela indicando quais são os erros (figura C.7).

Figura C.7



Clicando na linha do erro, volta-se para o programa *assembly*, no qual é indicada a linha do erro (figura C.8).

Figura C.8



C.3 Simulação

Agora vamos testar o programa. Para isso, chamamos o programa, usamos o comando **Debugger > MPLAB SIM** e configuramos o *clock* para 4 MHz utilizando o comando **Debbuger > Setting** (figuras C.9 e C.10).

Figura C.9

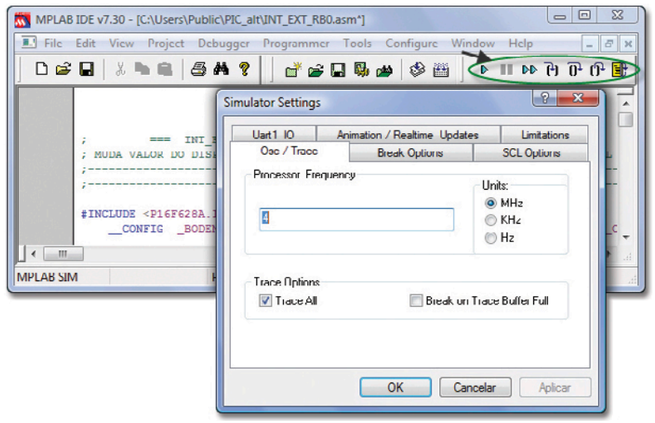
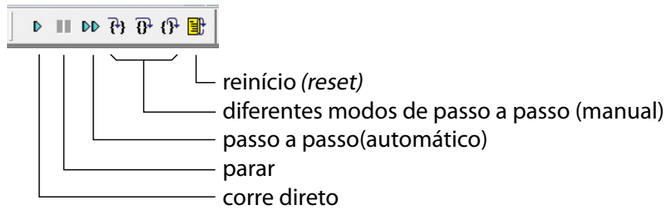


Figura C.10



Para monitorarmos os registradores durante o teste, usamos o comando **View > Watch** e, na tela **Watch**, selecionamos os registradores que queremos acompanhar divididos em quatro grupos: watch1, watch2, watch3 e watch4.

Para acompanharmos informações ligadas a tempo, utilizamos o comando **Debugger > Stopwatch**.

Para simularmos sinais em entradas, usamos o comando **Debugger > Stimulus** e podemos selecionar na tela as entradas e o tipo de sinal (*toggle*, *pulse* e *set*).

Se não forem detectados erros, podemos testar o programa fazendo-o correr sob controle, ou seja, nas condições oferecidas pelo *debugger*: *reset* (reinício), *run* (rodar o programa), passo a passo, *break point* (ponto de parada) etc.

Buscou-se, aqui, comentar partes importantes do MPLAB, sem, contudo, explorar todo seu potencial.

C.4 IC-PROG

Usaremos o IC-PROG para gravar o PIC16F628 depois de o programa ser compilado no MPLAB.

C.4.1 Configuração do IC-PROG

Para que o programa opere na língua portuguesa, utilizamos o comando **Settings > Options**. Clicando em **Settings > Device**, selecionamos o microcontrolador. Com a tecla F3, chamamos a janela de configuração de *hardware* do programa. Nessa janela, selecionamos (figura C.11):

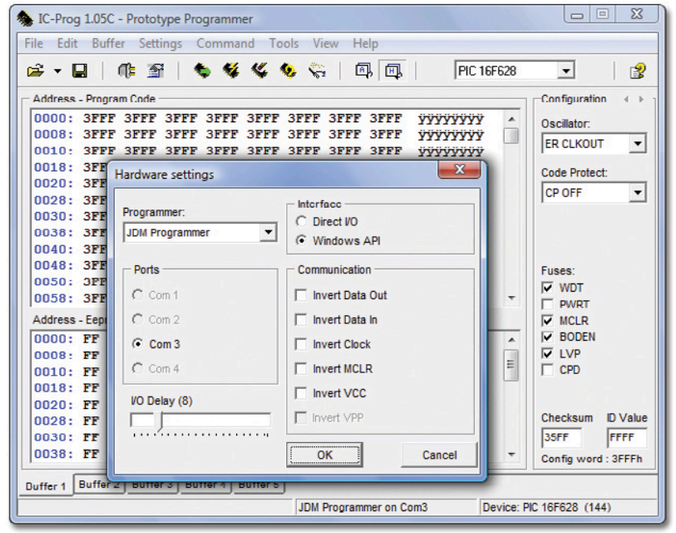
- JDM Programmer como programador desejado.
- A porta serial (COM1, COM2, COM3 ou COM4).
- O tipo de interface utilizado

  Direct I/O para Windows 95, Windows 98 e Windows ME.

  Windows API para Windows NT, Windows 2000 e Windows XP.

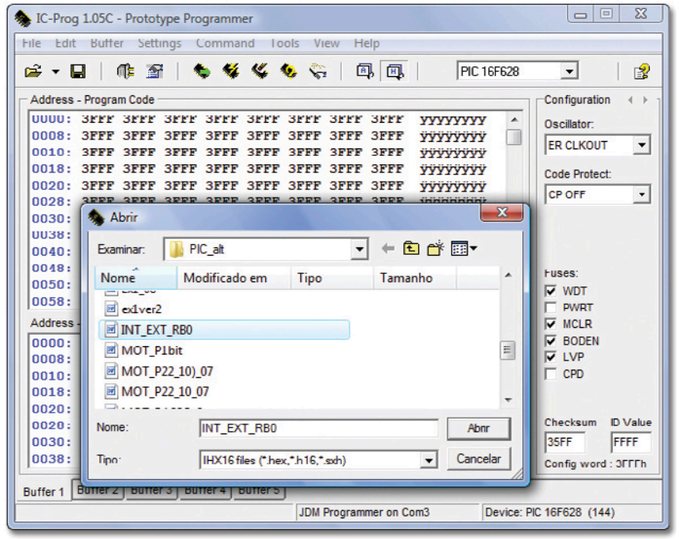
- Fator de retardo I/O. Processa com maior ou menor velocidade.

Figura C.11



Realizada a configuração, podemos selecionar o programa para ser gravado. Para isso, usamos **Open File** e selecionamos o programa de interesse, que deve ter extensão .hex. Essa seleção pode ser facilitada filtrando, na aba da janela que se abriu, somente programa com extensão .hex. Com o programa selecionado, escolhemos na aba **Oscillator** a condição correta, que depende do tipo de oscilador que gerou o *clock*. Agora, podemos iniciar a gravação clicando no ícone **Program All** da barra de menus (figura C.12).

Figura C.12



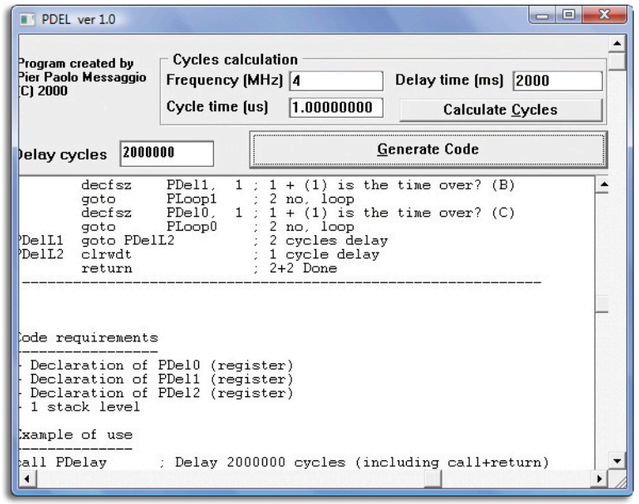
Então, clicamos em **Abrir** na janela menor e teremos a janela com os endereços de memória preenchidos conforme o programa (figura C.12).

Selecionamos **intRC CLKOUT** em **Oscilator** e clicamos em **Program All** (“raio”). Confirmamos a gravação. Estamos com o microcontrolador gravado e pronto para ser colocado no circuito em que deve trabalhar.

C.5 PICDEL

É um programa que gera *delay*. Em **Delay time (ms)**, colocamos o valor do *delay* que queremos em ms e clicamos em **Calculate Cycles** e, depois, em **Generate Code**. O programa vai gerar o *delay* solicitado. Colamos uma cópia do programa gerado em nosso programa e ganhamos o tempo de programar o *delay* (figura C.13).

Figura C.13



Referências bibliográficas

CAPUANO, F. G.; IDOETA, I. V. *Elementos de eletrônica digital*. São Paulo: Érica, 1998.

MALVINO, A. P. ; LEACH, D. P. *Eletrônica digital, princípios e aplicações*.vols. 1 e 2. São Paulo: McGraw-Hill,1987.

MELO, M. *Eletrônica digital*. São Paulo: McGraw-Hill, 1993.

PEREIRA, Fabio. *Microcontroladores PIC*. São Paulo: Érica, 2002.

SOUZA, D. J. *Desbravando o PIC*. São Paulo: Érica, 2000.

TOCCI, Ronald J. *Sistemas digitais: princípios e aplicações.* São Paulo: Pearson Prentice Hall, 2003.

TOKHEIM, R. L. *Introdução aos microprocessadores.* São Paulo: McGraw-Hill, 1985.

As figuras dos programas MPLAB, ICPROG e PICDEL foram obtidas nos próprios programas.


CENTRO PAULA SOUZA DO GOVERNO DE SÃO PAULO
CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

Excelência no ensino profissional

Administrador da maior rede estadual de educação profissional do país, o Centro Paula Souza tem papel de destaque entre as estratégias do Governo de São Paulo para promover o desenvolvimento econômico e a inclusão social no Estado, na medida em que capta as demandas das diferentes regiões paulistas. Suas Escolas Técnicas (Etecs) e Faculdades de Tecnologia (Fatecs) formam profissionais capacitados para atuar na gestão ou na linha de frente de operações nos diversos segmentos da economia.

Um indicador dessa competência é o índice de inserção dos profissionais no mercado de trabalho. Oito entre dez alunos formados pelas Etecs e Fatecs estão empregados um ano após concluírem o curso. Além da excelência, a instituição mantém o compromisso permanente de democratizar a educação gratuita e de qualidade. O Sistema de Pontuação Acrescida beneficia candidatos afrodescendentes e oriundos da Rede Pública. Mais de 70% dos aprovados nos processos seletivos das Etecs e Fatecs vêm do ensino público.

O Centro Paula Souza atua também na qualificação e requalificação de trabalhadores, por meio do Programa de Formação Inicial e Educação Continuada. E ainda oferece o Programa de Mestrado em Tecnologia, recomendado pela Capes e reconhecido pelo MEC, que tem como área de concentração a inovação tecnológica e o desenvolvimento sustentável.

FUNDAÇÃO
PADRE ANCHIETA