Revisão de Circuitos Digitais

Adaptações Prof. José Artur Quilici-Gonzalez

Elementos de Eletrônica Digital – Idoeta e Capuano

Embedded System Design – Vahid e Givargis

Logic and Computer Design Fundamentals – Mano e Kime

FUNÇÃO AND ou E

- Executa a multiplicação de 2 ou mais variáveis
- **F = x . y** (leia-se F = x **e** y)

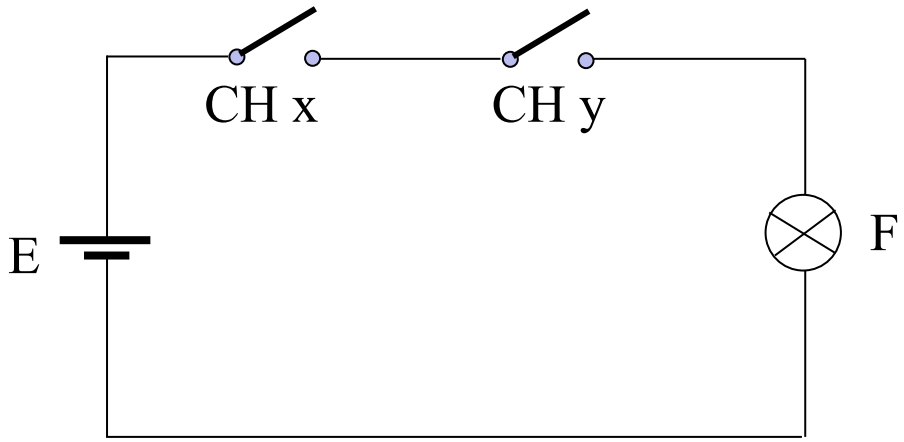
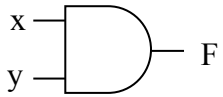


Tabela da Verdade

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1



F = x y
AND

Convenções:

Chave aberta = 0

Chave fechada = 1

FUNÇÃO OR ou OU

- Assume valor 1 quando uma ou mais variáveis for(em) 1
- **F = x + y** (leia-se F = x **ou** y)

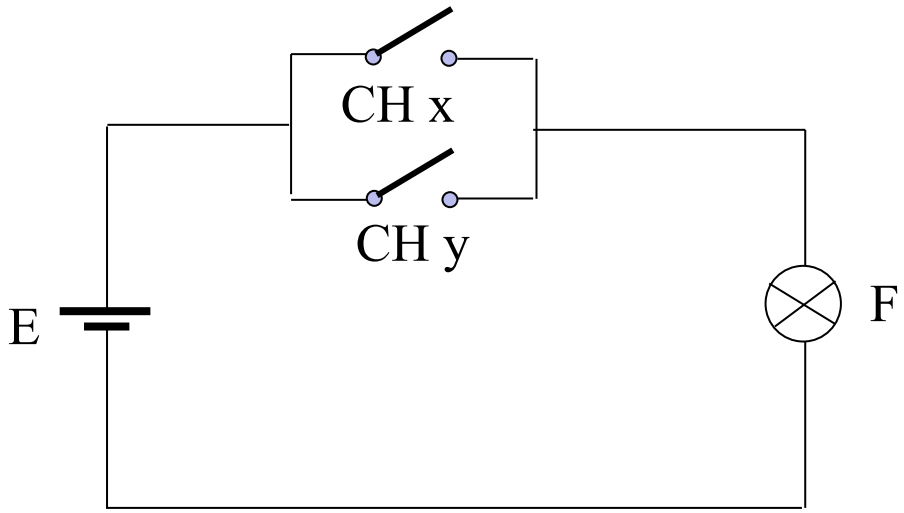
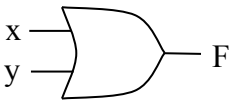


Tabela da Verdade

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1



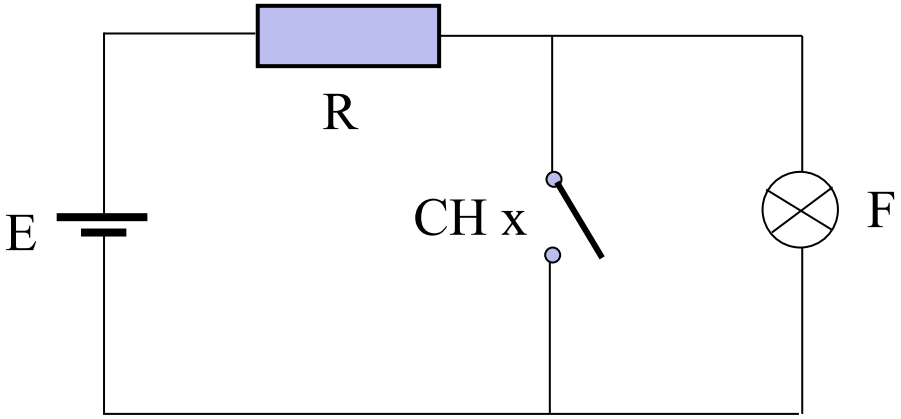
F = x + y
OR

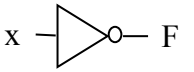
FUNÇÃO NOT ou NÃO

- Inverte ou complementa o estado da variável
- **F = x´** (leia-se F = **não** x)

Tabela da Verdade

x	F
0	1
1	0



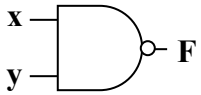


F = x'
Inversor

O inversor é o bloco lógico que executa a função NOT

FUNÇÃO NAND ou NÃO E ou NE

- É uma composição da função AND com a função NOT
- **F = (x . y)´**



F = (x y)'
NAND

Tabela da Verdade

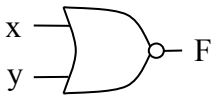
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

FUNÇÃO NOR ou NÃO OU ou NOU

- É uma composição da função OR com a função NOT
- **F = (x + y)´**

Tabela da Verdade

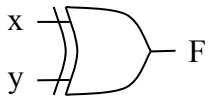
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0



F = (x+y)'
NOR

FUNÇÃO XOR ou OU Exclusivo

- Assume valor 1 quando as variáveis de entrada forem diferentes
- **F = x´.y + x.y´**



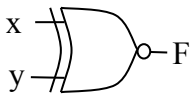
F = x ⊕ y
XOR

Tabela da Verdade

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

FUNÇÃO XNOR ou NOU Exclusivo

- Assume valor 1 quando as variáveis de entrada forem iguais
- **F = x´.y´ + x.y**



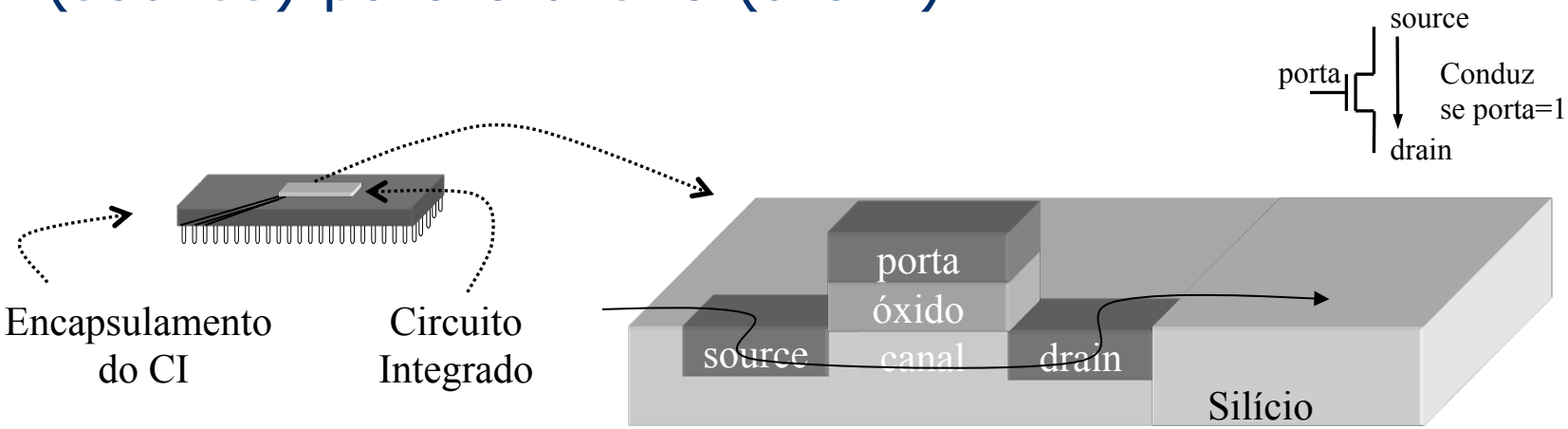
F = x ⊙ y
XNOR

Tabela da Verdade

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

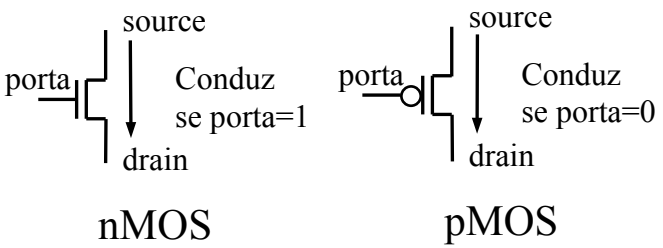
IMPLEMENTAÇÃO DA PORTA LÓGICA

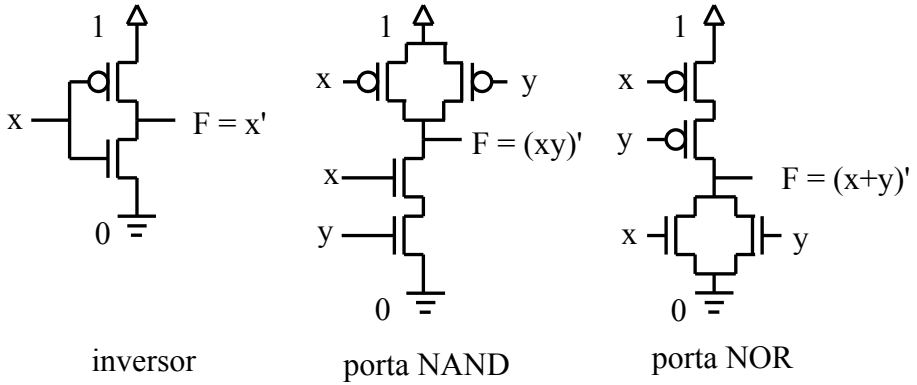
- Transistor CMOS
  - Componente elétrico básico nos Sistemas Digitais
  - Funciona como um interruptor aberto/fechado
  - Voltagem na "porta" controla a corrente da fonte (*source*) para o dreno (*drain*)



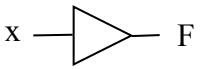
IMPLEMENTAÇÕES COM O CMOS

- **C**omplementary **M**etal **O**xide **S**emiconductor
- Níveis Lógicos
  - Tipicamente 0 é 0V, 1 é 5V
- Dois tipos básicos de CMOS
  - nMOS conduz se porta=1
  - pMOS conduz se porta=0
  - Por isso, “complementary”
- Portas básicas
  - Inversor, NAND, NOR



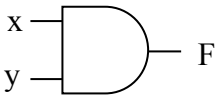


PORTAS LÓGICAS BÁSICAS



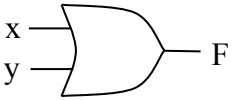
x	F
0	0
1	1

F = x
Driver



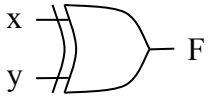
x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

F = x y
AND



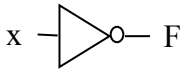
x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

F = x + y
OR



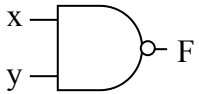
x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

F = x ⊕ y
XOR



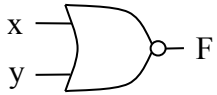
x	F
0	1
1	0

F = x'
Inversor



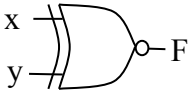
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

F = (x y)'
NAND



x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

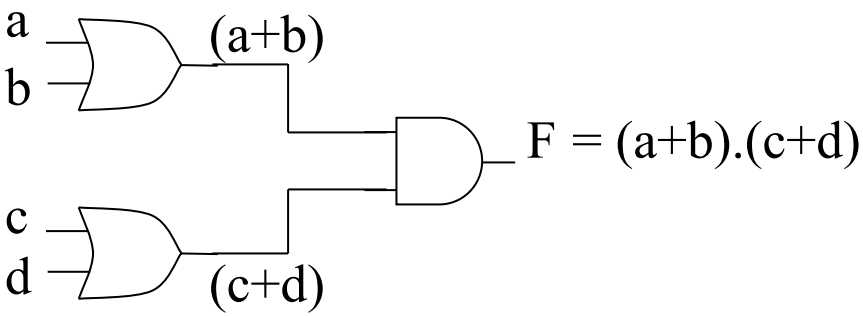
F = (x+y)'
NOR

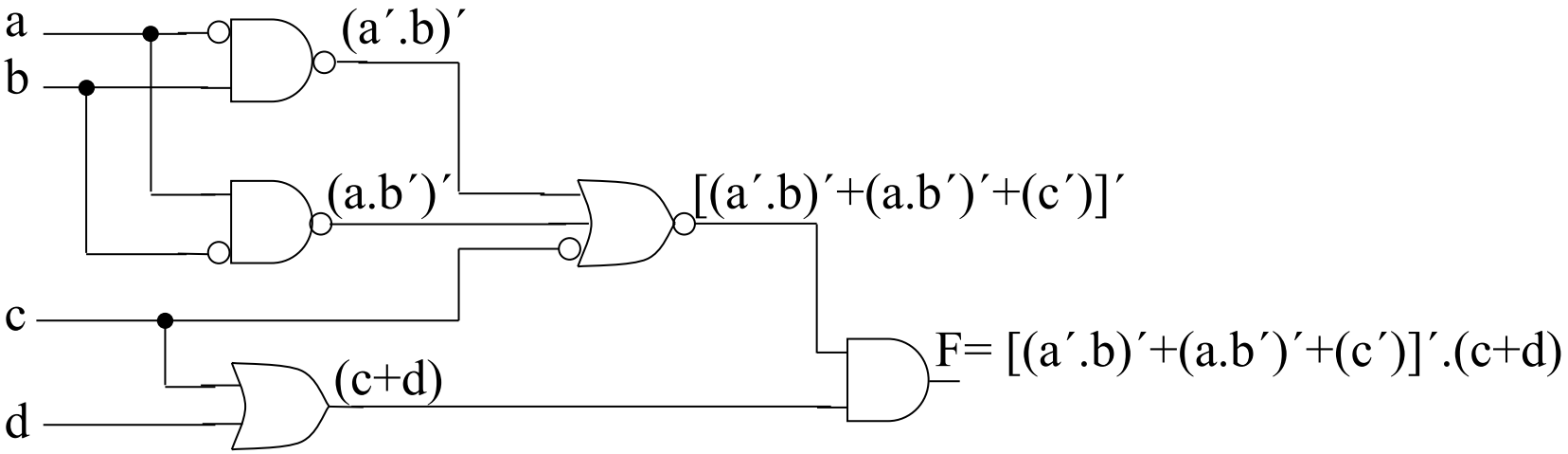


x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

F = x ⊙ y
XNOR

Expressões Booleanas Obtidas de Circuitos





Circuitos Obtidos de Expressões Booleanas

$F = \underbrace{abc}_{(1)} + \underbrace{\underbrace{(a+b)}_{(2)}.c}_{(3)}$

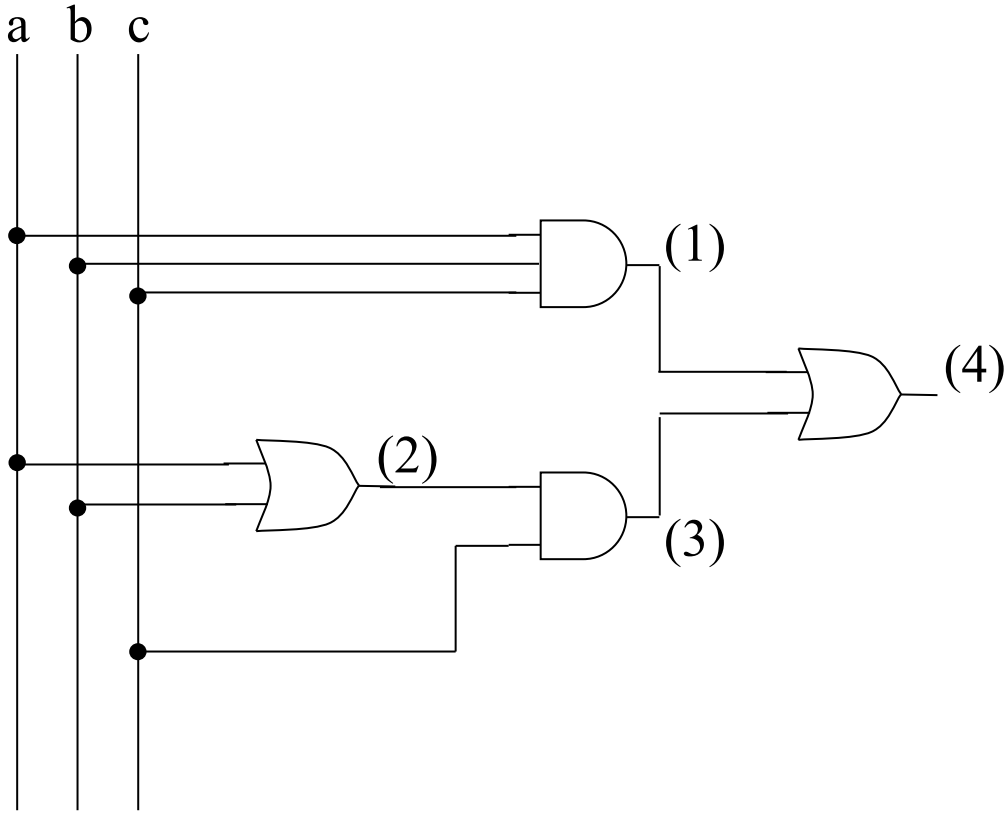
$\underbrace{\qquad\qquad}_{(4)}$

(1) porta AND com *a*, *b* e *c*

(2) porta OR com *a* e *b*

(3) porta AND com 2 e *c*

(4) porta OR com (1) e (3)



Tabelas da Verdade Obtidas de Expressões Booleanas

F=a.b´.c + a´.b

Entradas					Saída
a	b	c	ab´c	a´b	F
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	0	0	0

MAPAS DE KARNAUGH - COBERTURA MÍNIMA

- Mapa de Karnaugh
  - 1 representa **mintermo**
  - Círculo representa **implicante**
- **Cobertura Mínima**
  - Cobertura de todos os 1's com o no. mínimo de círculos

Soma de Produtos

F = abc'd' + a'b'cd + a'bcd + ab'cd

Soma de Produtos

ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	0	0	0
10	0	0	1	0

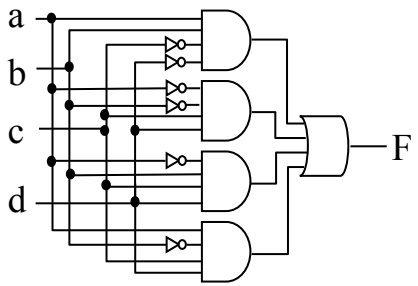
Cobertura Mínima

ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	0	0	0
10	0	0	1	0

Soma de Produtos

F = abc'd' + a'b'cd + a'bcd + ab'cd

Implementação Direta



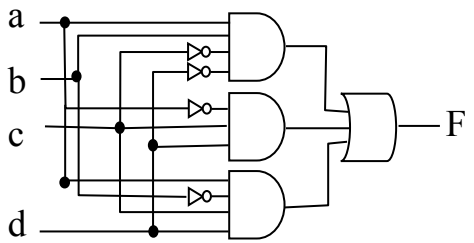
4 porta AND de 4 entradas
1 porta OR de 4 entradas

→ 40 transistores (cada entrada => 2 transistores)

Cobertura Mínima

F=abc'd' + a'cd + ab'cd

Implementação da Cobertura Mínima



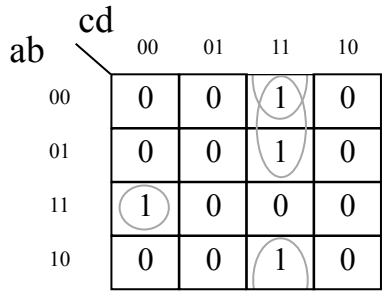
2 porta AND de 4 entradas
1 porta AND de 3 entradas
1 porta OR de 3 entradas

→ 28 transistores

MAPAS DE KARNAUGH - COBERTURA MÍNIMA PRIMA

- Busca o número mínimo de entradas nas portas AND
- **Implicante Primo**
  - Implicante não coberto por nenhum outro implicante
  - Círculos de tamanho máximo no mapa
- **Cobertura Mínima Prima**
  - Cobertura com um número mínimo de implicantes
  - Número mínimo de círculos máximos

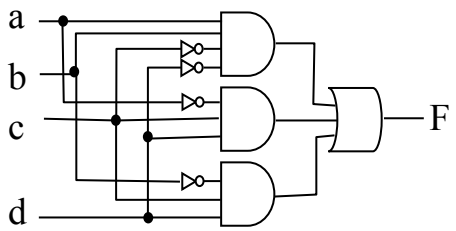
Cobertura Mínima Prima



Cobertura Mínima Prima

F=abc'd' + a'cd + b'cd

Implementação



1 porta AND de 4 entr.
2 porta AND de 3 entr.
1 porta OR de 3 entr.

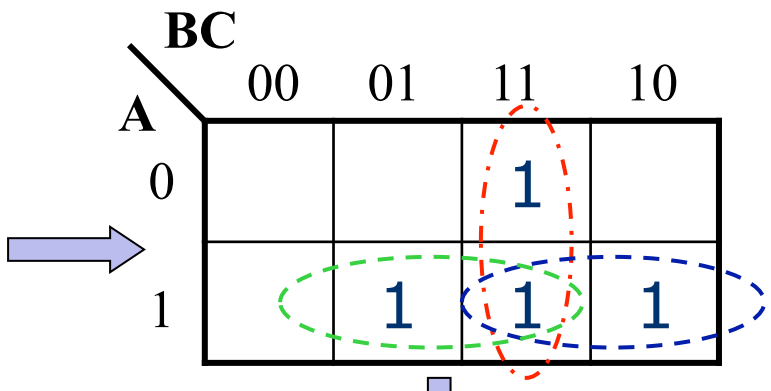
→ 26 transistores

Exemplo de Circuito Combinacional

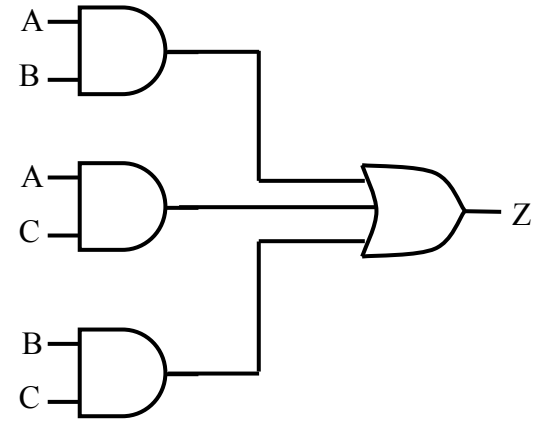
Especificação: construir um "**Circuito Votador**" de três entradas com uma saída que acompanha a maioria das entradas (i.e., se a maioria for zero, a saída será zero)

A	B	C	Z
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

$Z = A'BC + AB'C + ABC' + ABC$



$Z = AB + AC + BC$



Seleção de Dados

⇒Seleção de dados ou de informação é uma função crítica em Sistemas Digitais e Computadores

⇒Os circuitos que fazem a seleção possuem:

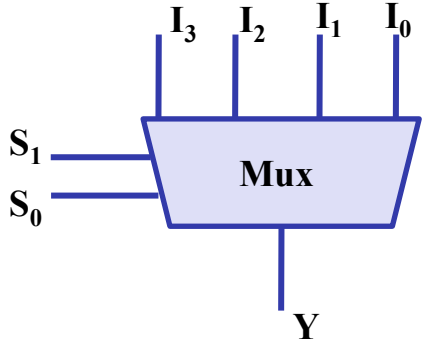
⇒um conjunto de entradas de informação, das quais uma deve ser selecionada

⇒uma única saída

⇒um conjunto de linhas de controle para fazer a seleção

⇒Estes Circuitos Lógicos são chamados de

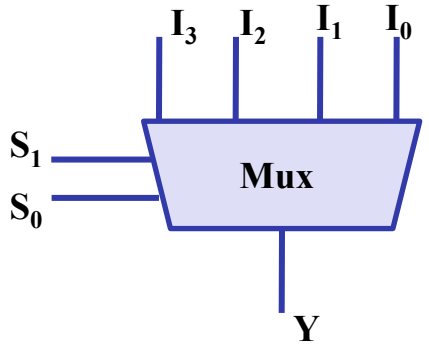
Multiplexadores



Multiplexadores

⇒Um Multiplexador seleciona informação de uma das **linhas de entrada** e a direciona para uma **linha de saída**

⇒Um Multiplexador típico tem *n* **entradas de controle** (S_{n-1} ,... S_0), 2^n **entradas de informação** (I_{2^n-1}, ... I_0) e uma **saída** Y



Multiplexador 2 x 1 Linha

⇒Como $2 = 2^1$, $n = 1$

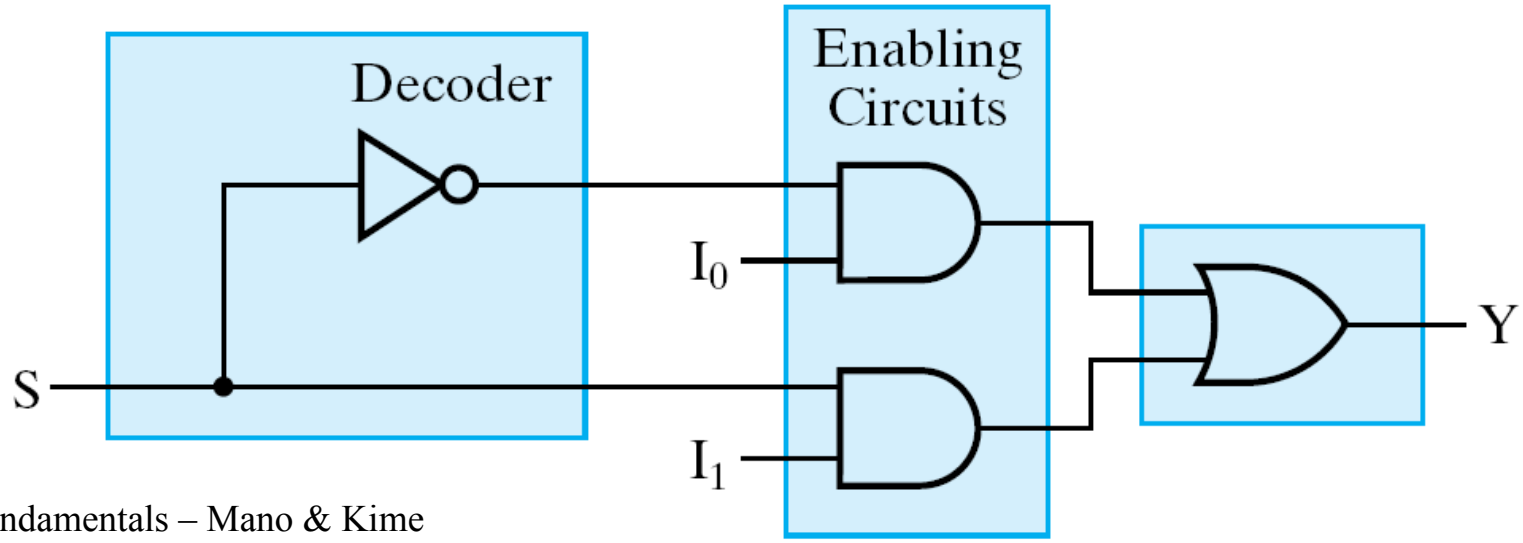
⇒A variável de seleção S tem dois valores:

⇒S = 0 seleciona a entrada I_0

⇒S = 1 seleciona a entrada I_1

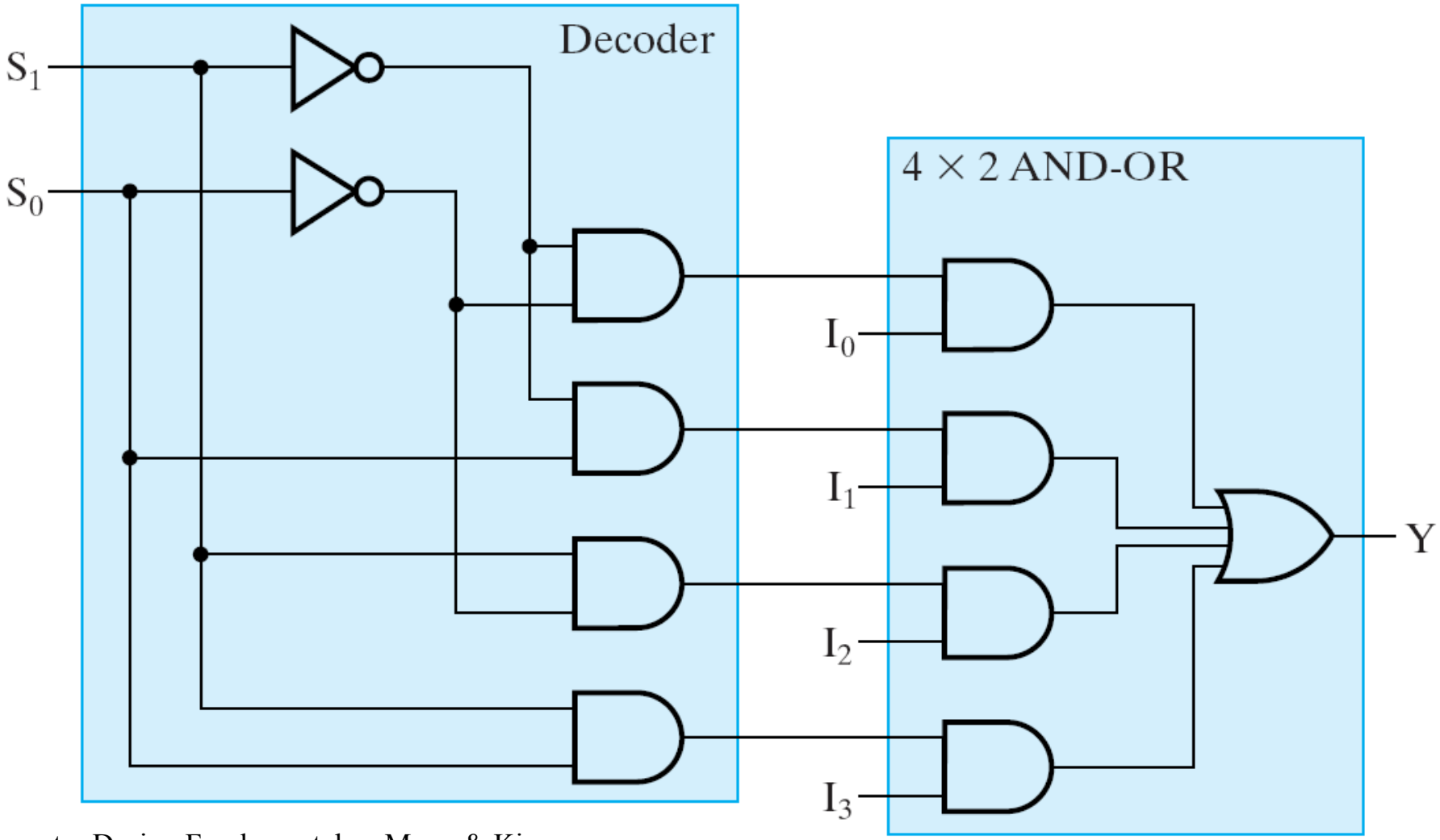
⇒Equação: $Y = S'I_0 + SI_1$

Um Multiplexador é essencialmente um Decodificador e uma porta OR



Multiplexador 4 x 1 Linha

⇒Como $4 = 2^2$, $n = 2$

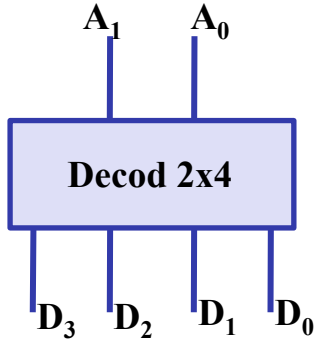


Logic and Computer Design Fundamentals – Mano & Kime

Decodificadores

•Um decodificador converte um **Código de Entrada** de *n*-bits em um **Código de Saída** de *m*-bits, com $n \leq m \leq 2^n$, de modo que cada palavra de código válida produza um único Código de Saída

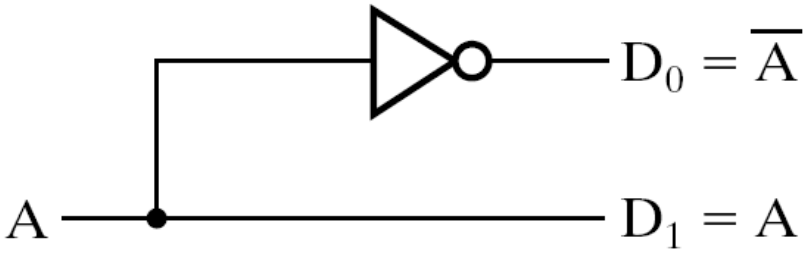
•Blocos funcionais de decodificação são chamados **Decodificadores *n* x *m*** linhas, com $m \leq 2^n$, e geram 2^n (ou menos) **mintermos** com as *n* variáveis de entrada



Decodificadores

Decodificador 1x2 linhas

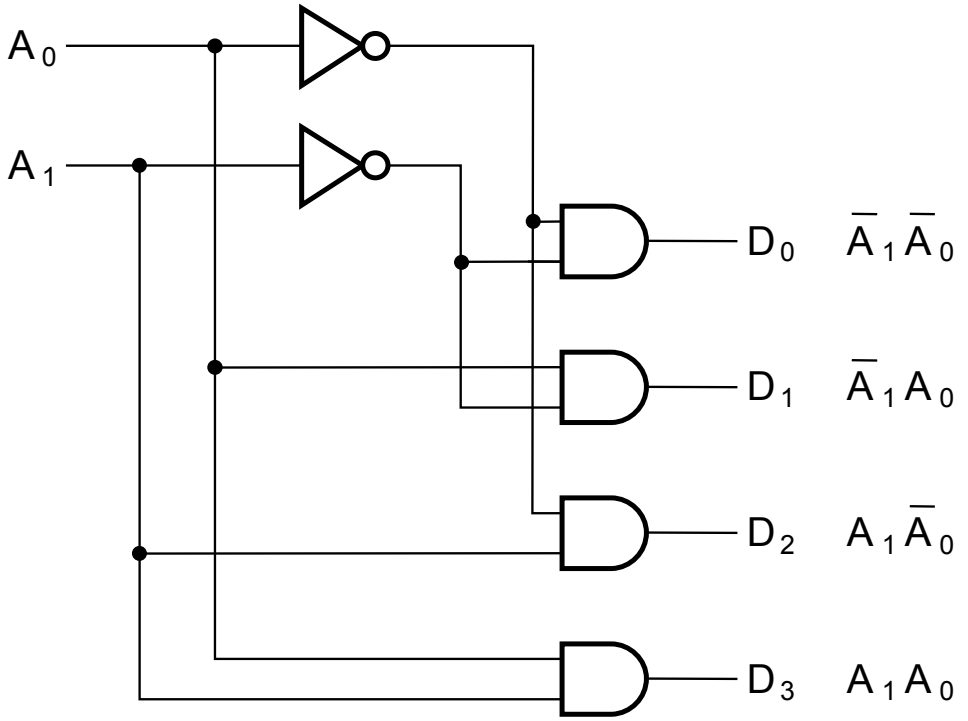
A	D_0	D_1
0	1	0
1	0	1



Decodificador 2x4 linhas

A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

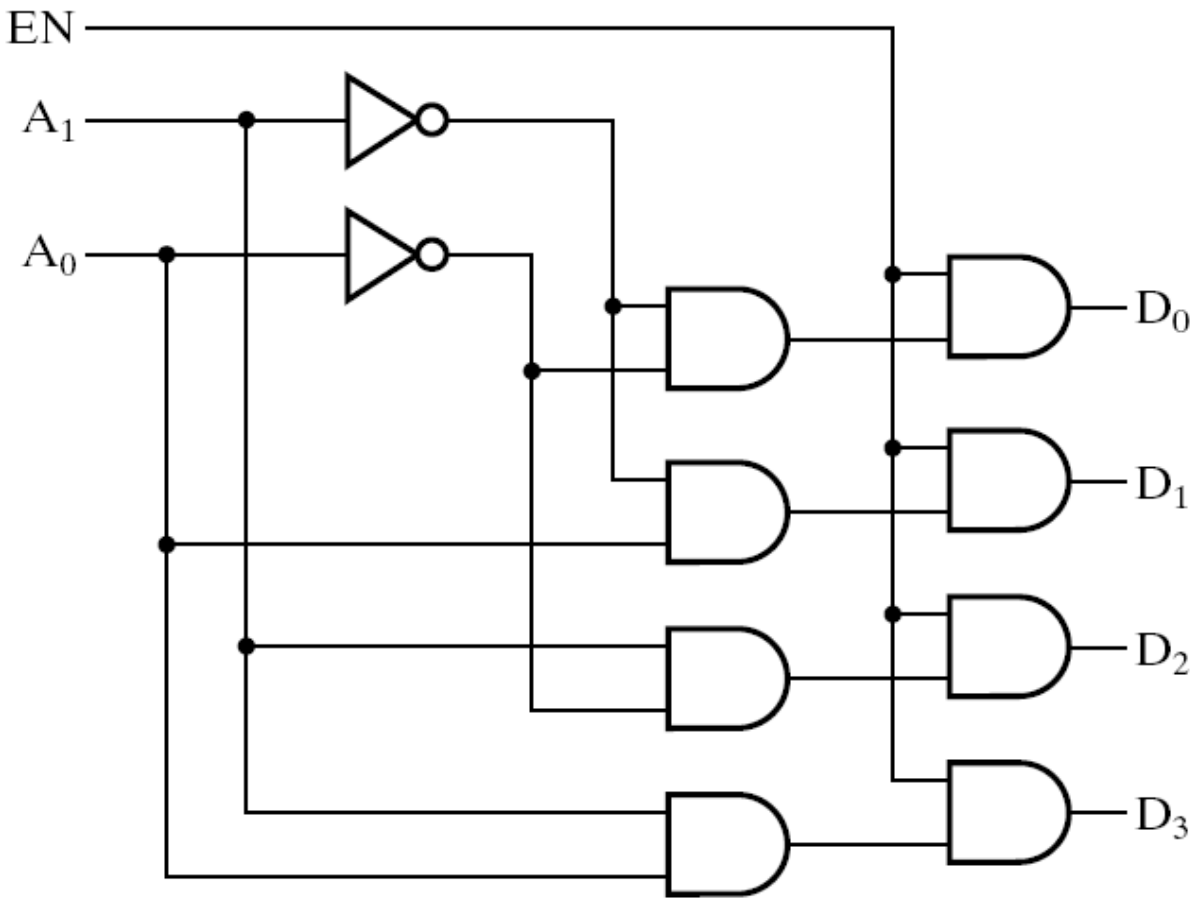
(a)



(b)

Decodificador com *Enable*

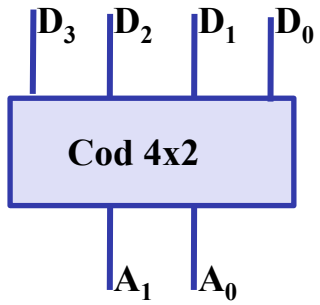
EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



Codificadores

•Um Codificador converte um **Código de Entrada** de *m*-bits em um **Código de Saída** de *n*-bits, com $n \leq m \leq 2^n$, de modo que cada palavra de código válida produza um único Código de Saída

•**Codificadores** normalmente convertem um código contendo exatamente um *bit* de valor 1 para um código binário correspondendo à posição em que o 1 aparece



Codificador de Prioridade

Em muitas situações, se um codificador tiver duas ou mais entradas acionadas, pode ser interessante que a entrada com a maior prioridade seja detectada e atendida primeiro

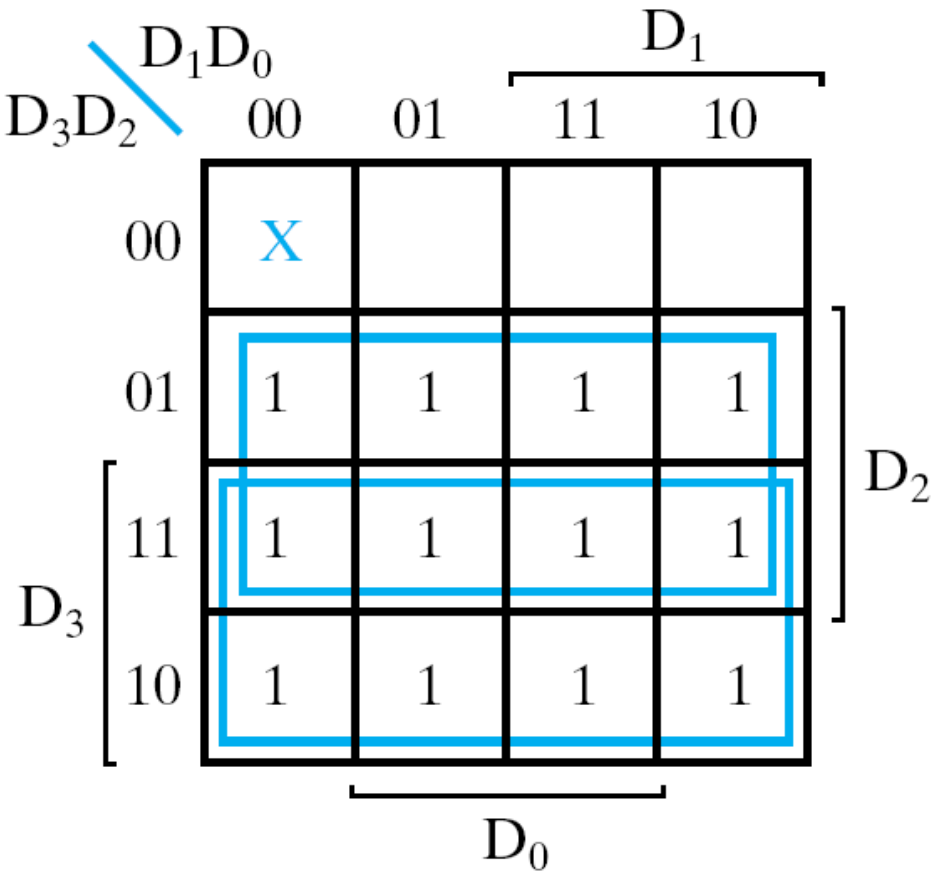
Tabela da Verdade do Codificador de Prioridade

maior prioridade

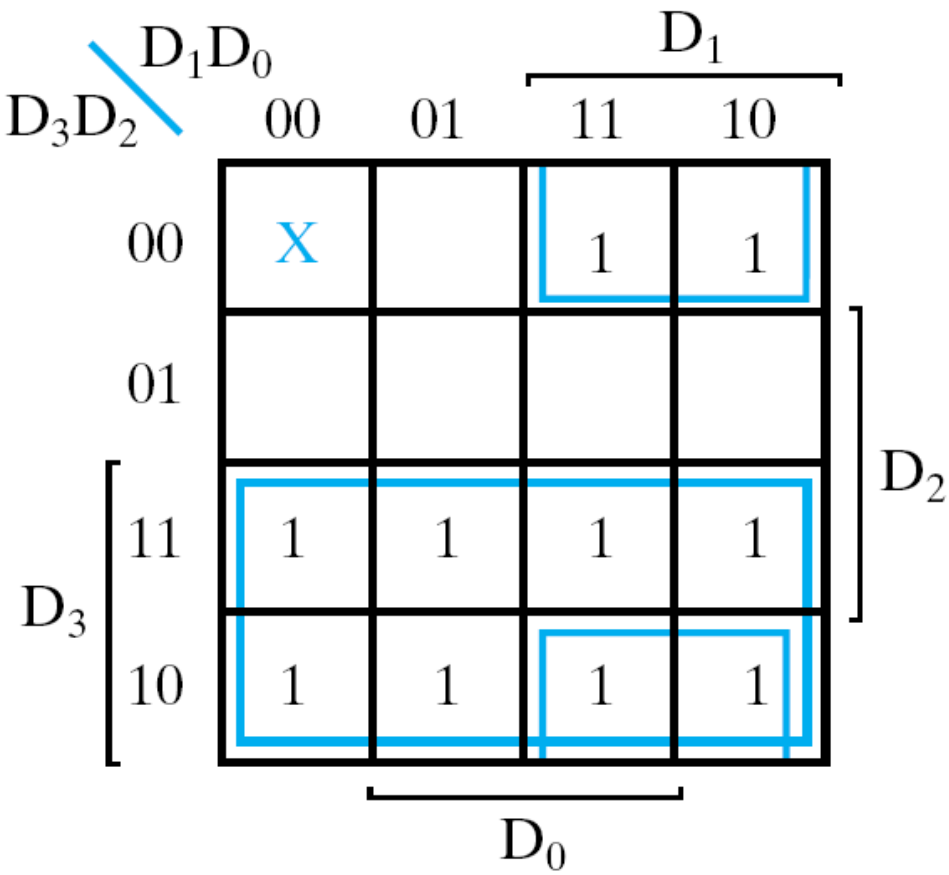
saída válida ou não

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Mapa de Karnaugh do Codificador de Prioridade



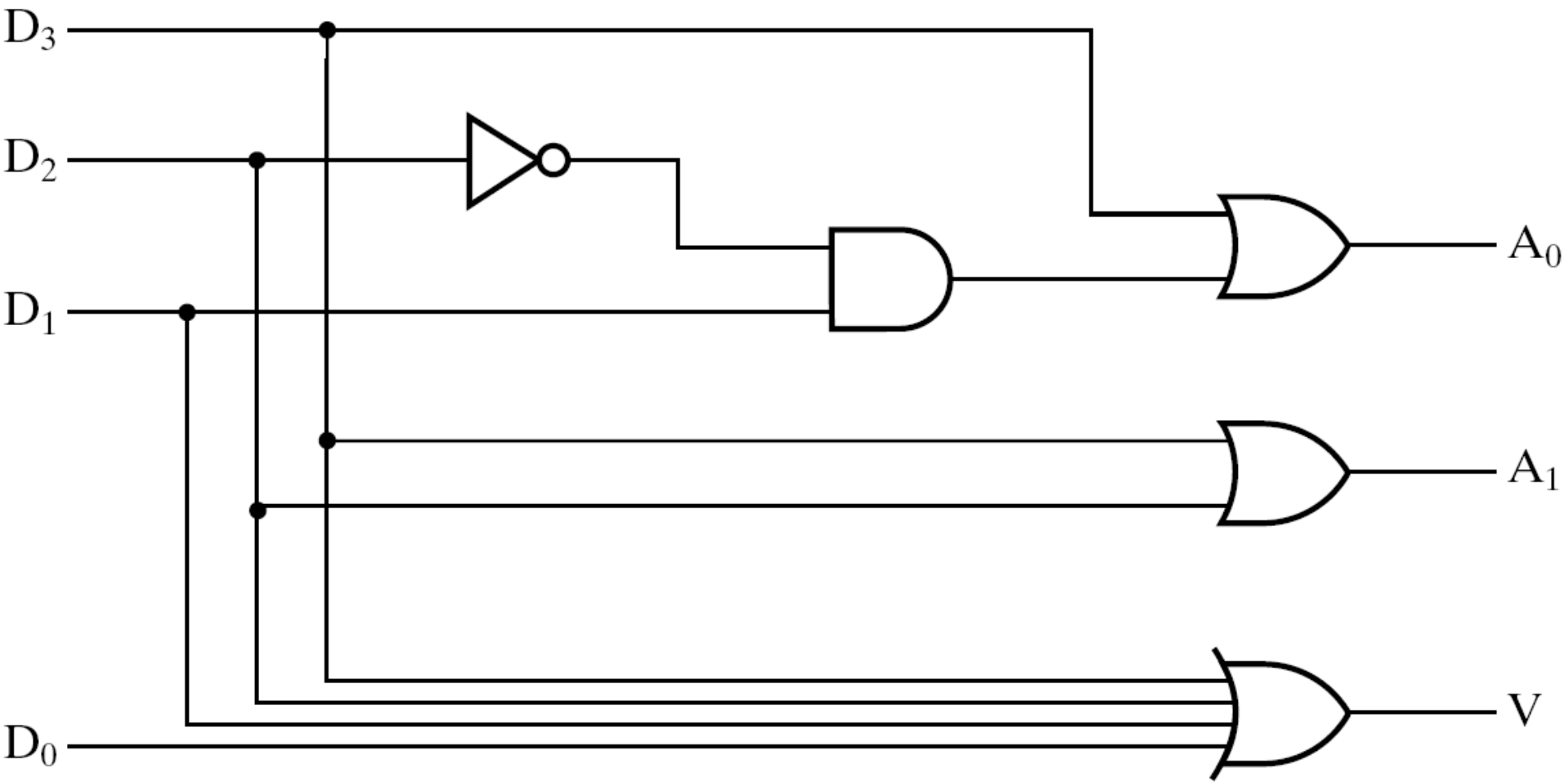
$A_1 = D_2 + D_3$



$A_0 = D_3 + D_1\overline{D}_2$

Logic and Computer Design Fundamentals – Mano & Kime

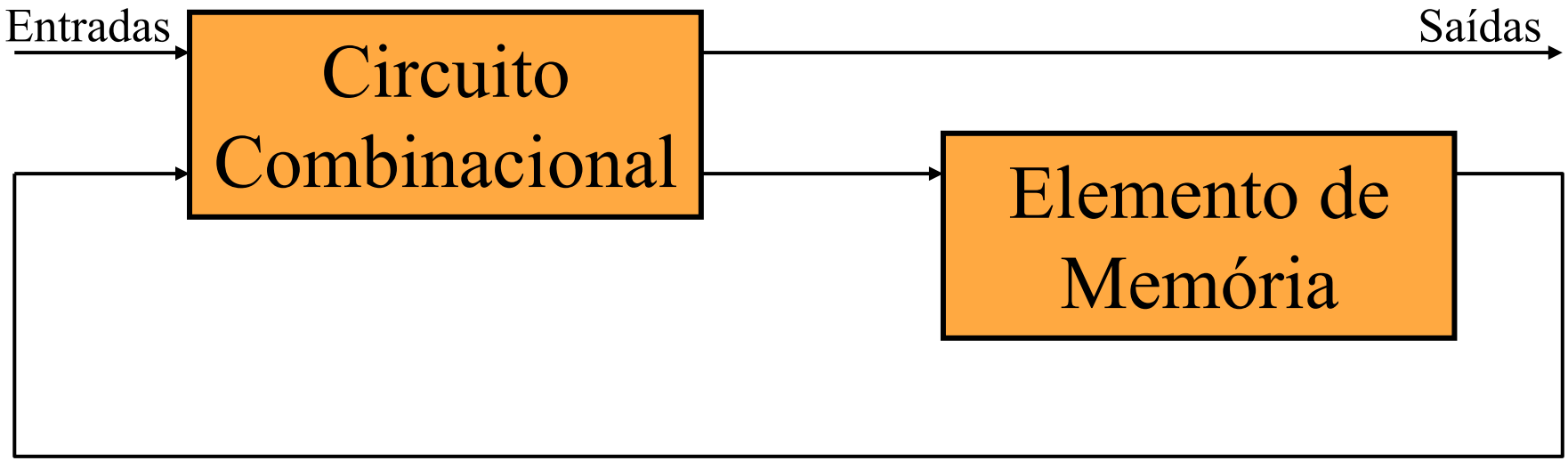
Implementação do Codificador de Prioridade



Logic and Computer Design Fundamentals – Mano & Kime

Circuitos Lógicos Sequenciais

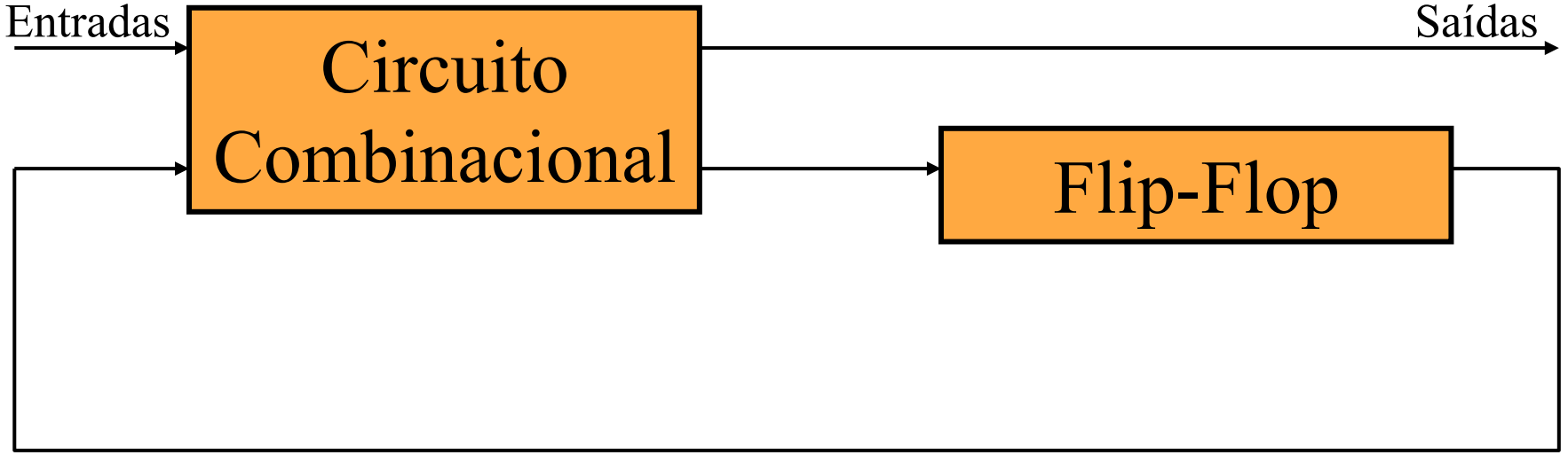
Circuito Lógico Sequencial é aquele que possui algum Elemento de Memória



A maioria dos **Sistemas Digitais** é constituída de **Circuitos Combinacionais** e de elementos de **Memória**

Elemento de Memória

O Elemento de Memória mais importante é o *Flip-Flop* (*FF*)

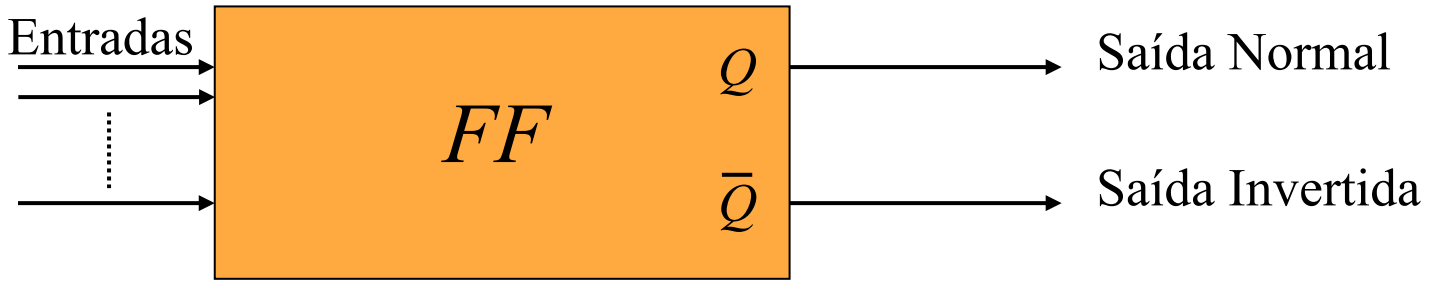


Normalmente um *FF* é implementado a partir de Portas Lógicas

Latch é o tipo mais simples de *FF* enquanto que o termo *Multivibrador Biestável* é a denominação mais técnica para um *FF*

Flip-Flop

Um *FF* pode ter uma ou mais Entradas e apresenta duas Saídas, denominadas Q e Q', opostas entre si



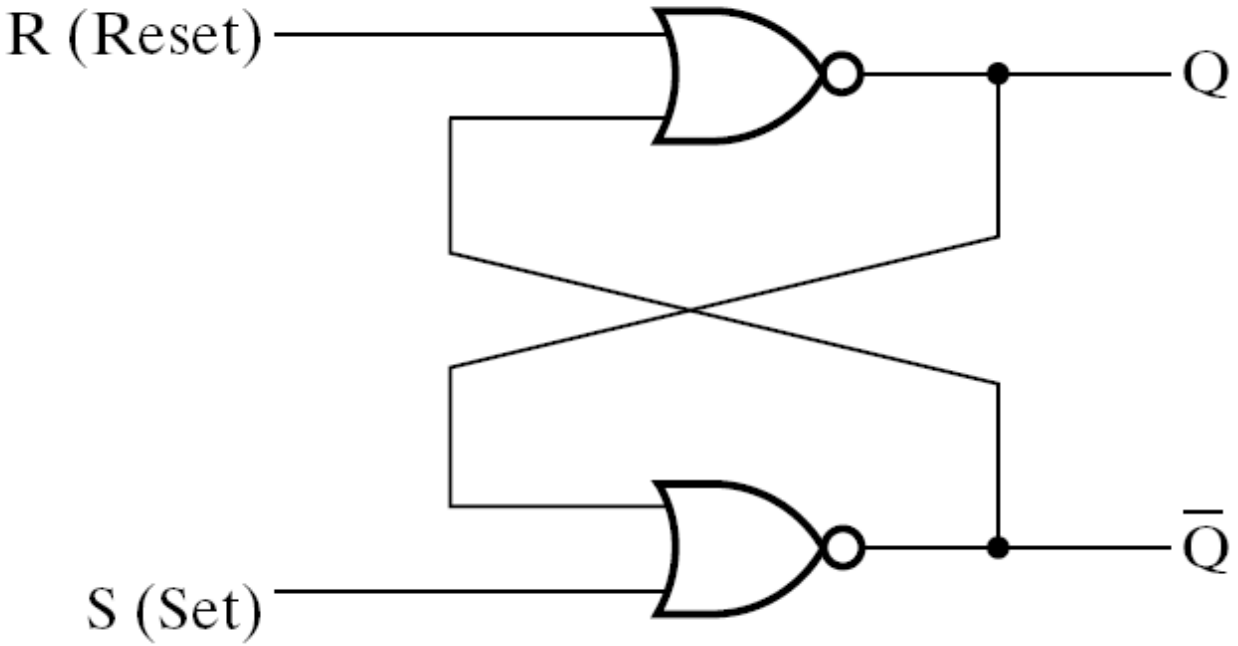
A saída Q é denominada **Saída Normal** do *FF* e Q' é a **Saída Invertida** do *FF*

Quando se diz que um *FF* está no estado **ALTO** ou estado ***SET***, isto significa que $Q = 1$ e $Q' = 0$

Quando se diz que um *FF* está no estado **BAIXO** ou estado ***RESET***, isto significa que $Q = 0$ e $Q' = 1$

Latch SR com Portas NOR

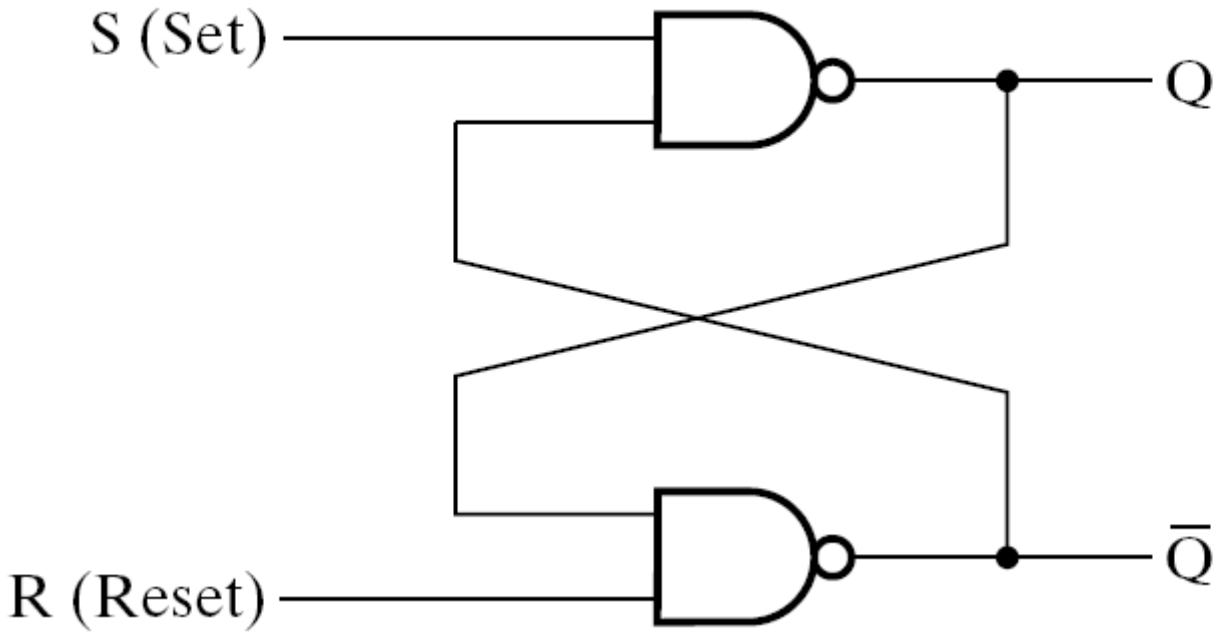
O elemento de memória mais básico é o *Latch*, normalmente usado dentro de *FFs*



S	R	Q	$\overline{Q}$	
1	0	1	0	Estado
0	0	1	0	*SET*
0	1	0	1	Estado
0	0	0	1	*RESET*
1	1	0	0	Estado Inválido

Aplicando-se 1 nas duas entradas, as saídas vão para 0, violando o princípio de que uma deve ser o complemento da outra

Latch S´R´ com Portas *NAND*

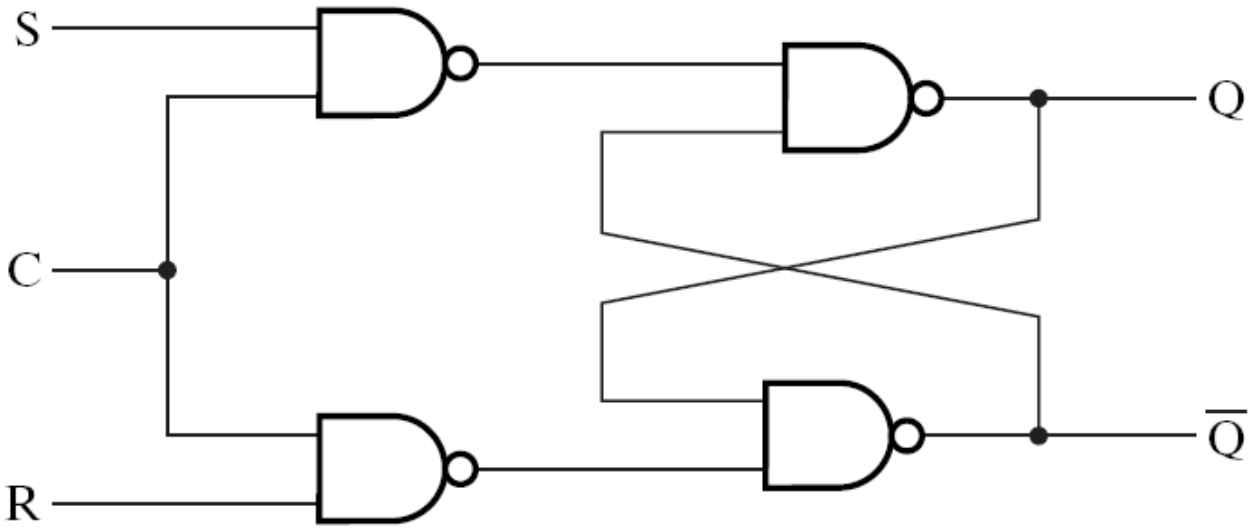


S	R	Q	$\overline{Q}$	
0	1	1	0	Estado *SET*
1	1	1	0	
1	0	0	1	Estado *RESET*
1	1	0	1	
0	0	1	1	Estado Inválido

Aplicando-se 0 nas duas entradas, as saídas vão para 1, violando o princípio de que uma deve ser o complemento da outra

Latch SR com Entrada de Controle

A entrada de controle *C* funciona como um sinal habilitador

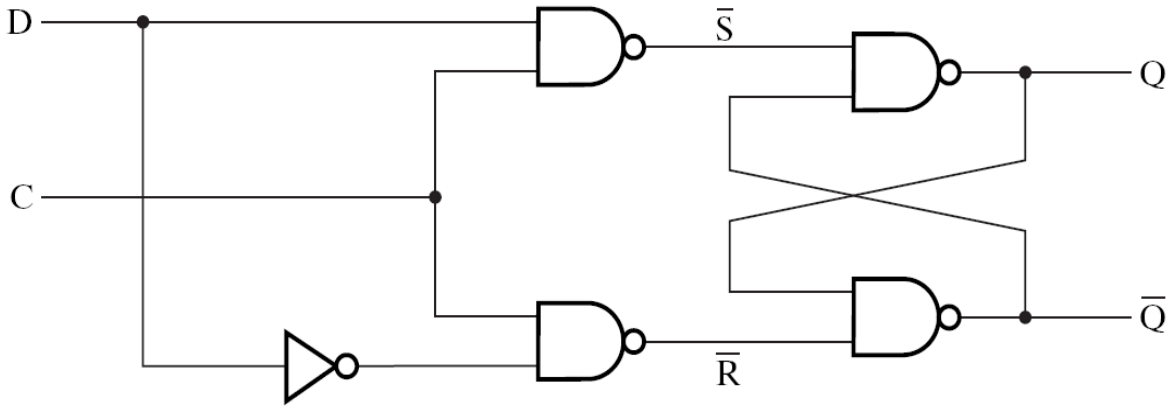


C	S	R	Próximo estado de *Q*
0	X	X	Não se altera
1	0	0	Não se altera
1	0	1	*Q=0*; Estado *Reset*
1	1	0	*Q=1*; Estado *Set*
1	1	1	Estado inválido

Somente quando a entrada de controle for *C =1*, a informação contida em *S* e *R* será passada para o *latch S'R'*

Latch D

Acrescentando-se um inversor no *Latch SR*, obtém-se o *Latch D*

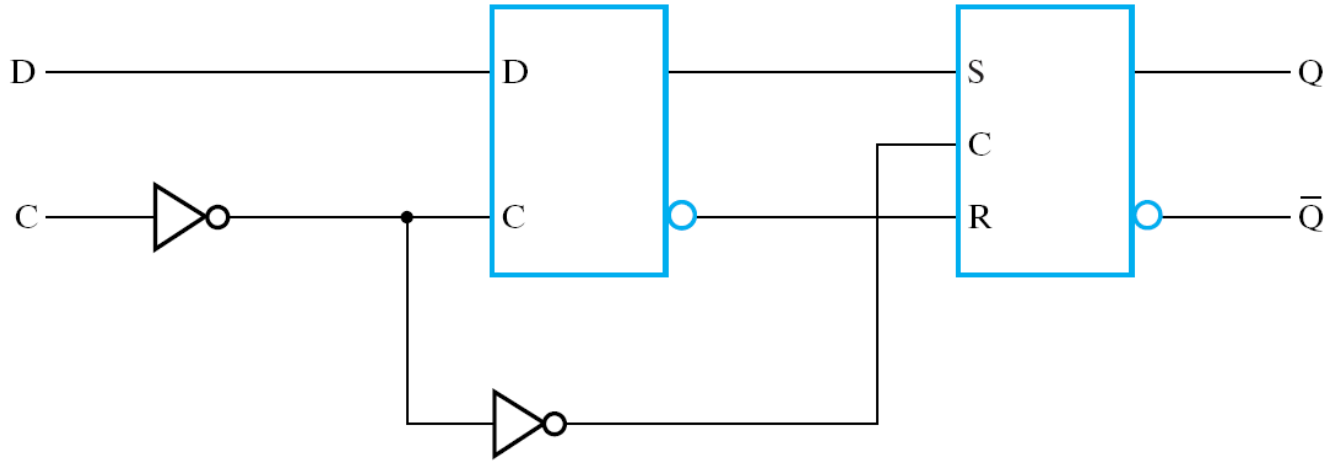


C	D	Próximo estado de Q
0	X	Não se altera
1	0	Q=0; Estado *Reset*
1	1	Q=1; Estado *Set*

Para o *Latch D* não há Estado Inválido !

Positive-Edge-Triggered Flip-Flop

O ***FF* Disparado por Borda de Subida** consiste de um *latch D*, outro *latch SR* e dois inversores



Este *FF* só muda sua saída durante a transição do *clock* de *0* para *1*

Quando *C=0*, o primeiro *latch* torna-se transparente, mas o segundo *latch* fica bloqueado

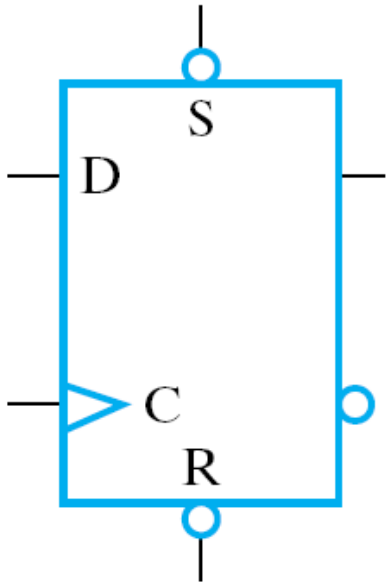
Quando *C* passa de *0* para *1*, o último valor de *D* é retido e passado para a entrada *S* do segundo *latch*, que agora pode copiar este valor para sua saída *Q*

Entradas Diretas ou Assíncronas

Imediatamente após ligar ou *resetar*, antes de iniciar a operação, um circuito sequencial pode ser colocado num estado conhecido

Normalmente isto é feito de forma **assíncrona**, i.e., sem levar em conta o *clock* do circuito

S	R	C	D	Q	$\overline{Q}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	Undefined	
1	1	↑	0	0	1
1	1	↑	1	1	0



Entradas diretas *S* e *R* que controlam o estado do *FF* são usadas para este fim

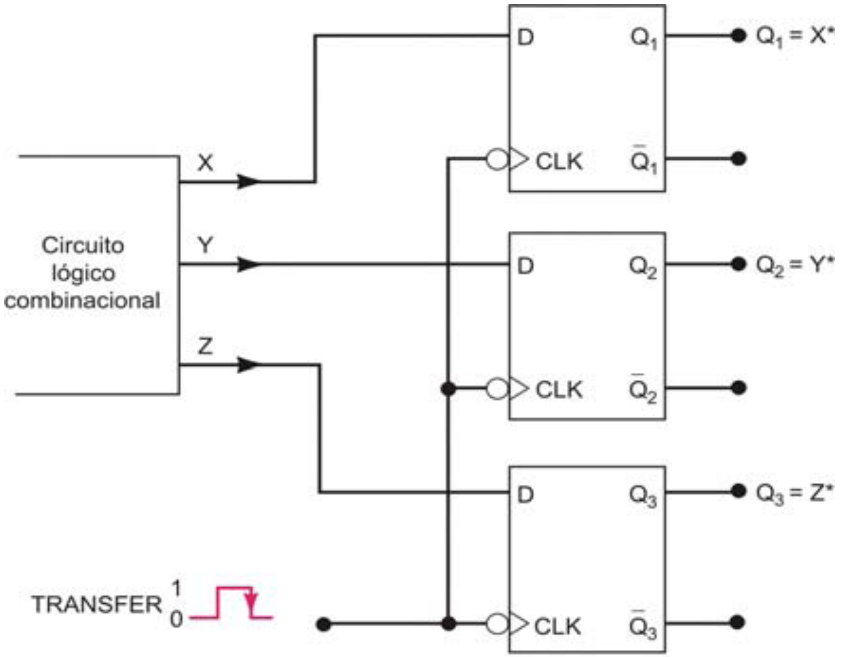
Por ex., um *0* colocado em *S* leva o *FF* para o estado *SET (Q=1)*, enquanto que um *0* colocado em *R* leva o *FF* para o estado *RESET (Q=0)*

Transferência de Dados em Paralelo

Muitas vezes é necessário armazenar os valores de saída de um **Circuito Lógico Combinacional** para processamento posterior

Na fig. ao lado, as saída *X, Y* e *Z* de um circuito lógico são transferidas para os *FFs* Q_1, Q_2 e Q_3 na borda de descida do pulso *TRANSFER* nas entradas *CLK,* que são comuns

Esse é um exemplo de **transferência paralela** de um dado binário; os três bits *X, Y* e *Z* são transferidos ***simultaneamente***



*Após a ocorrência da borda de descida.

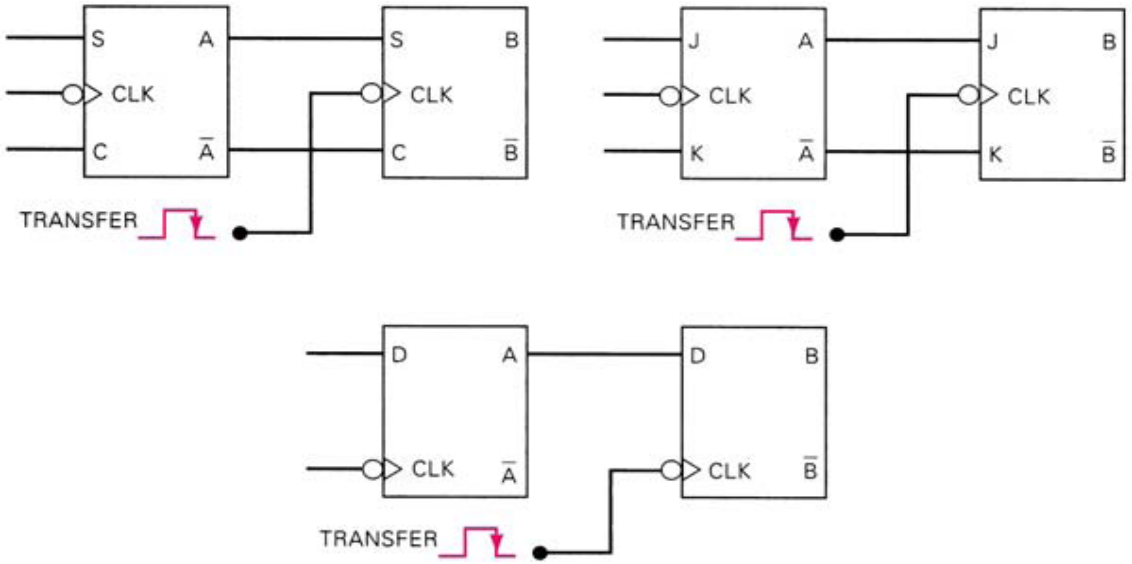
Transferência Síncrona de Dados

O uso mais comum de *FFs* é no armazenamento de dados ou informações

Esses dados são geralmente armazenados em grupos de *FFs* denominados **Registradores**

A fig. ao lado ilustra a transferência de dados entre dois *FFs A* e *B* na borda de descida do pulso *TRANSFER*

Neste caso, trata-se de uma **transferência síncrona**, visto que as entradas de controle síncronas (*S* e *C*, ou *J* e *K*, ou *D*) e a entrada *CLK* foram usadas para realizar a transferência

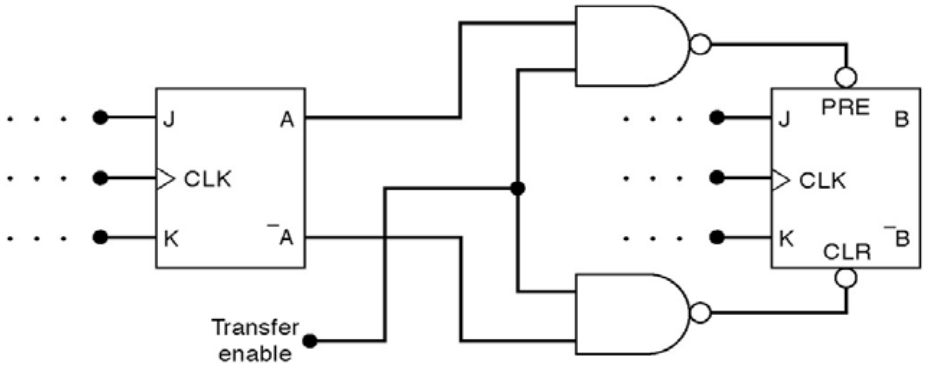


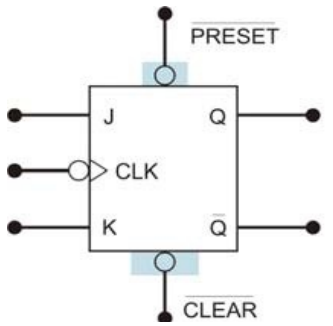
Transferência Assíncrona de Dados

Usando-se as entradas assíncronas (ou diretas) *PRE (PRESET)* e *CLR (CLEAR)*, ativas em nível baixo, ocorre uma **transferência assíncrona** (independentemente das entradas síncronas *J* e *K* e do *clock* do *FF*)

Quando a linha *TRANSFER ENABLE* é colocada em nível ALTO, uma das saídas das portas *NAND* vai para o nível BAIXO, dependendo do estado das saídas *A* e *A´*

Esse nível BAIXO vai setar ou *resetar* o *FF B* para o mesmo estado do *FF A*





$\overline{PRESET}$	$\overline{CLEAR}$	Resposta do FF
1	1	Operação com clock*
0	1	Q = 1 (independente do CLK)
1	0	Q = 0 (independente do CLK)
0	0	Não usada

*Q responderá a JK e CLK

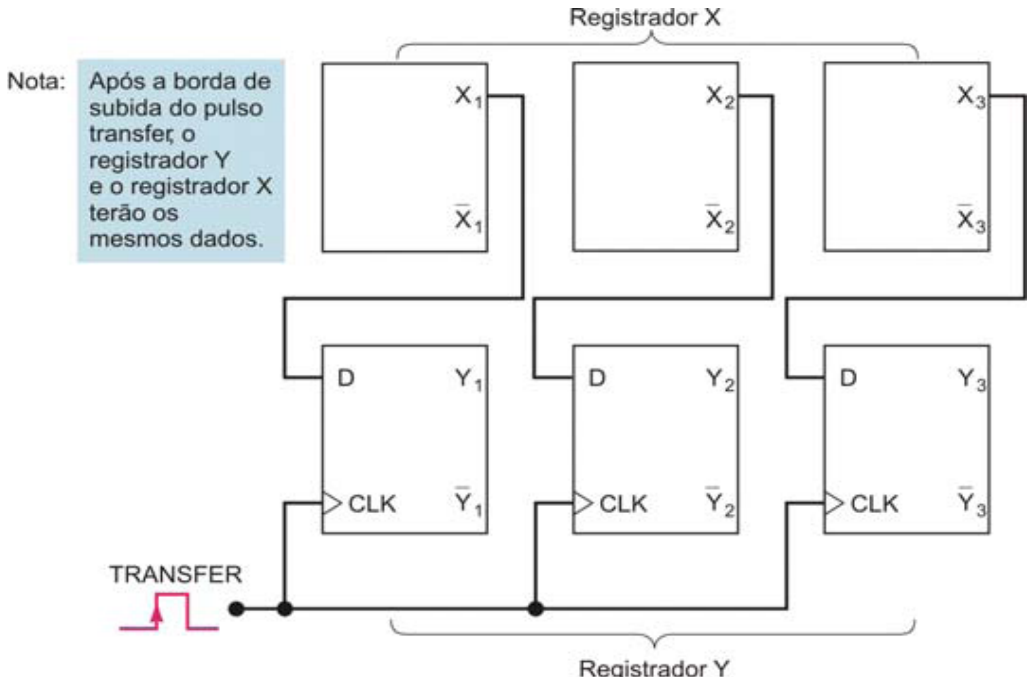
Transferência Paralela de Dados

A figura ilustra uma transferência de dados de um registrador para outro usando *FFs D*

Na aplicação da borda de subida do pulso TRANSFER, o nível armazenado em X_1 é transferido para Y_1

X_2 para Y_2 , e X_3 para Y_3

Trata-se de uma **transferência paralela** e **síncrona**



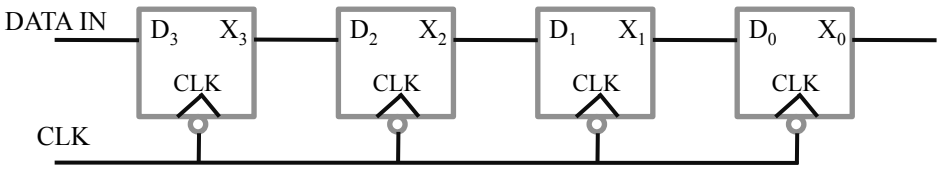
A **transferência paralela** não altera o conteúdo do registrador que é a fonte dos dados

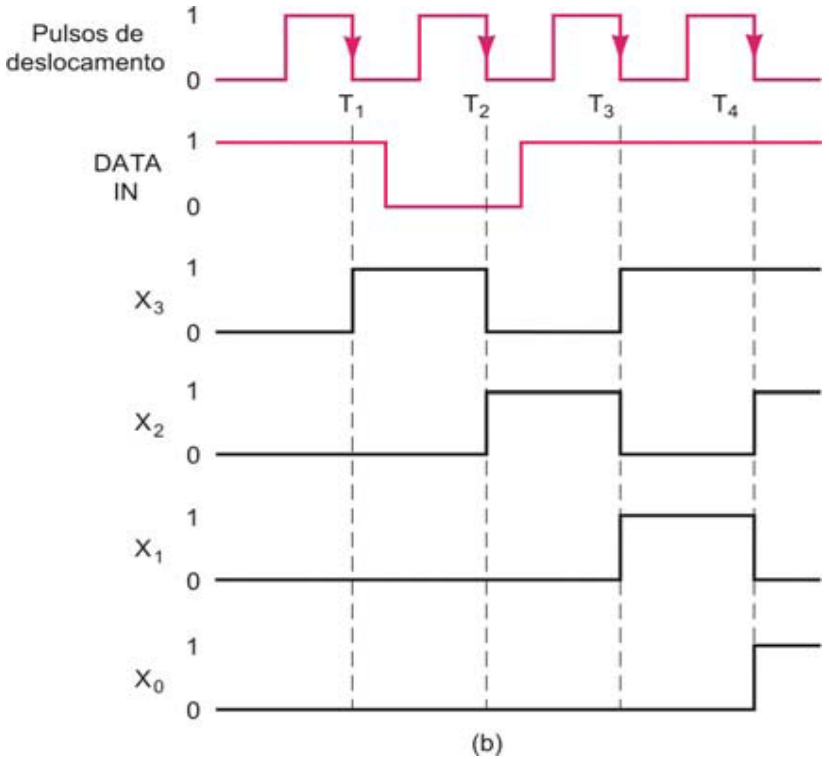
Registrador de Deslocamento

Um **Registrador de Deslocamento** é um grupo de *FFs* organizados de modo que os valores binários armazenados nos *FFs* sejam deslocados de um *FF* para o seguinte a cada pulso de *clock*

O valor da saída X_3 é transferido para X_2, o valor de X_2 para X_1, e o de X_1 para X_0

Dessa forma, quando ocorre uma borda de descida no pulso de deslocamento, cada *FF* recebe o valor armazenado previamente no *FF* à esquerda



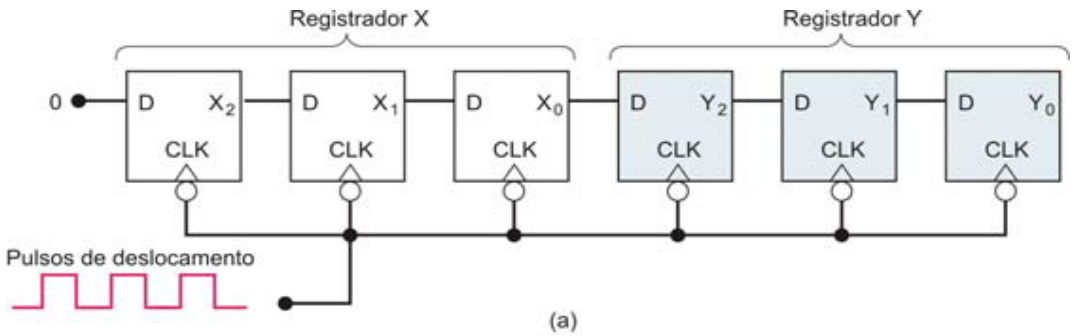


Transferência Serial de Dados

A transferência de dados entre registradores *X* e *Y* pode ser feita de modo serial com registradores de deslocamento usando *FFs D* (requerem menos conexões que os *FFs JK*)

Observe que X_0, o último *FF* do registrador *X*, está conectado à entrada *D* de Y_2, o primeiro *FF* do registrador *Y*

Quando os pulsos de deslocamento são aplicados, a transferência de informação acontece da seguinte forma: $X_2 => X_1 => X_0 => Y_2 => Y_1 => Y_0$



(a)

X_2	X_1	X_0	Y_2	Y_1	Y_0	
1	0	1	0	0	0	Antes dos pulsos serem aplicados
0	1	0	1	0	0	Após o primeiro pulso
0	0	1	0	1	0	Após o segundo pulso
0	0	0	1	0	1	Após o terceiro pulso

(b)

Divisão de Frequência e Contagem

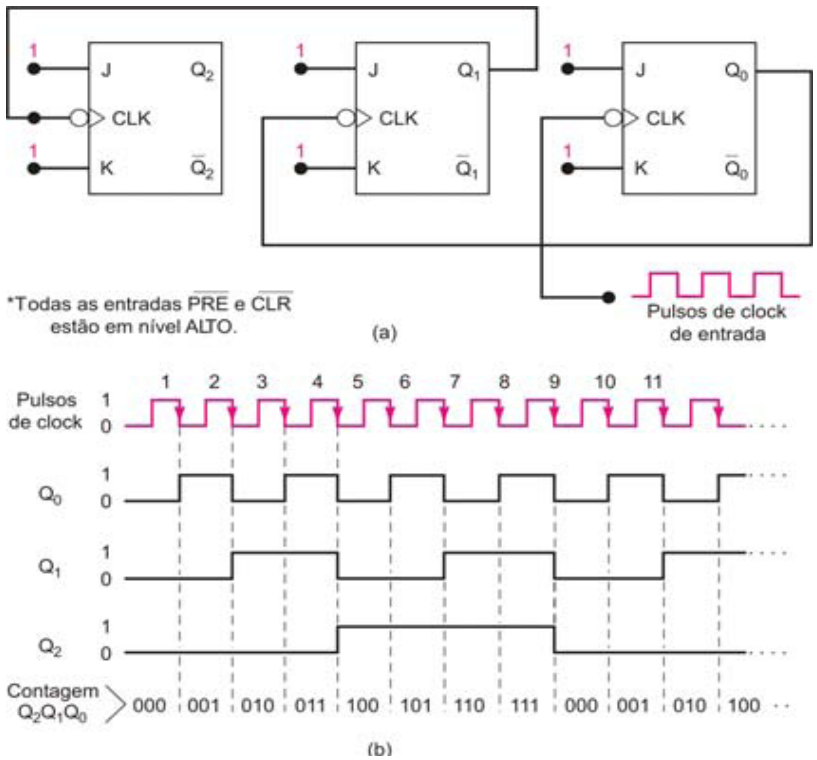
Considere a situação em que as duas entradas dos *FFs JK* estão em *1*, para que ele mude de estado sempre que o sinal em sua entrada *CLK* for do nível ALTO para BAIXO

Os pulsos de *clock* são aplicados apenas na entrada *CLK* do *FF* Q_0, de tal modo que Q_0 comuta na borda de descida de cada pulso na entrada de *clock*

Assim, a forma de onda da saída Q_0 tem uma frequência que é exatamente a metade da frequência dos pulsos do *clock*

Usando *N FFs*, a frequência de saída do último *FF* será igual a $1/2^N$ da frequência de entrada

Trata-se, portanto, de um **divisor de frequência**



Operação de Contagem

Além de funcionar como um divisor de frequência, o circuito anterior também funciona como um **Contador Binário**

A Tabela de Estados ao lado mostra que os primeiros oito estados de $Q_2Q_1Q_0$ devem ser reconhecidos como uma contagem binária sequencial de *000* a *111*

Após a primeira borda de descida do *clock*, os *FFs* passam para o estado *001* (Q_2=*0*, Q_1=*0*, Q_0=*1*) que representa 001_2

E assim sucessivamente ...

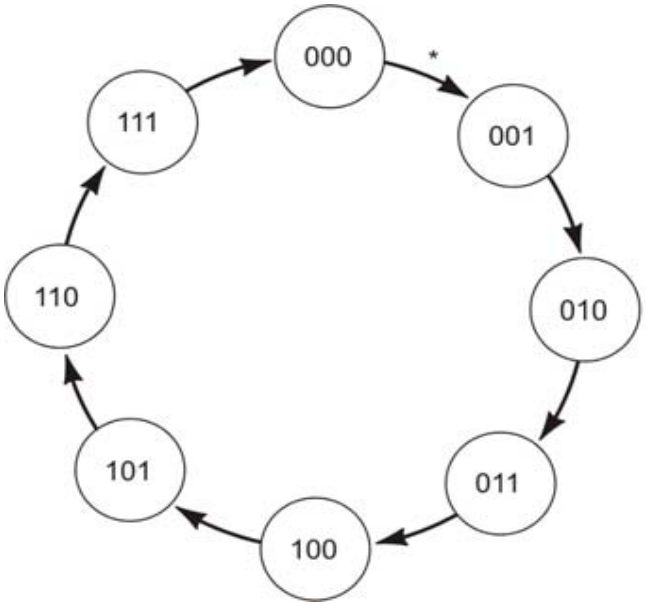
2^2 Q_2	2^1 Q_1	2^0 Q_0	
0	0	0	Antes de aplicar os pulsos de clock
0	0	1	Após o pulso #1
0	1	0	Após o pulso #2
0	1	1	Após o pulso #3
1	0	0	Após o pulso #4
1	0	1	Após o pulso #5
1	1	0	Após o pulso #6
1	1	1	Após o pulso #7
0	0	0	Após o pulso #8 retorna para 000
0	0	1	Após o pulso #9
0	1	0	Após o pulso #10
0	1	1	Após o pulso #11
.	.	.	. . .
.	.	.	. . .
.	.	.	. . .

Diagrama de Transição de Estados

Outra forma de mostrar como os estados dos *FFs* mudam a cada pulso de *clock* aplicado é através do **Diagrama de Transição de Estados**

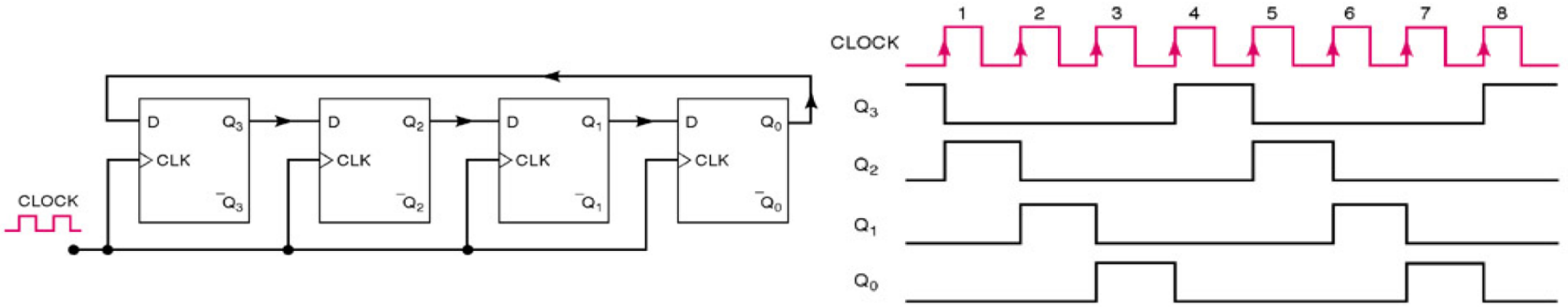
Observando o estado de um círculo em particular, vê-se qual é o estado anterior e o posterior

Por ex., observando o estado *000*, vê-se que ele é alcançado quando o contador está no estado *111*, e o pulso de *clock* é aplicado. Da mesma forma, vê-se que o estado *000* sempre é seguido pelo estado *001*



Como este contador tem $2^3=8$ estados diferentes, ele é denominado **contador de módulo 8**, sendo que o **valor do módulo** indica o **número de estados** da sequência de contagem

Contadores com Registradores de Deslocamento



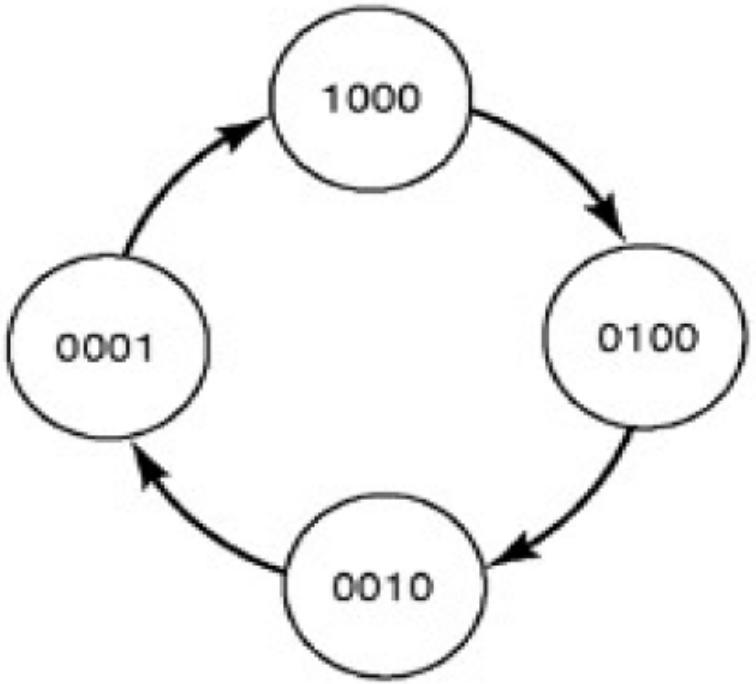
Contadores com registradores de deslocamento usam a realimentação, o que significa que a saída do último *FF* é conectada de volta ao primeiro *FF*. Trata-se, portanto, de um **registrador de deslocamento circular**

Na maioria dos casos, somente um único *1* está no registrador, e esse *1* circula pelo registrador enquanto pulsos de *clock* forem aplicados. Por esta razão, ele também é conhecido como **Contador em Anel**

Diagrama de Estados do Contador em Anel

Q_3	Q_2	Q_1	Q_0	CLOCK pulse
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7
.	.	.	.	.
.	.	.	.	.

(c)



(d)

Esse contador funciona como um **contador de módulo 4**, uma vez que ele tem **quatro estados distintos** antes que a sequência se repita

De modo geral, um **Contador em Anel** necessitará de mais *FFs* do que um **Contador Binário** de mesmo módulo. Por ex., um Contador em Anel de módulo 8 necessita de oito *FFs*, enquanto um Contador Binário de módulo 8 requer apenas três