Sistemas Digitais

Transferências entre Registradores

Referência Bibliográfica:

Logic and Computer Design Fundamentals – Mano & Kime

Adaptações: josé artur quilici-gonzalez

Sumário

- **Transferências entre Registradores**
  - Registrador
  - RTL – *Register Transfer Language*
  - Transferências baseadas em Multiplexador
  - Transferências baseadas em Barramento
  - Transferência serial
  - Micro-operações

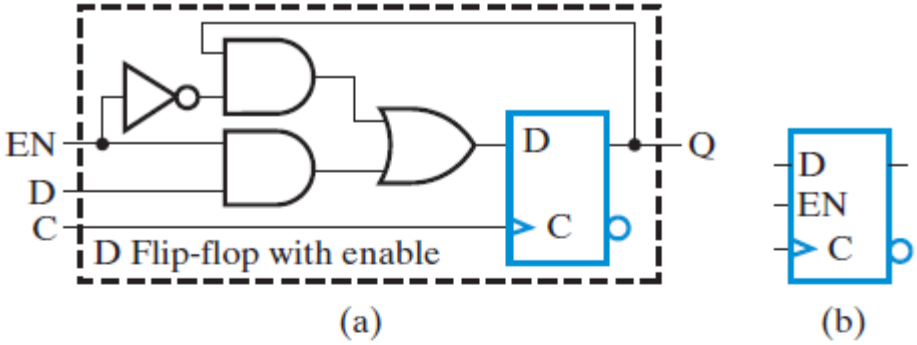
Registradores

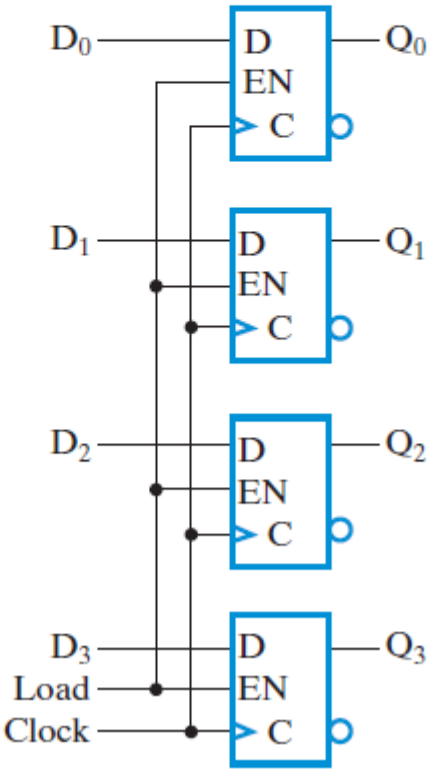
- **Registrador** – um conjunto de elementos de armazenagem binária (*Flip-Flops*)
- Em teoria, o comportamento de um registrador segue uma lógica sequencial que pode ser definida por uma **Tabela de Estados**
- É mais comum imaginar um registrador como o **armazenador de um vetor de valores binários**
- Frequentemente usado para o armazenamento, a movimentação e o processamento de dados

Armazenagem em Registrador

- **Expectativas:**
  - Um registrador pode armazenar **informação por múltiplos ciclos de *clock***
  - A operação de "armazenar" ou "carregar" informação deve ser controlado por um sinal
- **Realidade:**
  - Um registrador com *flip-flop D* carrega **informação nova a cada ciclo de *clock***
- **Como materializar as expectativas:**
  - Usando um sinal para bloquear o *clock* do registrador,
  - Usando um sinal para controlar o retorno da saída do registrador para suas entradas, ou
  - Usando *flip-flops SR* ou *JK,* que em (*0,0*) mantêm o **Estado Atual**
- ***Load*** ou **Carrega** é um nome frequente para o sinal que controla a armazenagem e carga do registrador
  - ***Load* = 1**: Carrega os valores das entradas de dados
  - ***Load* = 0**: Armazena (mantém) os valores no registrador

Exemplo de Registrador de 4 *Bits*



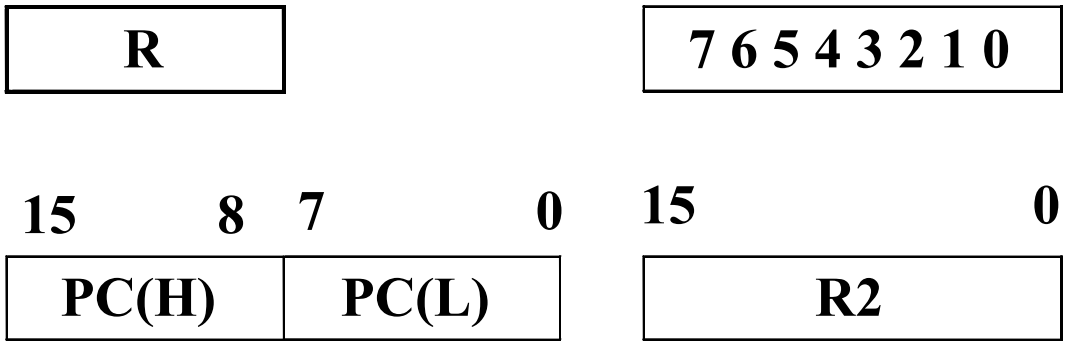


- O sinal **EN** seleciona entre o *bit* de entrada **D** e o *bit* **Q** de saída do FF
- Para **EN=1**, **D é selecionado** e um novo valor é carregado
- Para **EN=0**, **Q é selecionado** e a saída é carregada novamente no FF, preservando o Estado Presente

Transferência entre Registradores

- **Operações de Transferência entre Registradores** Compreendem a movimentação e o processamento de dados armazenados nos registradores
- **Três componentes básicos especificam as operações**:
  - conjunto de registradores no sistema
  - operações realizadas nos dados binários dos registradores
  - controle para supervisionar a sequência de operações
- **Operações Elementares** – *load* (carregar), *count* (contar), *shift* (deslocar), *add* (somar), *bitwise "OR"* (operação *OR bit* a *bit*) etc.
  - As operações elementares são chamadas ***micro-operações***

Notação para Registrador



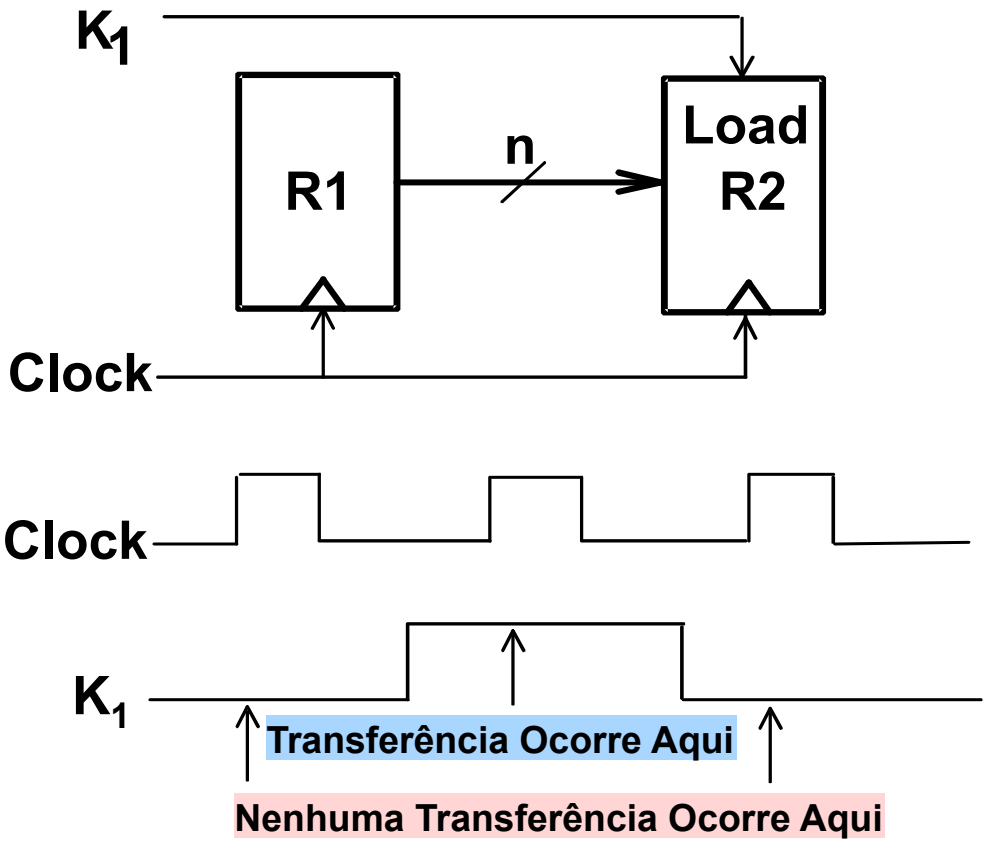
- **Letras e números** – denotam um registrador (ex. R2, PC, IR)
- **Parênteses ( )** – denotam uma faixa de *bits* do registrador (ex. R1(1), PC(7:0), AR(L))
- **Flecha (←)** – denota transferência de dado (R1 ← R2, PC(L) ← R0)
- **Vírgula** – separa operações paralelas
- **Colchetes [ ]** – Especificam um endereço da memória (ex. R0 ← M[AR], R3 ← M[PC])

Transferência Condicional

- *If* (K1 =1) *then* (R2 ← R1) é abreviado para

  K1: (R2 ← R1)

  em que K1 é uma **variável de controle** especificando uma execução condicional de uma **micro-operação**



Diferenças entre a Notação RTL e VHDL

Textbook RTL, VHDL, and Verilog Symbols for Register Transfers

Operation	Text RTL	VHDL	Verilog
Combinational assignment	=	= (sequential)	assign = (blocking)
Register transfer	←	<= (concurrent)	<= (nonblocking)
Addition	+	+	+
Subtraction	–	–	–
Bitwise AND	^	and	&
Bitwise OR	∨	or	|
Bitwise XOR	⊕	xor	^
Bitwise NOT	¯ (overline)	not	~
Shift left (logical)	sl	sll	<<
Shift right (logical)	sr	srl	>>
Vectors/registers	*A*(3:0)	*A*(3 down to 0)	*A*[3:0]
Concatenation	||	&	{ , }

Logic and Computer Design Fundamentals – Mano e Kime

Micro-operações

- **Agrupamento das Micro-operações por Afinidade**:
  - **Transferência** - move dados de um conjunto de registradores para outro
  - **Aritmético** – realiza operações aritméticas nos dados dos registradores
  - **Lógico** - manipula dados ou realiza operações lógicas entre *bits*
  - **Deslocamento *(Shift)*** – desloca dados nos registradores

Operações Aritméticas

+ Adição
– Subtração
* Multiplicação
/ Divisão

Operações Lógicas

∨ OU Lógico (*Logical OR*)
∧ E Lógico (*Logical AND*)
⊕ *Logical Exclusive OR*
— ou ' *Not*

Exemplo de Micro-operações

- Adicionar o conteúdo de R1 ao de R2 e colocar o resultado em R1

  R1← R1 + R2

- Multiplicar o conteúdo de R1 pelo de R6 e colocar o resultado em PC

  PC ← R1 * R6

- Fazer o Exclusive OR do conteúdo de R1 com o conteúdo de R2 e colocar o resultado em R1

  R1 ← R1 ⊕ R2

Exemplo de Micro-operações (Cont.)

- Fazer o Complemento de 1 do conteúdo de R2 e colocá-lo em PC
- **PC ← $\overline{\text{R2}}$**

- Sob a condição **K1 OR K2**, o conteúdo de R1 é o ***Logic bitwise ORed*** com o conteúdo de R3 e o resultado colocado em R1
- **(K1 + K2): R1 ← R1 ∨ R3**

- OBS: "+" (como em $K_1 + K_2$) significa "OR"
  Em R1 ← R1 + R3, "+" significa "*plus*" ou "mais"

Expressões de Controle

- A **expressão de controle** para uma operação aparece à esquerda da operação e é separada por dois pontos "**:**"
- Expressões de controle especificam a **condição lógica** para que a operação ocorra
- Valores da expressão de controle:
  - **Lógico "1"** -- a operação ocorre
  - **Lógico "0"** -- a operação não ocorre

- Exemplo:
- $\overline{X}\,K1 : R1 \leftarrow R1 + R2$
  $X\,K1 : R1 \leftarrow R1 + \overline{R2} + 1$
- A variável K1 habilita a operação, seja ela de adição ou subtração
- Se X = 0, então $\overline{X} = 1$, e $\overline{X}\,K1 = 1$, ativando a adição de R1 e R2
- Se X = 1, então X K1 = 1, ativando a adição de R1 e o complemento de 2 de R2 (subtração)

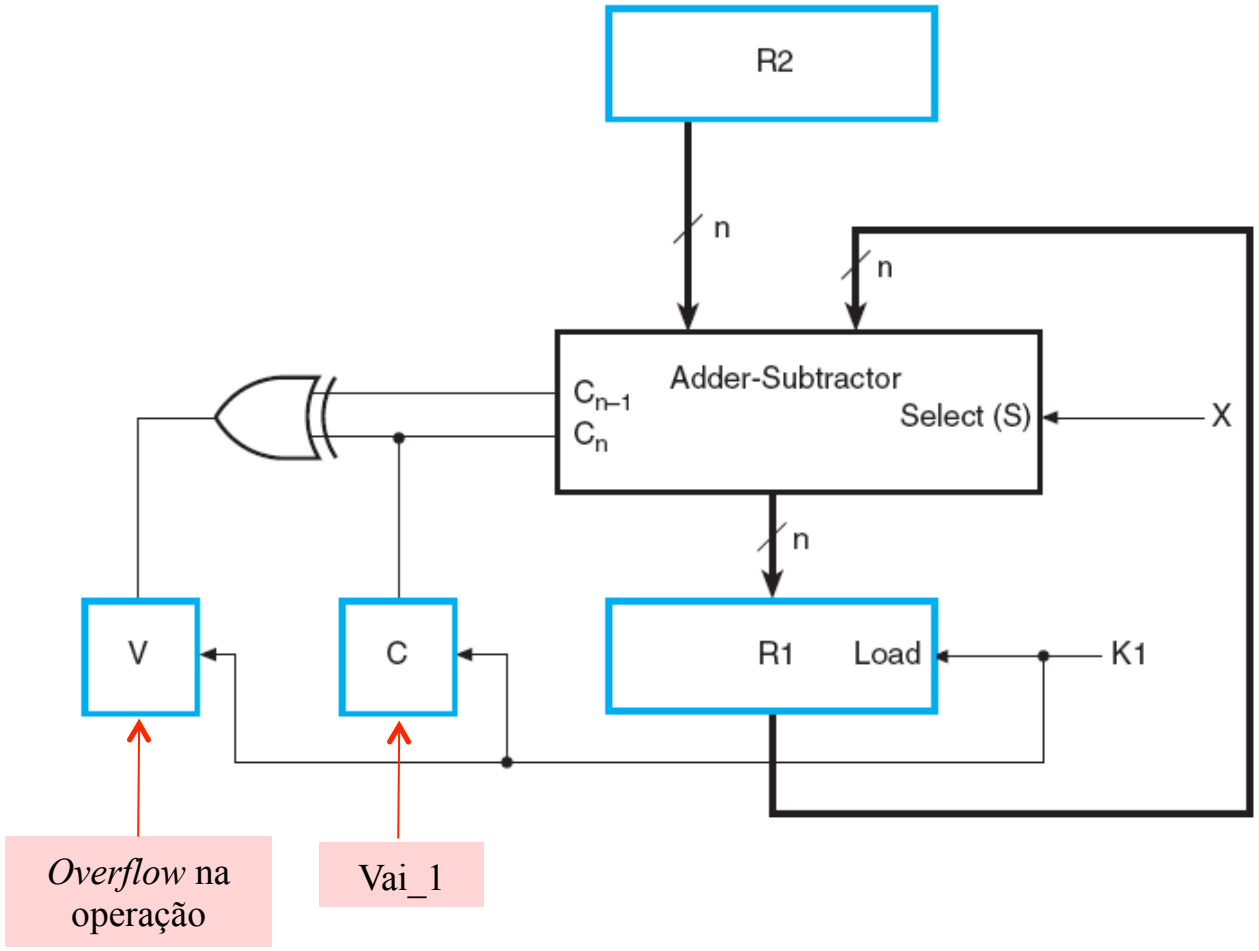
Expressões de Controle - Implementação

$\bar{X}K_1$: $R1 \leftarrow R1+R2$

XK_1: $R1 \leftarrow R1+\bar{R2}+1$

porém,

$\bar{X}K_1+XK_1=(\bar{X}+X)K_1=K_1$



A variável de controle X seleciona a operação (soma ou subtração) e a variável de controle K_1 carrega o resultado em $R1$, C e V

Micro-operações Aritméticas

Designação Simbólica	Descrição
$R0 \leftarrow R1 + R2$	Adição
$R0 \leftarrow \overline{R1}$	Complemento de 1
$R0 \leftarrow \overline{R1} + 1$	Complemento de 2
$R0 \leftarrow R2 + \overline{R1} + 1$	R2 menos R1 (Compl. de 2)
$R1 \leftarrow R1 + 1$	Incremento (*count up*)
$R1 \leftarrow R1 - 1$	Decremento (*count down*)

- Note que qualquer registrador pode ser especificado como fonte 1(ou origem 1), fonte 2, ou destino
- Estas micro-operações simples operam na palavra inteira

Micro-operações Lógicas

Designação Simbólica	Descrição
$R0 \leftarrow \overline{R1}$	*Bitwise NOT*
$R0 \leftarrow R1 \vee R2$	*Bitwise OR ("sets bits")*
$R0 \leftarrow R1 \wedge R2$	*Bitwise AND ("clears bits")*
$R0 \leftarrow R1 \oplus R2$	*Bitwise EXOR ("complements bits")*

- Micro-operações lógicas são úteis para manipular os *bits* armazenados nos registradores
- Essas operações consideram separadamente cada bit, que é visto como uma variável binária

Micro-operações Lógicas (cont.)

- Seja R1 = 10101010,
  e R2 = 11110000
- Então após a operação, R0 assume os valores:

R0	Operação
01010101	$R0 \leftarrow \overline{R1}$
11111010	$R0 \leftarrow R1 \vee R2$
10100000	$R0 \leftarrow R1 \wedge R2$
01011010	$R0 \leftarrow R1 \oplus R2$

Micro-operações de Deslocamento (*Shift*)

- Seja R2 = 11001001
- Então, após a operação,

  R1 assume os valores:

Designação Simbólica	Descrição
R1 ← sl R2	*Shift Left*
R1 ← sr R2	*Shift Right*

R1	Operação
10010010	R1 ← sl R2
01100100	R1 ← sr R2

- Estas operações de *shifts* completam com zeros as novas posições (*zero fill*)
- Às vezes, um *FF* separado é usado para fornecer um novo dado (*bit*) de entrada (*data shifted in*), ou para "guardar" os dados de saída (*data shifted out*)

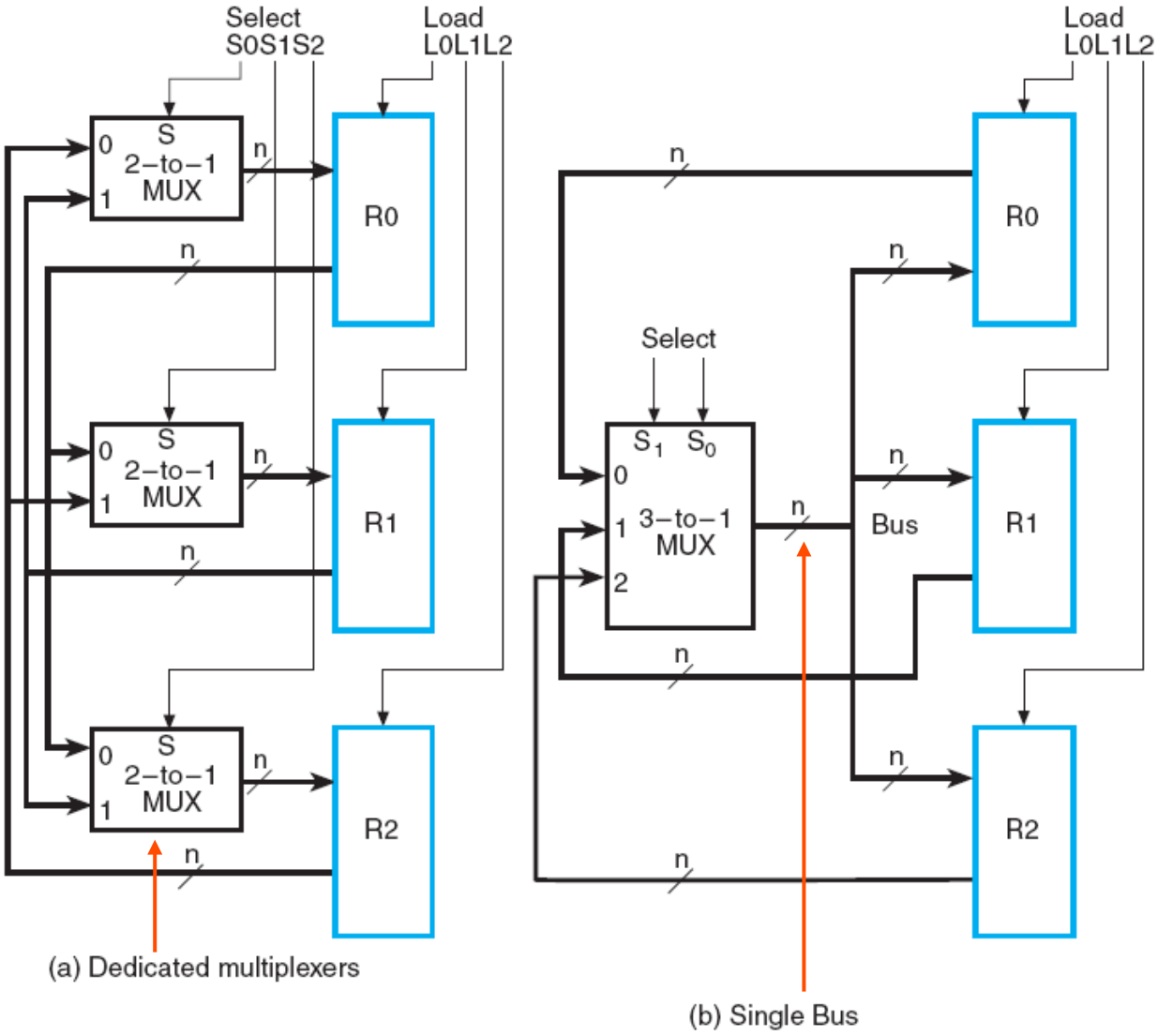
Estruturas para Transferência entre Registradores

- **Transferências Baseadas em Multiplexadores** – Múltiplas entradas são selecionadas por um multiplexador dedicado do registrador
- **Transferências Baseadas em *Bus*** – Múltiplas entradas são selecionadas por um multiplexador compartilhado comandando um *bus* (barramento) que alimenta as entradas de múltiplos registradores
- ***Three-State Bus*** - Múltiplas entradas são selecionadas por *drivers* de 3-estados (*3-state drivers*), cujas saídas se conectam a um *bus* que alimenta múltiplos registradores
- **Outras Estruturas de Transferência** - Uso de múltiplos multiplexadores, múltiplos *buses*, e combinações de todos os elementos acima citados

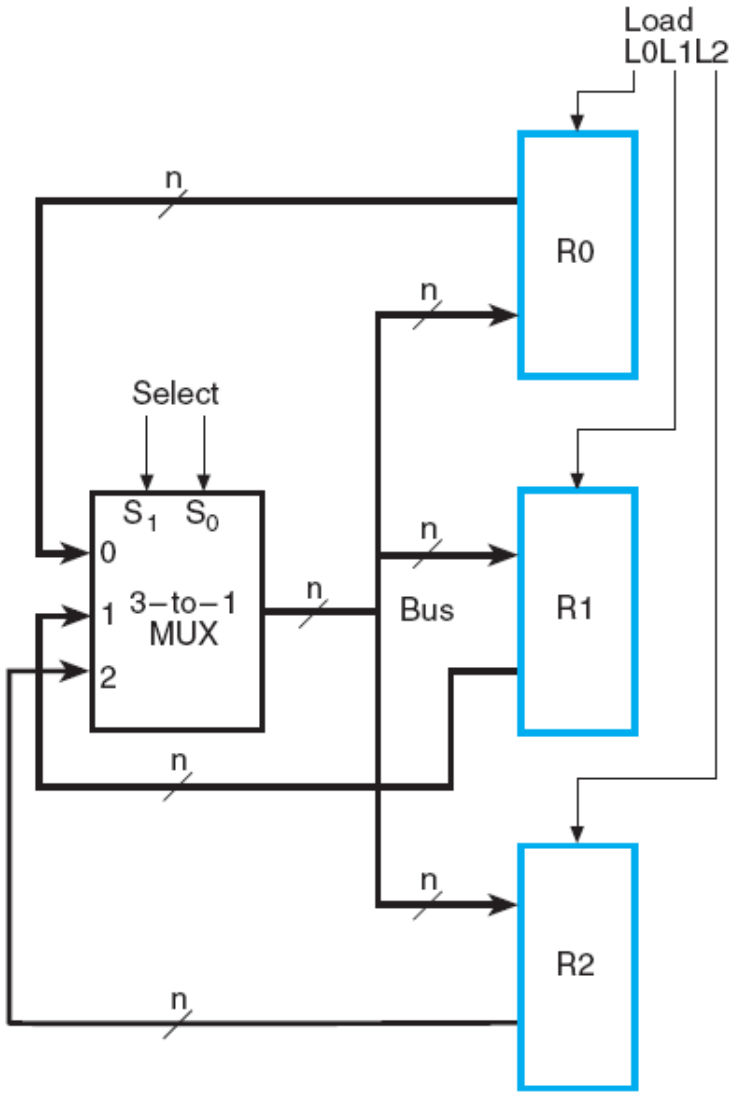
Multiplexadores Dedicados X Bus Único

Na solução com *bus* único há uma redução de *hardware* (os *muxes* dedicados são substituídos por um *bus*) porém com limitações nas opções de transferências simultâneas

Por ex., R0←R1, R1←R0 só é possível implementar na arquitetura com multiplexadores dedicados



Exemplo de Transferências entre Registradores



Register Transfer	Select S1	Select S0	Load L2	Load L1	Load L0
$R0 \leftarrow R2$	1	0	0	0	1
$R0 \leftarrow R1,\ R2 \leftarrow R1$	0	1	1	0	1
$R0 \leftarrow R1,\ R1 \leftarrow R0$	Impossible				

A tabela acima mostra como as **variáveis de controle** precisam ser ajustadas para a realização de algumas transferências entre registradores (com **implementação de *bus* único**)

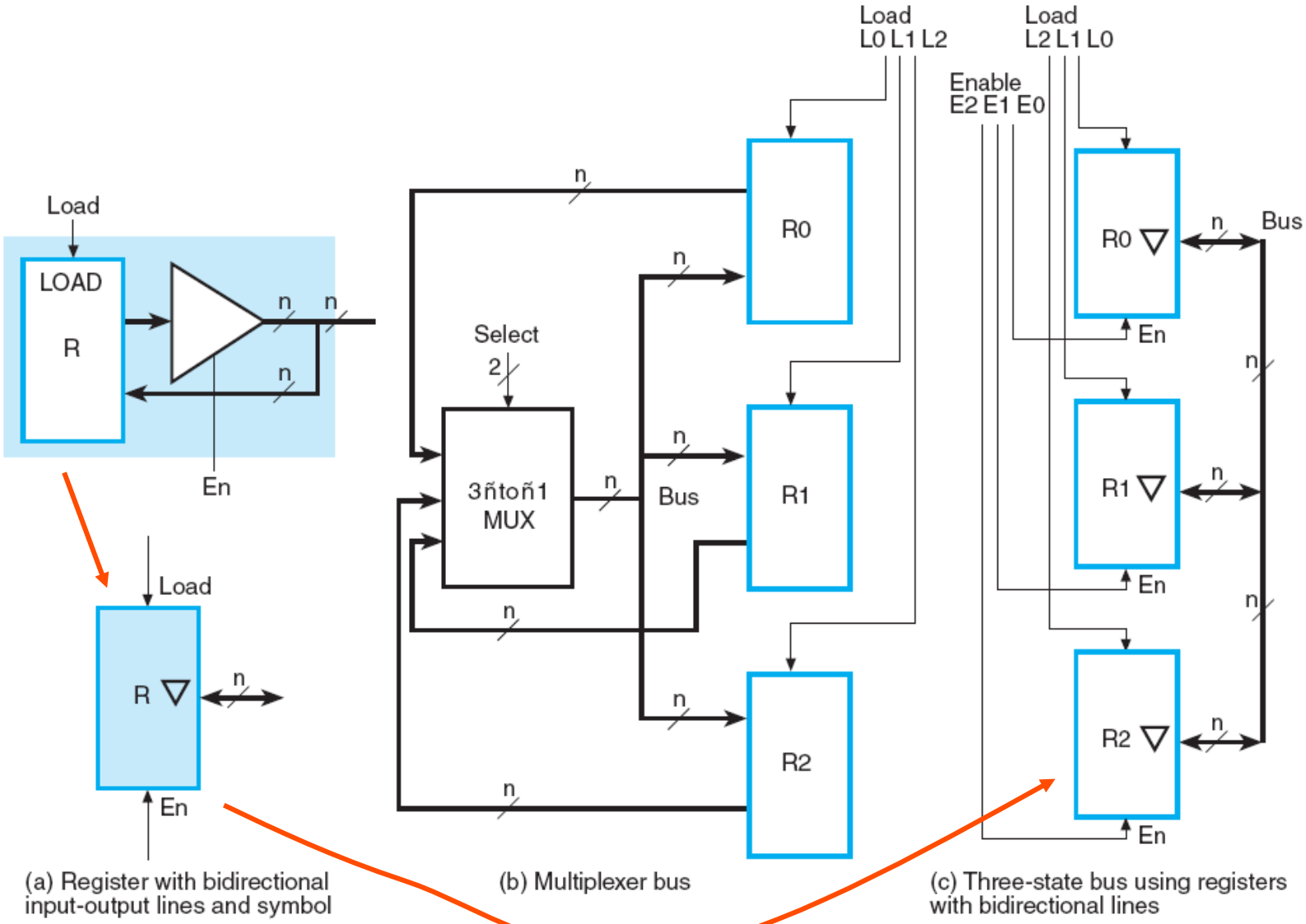
Note que estas transferências, efetivamente, ocorrerão na próxima transição positiva do *clock* (não mostrado neste diagrama)

Mas, R0←R1, R1←R0 só é possível realizar na **implementação com multiplexadores dedicados**

Logic and Computer Design Fundamentals – Mano e Kime

Bus com *Mux* X *Bus* com *Three-state Drivers*

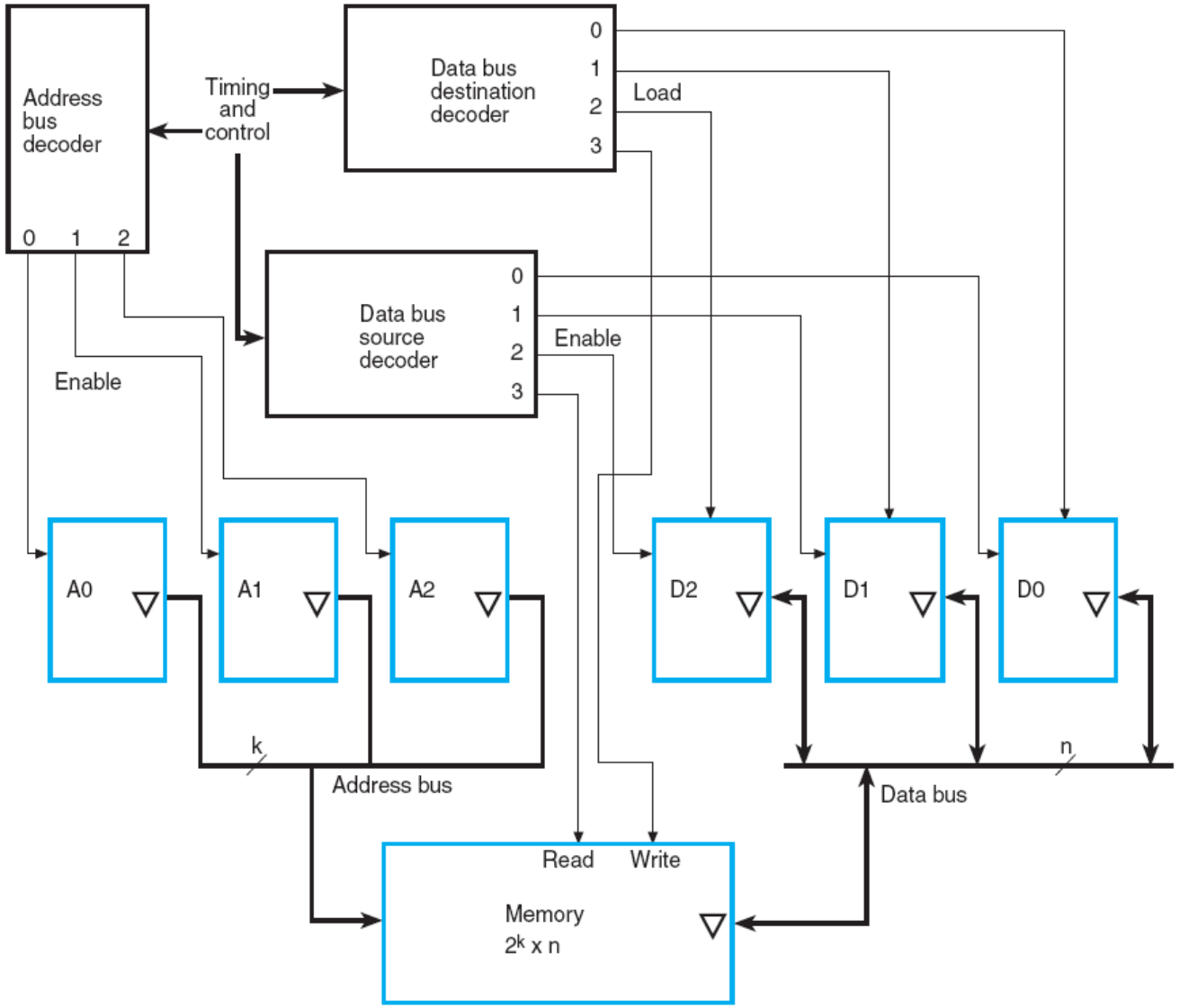
O uso do *three-state bus*, com conexões bidirecionais de entrada-saída, é particularmente efetivo entre circuitos lógicos em encapsulamentos físicos distintos



Transferências Envolvendo Memória

Como os **Barramentos de Dados** e **de Endereço** são acessados por dispositivos em encapsulamentos físicos distintos, estes barramentos são implementados usando registradores com *buffers* de três estados

Note que existe a possibilidade da **troca direta** de dados entre os **Registradores de Dados** (D0, D1, D2), enquanto a Memória permanece inativa

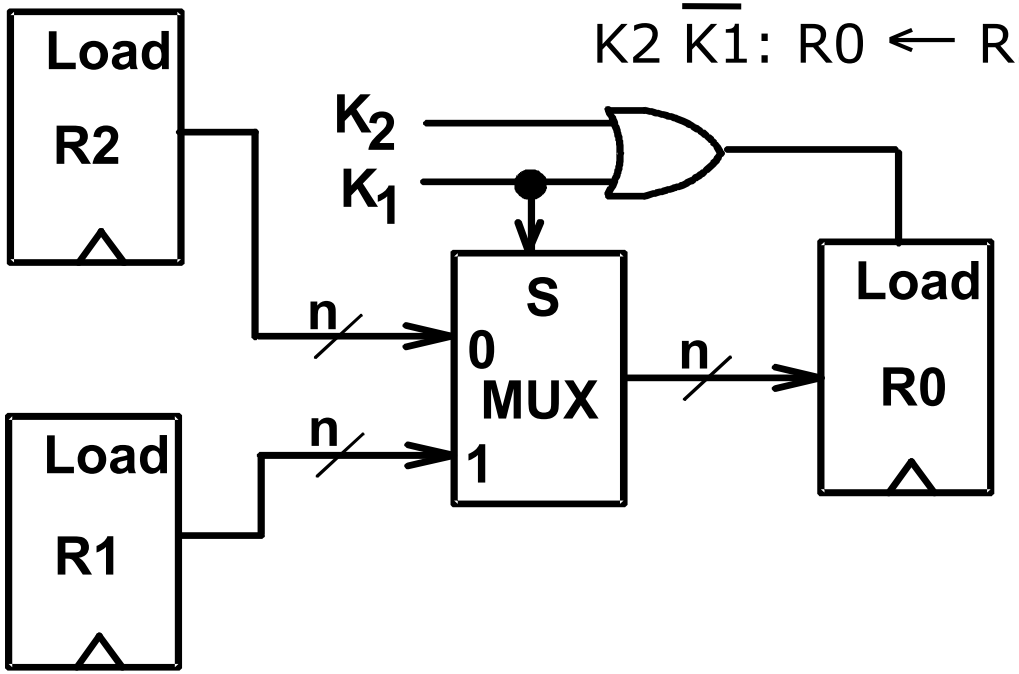


Ex. de Transferências
Write: M[A1] ←D2
Read: D1 ← M[A2]

Transferências Baseadas em Multiplexador

- Multiplexadores conectados às entradas dos registradores produzem estruturas de transferência flexíveis (Obs: os *clocks* foram omitidos por questão de clareza)
- As transferências são: K1: R0 ← R1

  K2 $\overline{K1}$: R0 ← R2

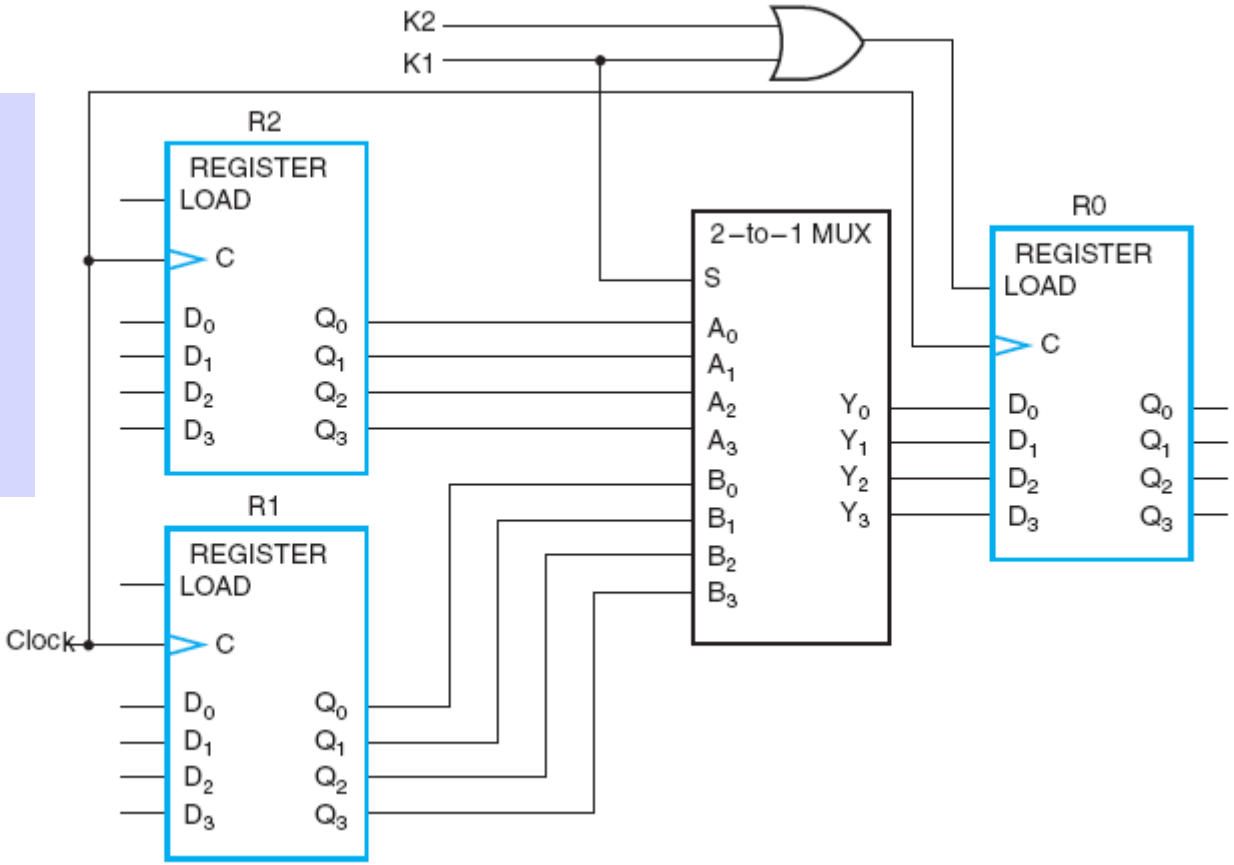


Transferências Baseadas em Multiplexador
Diagrama Lógico

If (K_1=*1*) then (*R0*←*R1*)
else
if (K_2=*1*) then (*R0*←*R2*)

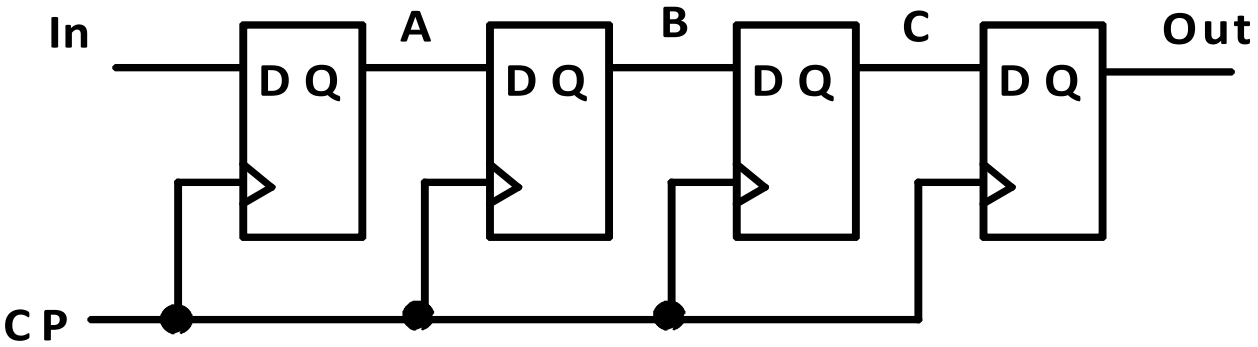
Esta expressão é equivalente a:

K_1:*R0*←*R1*, $\bar{K}_1 K_2$:*R0*←*R2*



Registradores de Deslocamento

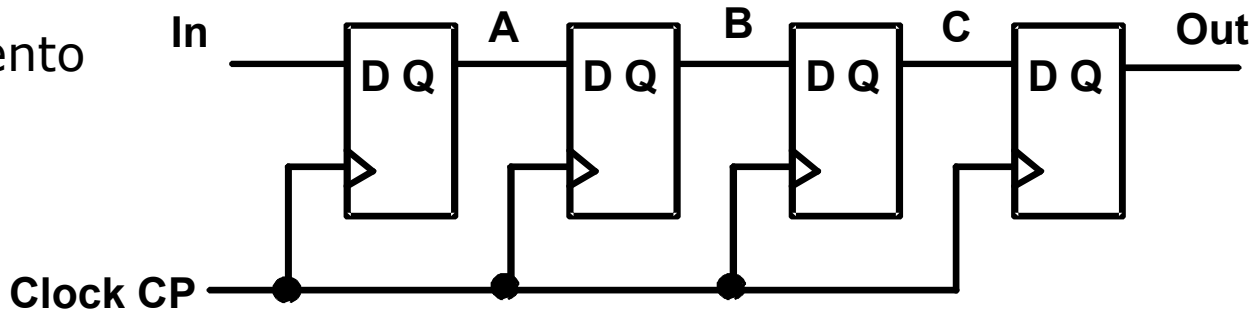
- **Registradores de Deslocamento** (*Shift Registers*) movimentam dados lateralmente dentro do registrador em direção à posição MSB ou LSB
- No caso mais simples, o registrador de deslocamento é simplesmente um conjunto de *FF D* conectados em linhas como este:



- A entrada de dados, *In*, é chamada *entrada serial ou shift right input*
- A saída de dados, *Out*, é frequentemente chamada *saída serial*
- O vetor (A, B, C, Out) é chamado *saída paralela*

Registradores de Deslocamento (cont.)

- O comportamento de um registrador de deslocamento serial é observado na saída da direita
- T0 é o estado do registrador antes da ocorrência do primeiro pulso de *clock*
- T1 é o estado depois do primeiro pulso e antes do segundo
- Estados inicialmente desconhecidos são marcados com "?"



CP	In	A	B	C	Out
T0	0	?	?	?	?
T1	1	0	?	?	?
T2	1	1	0	?	?
T3	0	1	1	0	?
T4	1				
T5	1				
T6	1				

Registradores de Deslocamento com Carga Paralela

- Acrescentado-se um *mux* entre cada estágio do registrador de deslocamento, os dados podem ser deslocados ou carregados
- Se o *SHIFT* for BAIXO, A e B são substituídos pelos dados nas linhas de D_A e D_B, caso contrário, os dados se deslocam para a direita a cada *clock*
- Acrescentando-se mais *bits*, é possível construir registradores de deslocamento com carga paralela de *n*-bit