



数字逻辑

第五章 数字硬件实现 ——寄存器

北京理工大学计算机学院

提纲

1 数字硬件实现

2 寄存器及传输

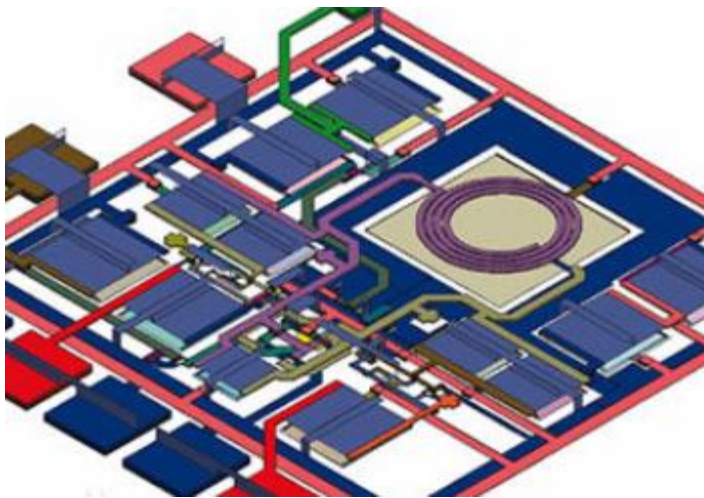
3 多寄存器传输

4 寄存器传输控制

5 小结

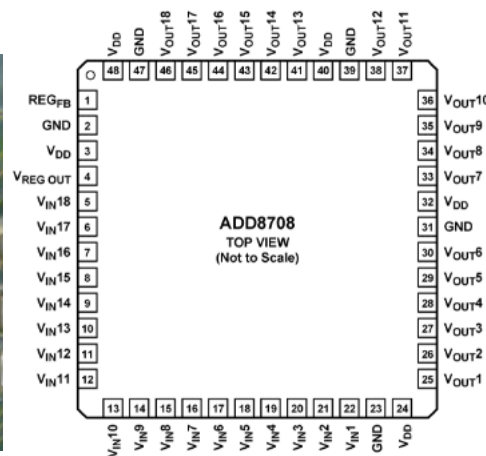
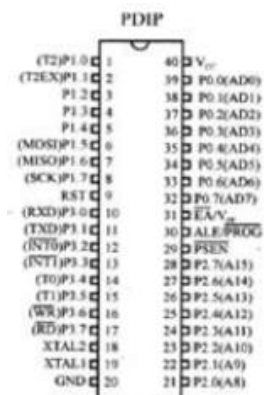
1 集成电路

- 数字电路是采用集成电路（IC）构建而成
- IC是一种硅半导体晶体，俗称芯片，包含实现逻辑门和存储单元的电子元件
- 芯片通过表面上的包含字母和数字的名字辨识



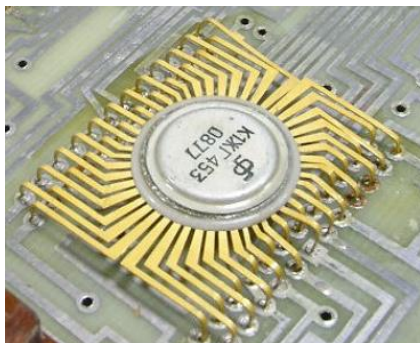
1.1 集成电路

- 芯片采用陶瓷或塑料封装，由内部电路引出与外部电路相连的接线（引脚）
- 引脚构成了芯片的接口
 - 脚跟、脚趾、脚侧



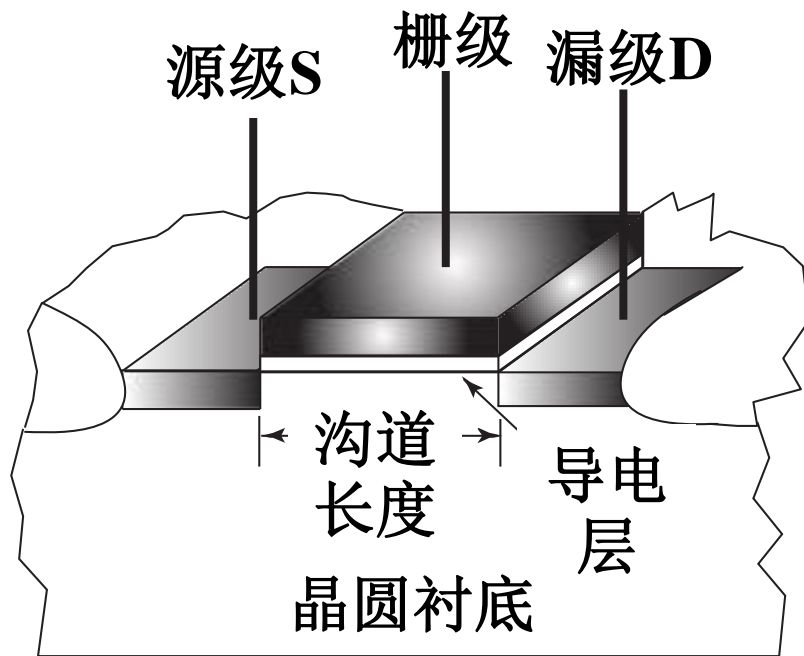
1.1 集成电路

- 按照逻辑门的集成度区分
 - 小规模集成（SSI）：几个独立的基本门
 - 中规模集成（MSI）：10~100个门
 - 大规模集成（LSI）：100~几千个门
 - 超大规模集成（VLSI）：几千~几亿个门



1.2 CMOS电路

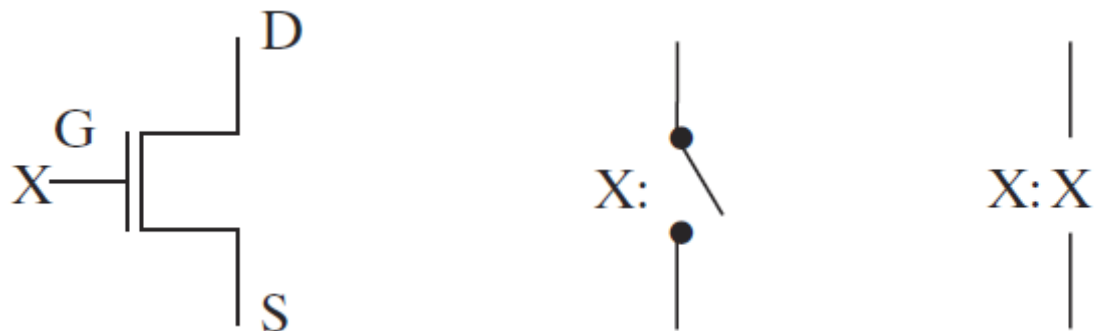
- 采用硅的互补金属氧化物半导体（CMOS）作为工艺实现的集成电路
- 具有密度高、性能高、功耗低的特点
- CMOS的工艺基础是MOS晶体管



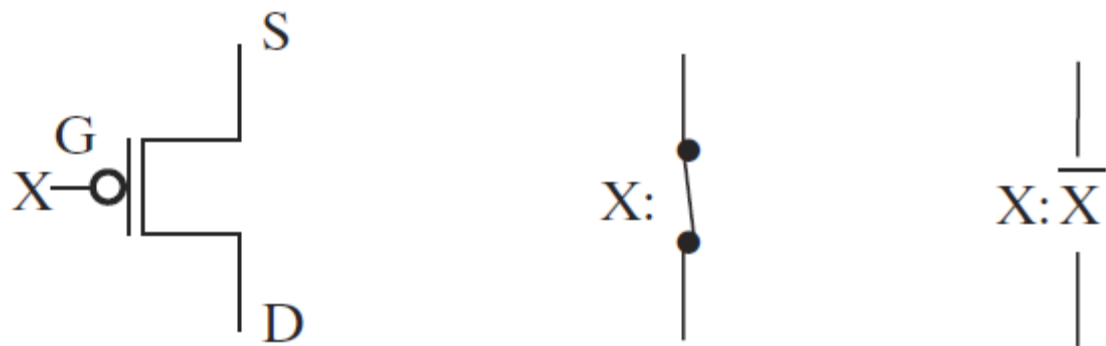
1.2 CMOS电路

■ 两种CMOS晶体管模型

- n沟道：触点习惯上为常开

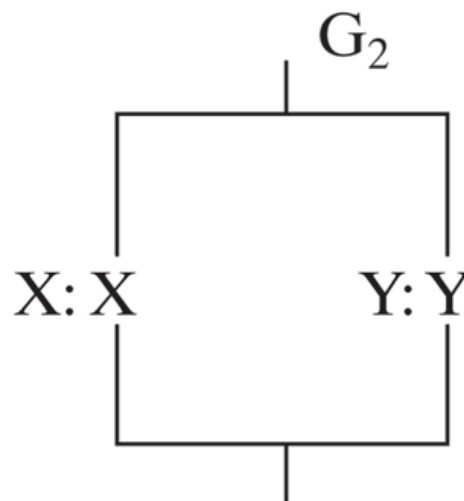


- p沟道：源级和漏级位置互换，触点习惯上为常闭



1.2 CMOS电路

- CMOS是大多数集成电路（电子电路）的实现基础
- 例：开关电路

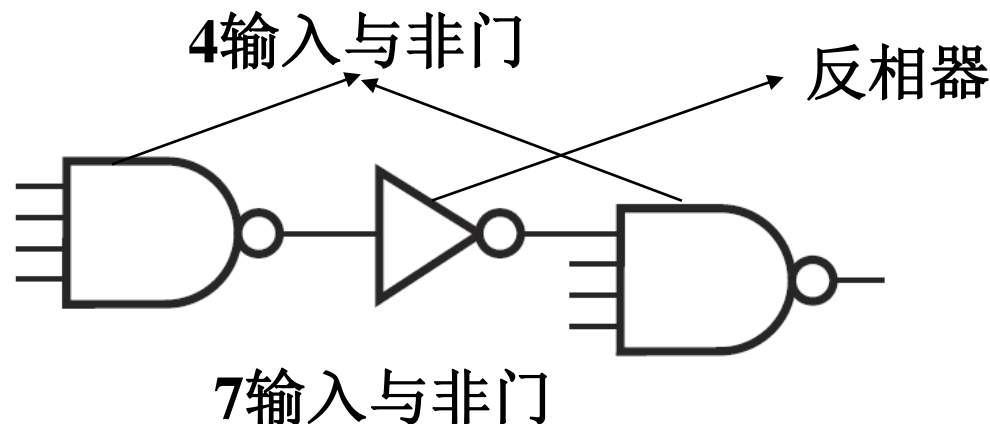


1.3 设计空间

- 电路设计是将逻辑门实现的逻辑电路映射到晶体管实现的电子电路，是从逻辑级到电路级的转换过程
- 实现特定设计的技术定义了可用的基本元件及其属性，而设计空间描述了所用技术及其参数的集合
 - 扇入、扇出、噪声容限、门的成本、传播延迟、功耗等工艺参数

1.3 设计空间

- 扇入：一个门可能的输入数
 - 通常不超过4~5个
 - 大扇入门使用低扇入门连接而成
- 扇出：一个门输出驱动的标准负载数
 - 大扇出门使用多个门并行实现
- 成本：晶体管大小、数目等因素



1.4 可编程逻辑器件

- 可编程逻辑器件（PLD）包含了用来实现逻辑功能的结构和控制内部连接或存储所需的信息。
- 可以通过硬件编程（programming）实现新功能
 - 只读存储器（ROM）
 - 可编程逻辑阵列（PLA）
 - 可编程阵列逻辑（PAL）
 - 现场可编程门阵列（FPGA）

1.4 可编程逻辑器件

■ 可编程实现技术

- 熔丝

- 反熔丝

- 掩膜编程

- 编程点的存储单元

- 晶体管开关

永久性、固化编程

可重编程

1.4 可编程逻辑器件

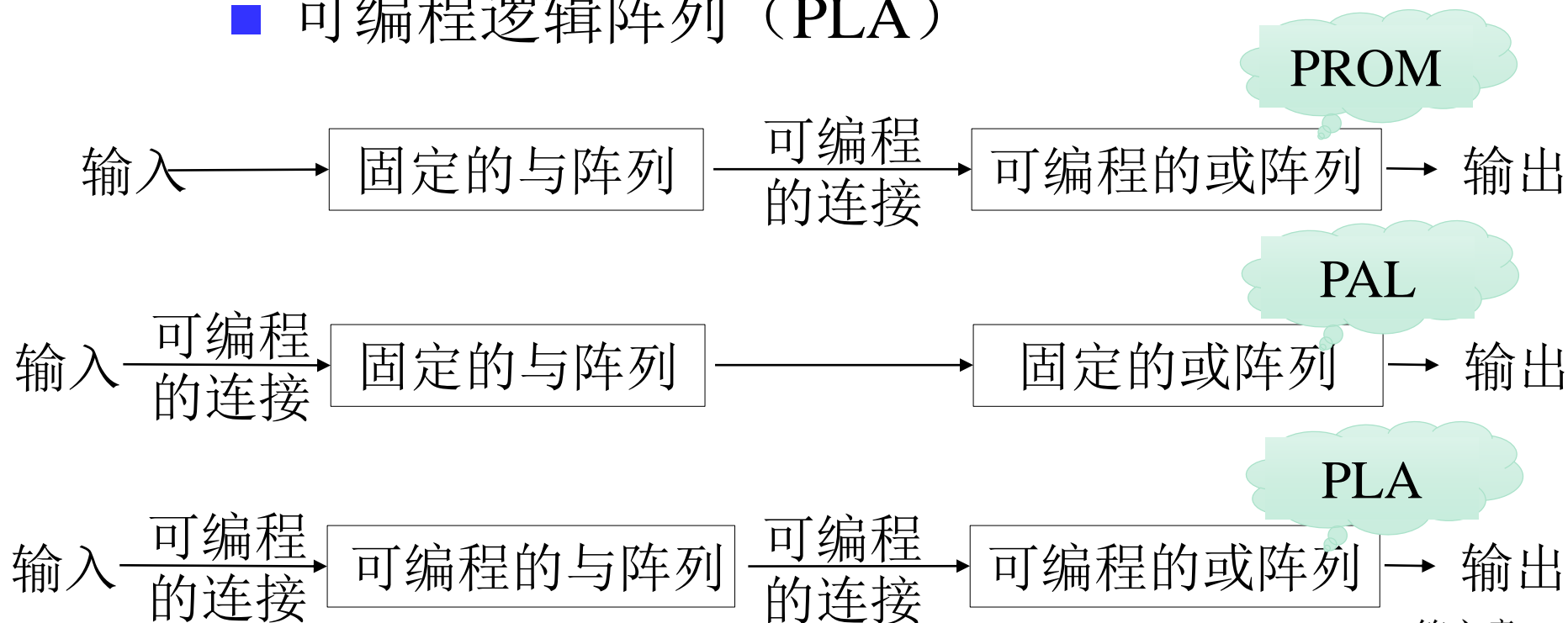
■ 晶体管开关

- 光可擦除（UVEPROM）：采用紫外线照射，浮置栅上的电荷即可泄漏掉。
- 电可擦除（EEPROM）：采用高于正常值的电压擦除。
 - 闪存技术（Flash）：EEPROM能在字节水平上进行删除和重写而不是整个芯片擦写，而闪存的大部分芯片需要块擦除

1.4 可编程逻辑器件

■ 可编程器件的三种基本结构

- 可编程只读存储器（PROM）
- 可编程阵列逻辑（PAL）
- 可编程逻辑阵列（PLA）



1.4 可编程逻辑器件

■ 只读存储器（ROM）

- 本质上是“永久”存储二进制信息的器件
- 非易失性（断电/通电）

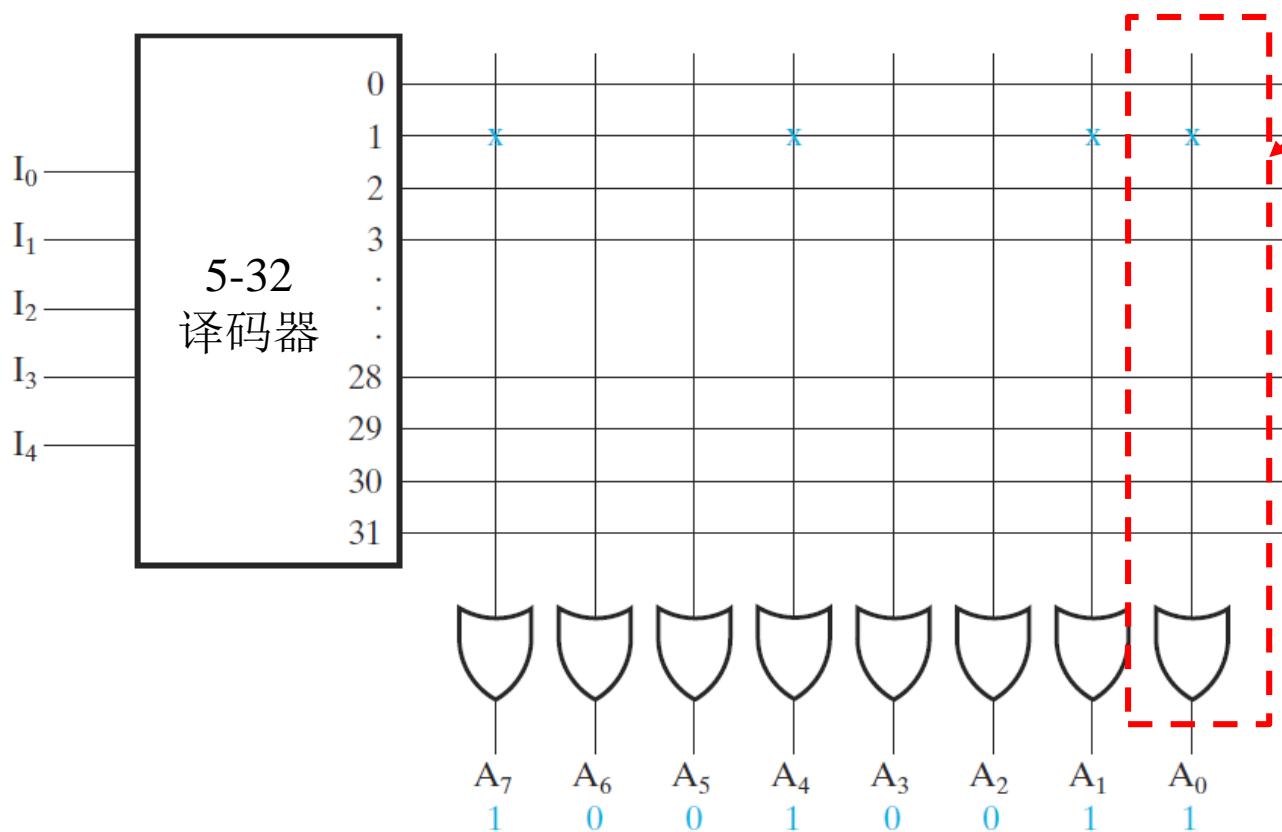


- 输入提供地址
- 输出提供由地址选定的存储字的数据位
- 没有数据输入（写操作）

1.4 可编程逻辑器件

■ 只读存储器（ROM）

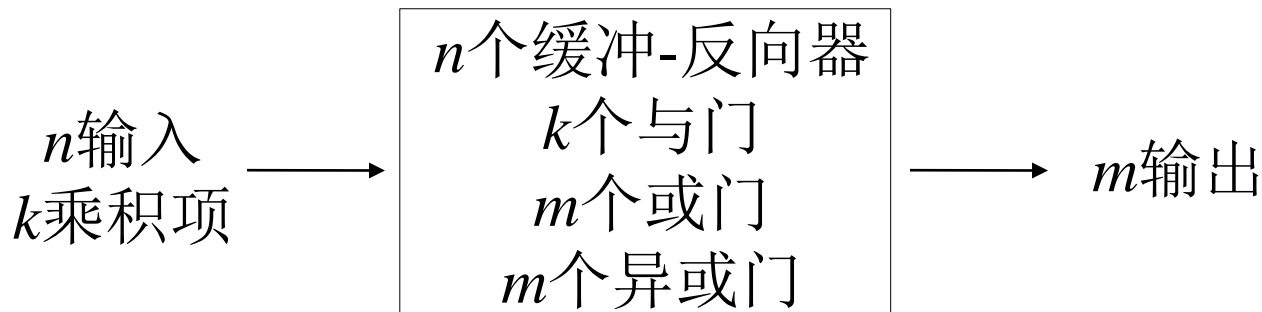
■ 例： $2^5 \times 8$ ROM内部逻辑结构



1.4 可编程逻辑器件

■ 可编程逻辑阵列（PLA）

- 与门阵列代替译码器，不提供变量的全译码
- 通过编程阐释输入变量的乘积项

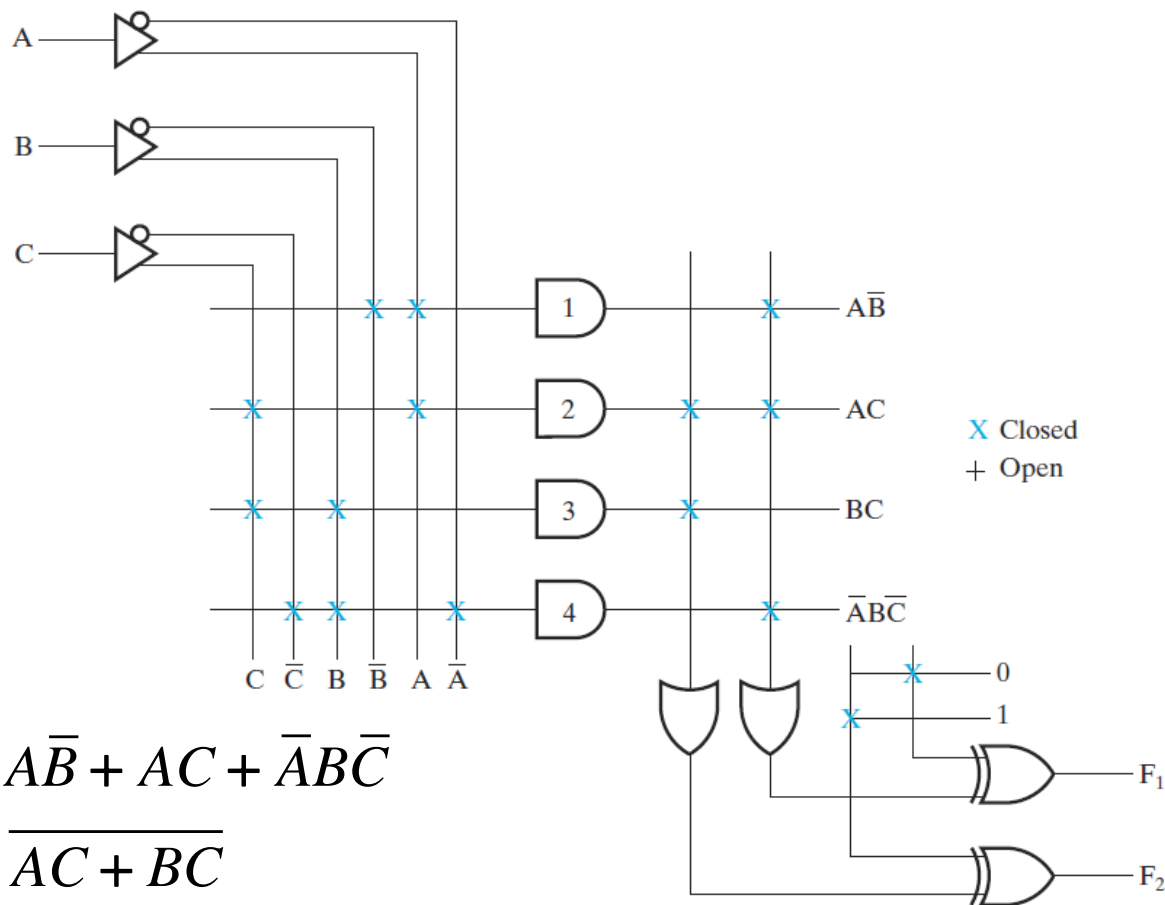


- 输入和与门之间有 $2n \times k$ 可编程连接点
- 与门和或门之间有 $k \times m$ 可编程连接点
- 或门和异或门之间有 m 可编程连接点

1.4 可编程逻辑器件

■ 可编程逻辑阵列 (PLA)

■ 例：3个输入、2个输出的内部逻辑图

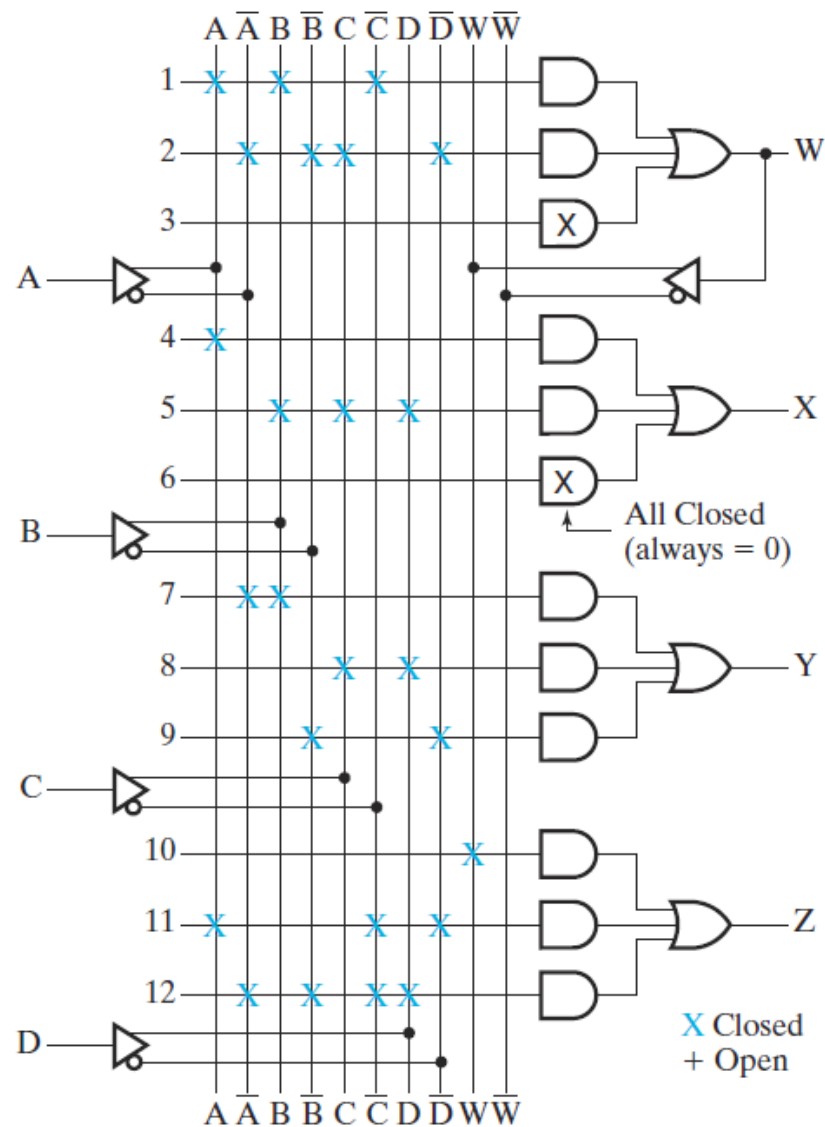


$$F_1 = AB + AC + \bar{A}BC$$

$$F_2 = \overline{AC + BC}$$

1.4 可编程逻辑器件

- 可编程阵列逻辑器件 (PAL)
 - 或门阵列固定、与门阵列可编程
 - 无法共享与门输出



1.4 可编程逻辑器件

■ 可编程阵列逻辑器件 (PAL)

■ 例:

$$W = \sum m(2, 12, 13)$$

$$X = \sum m(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$Y = \sum m(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

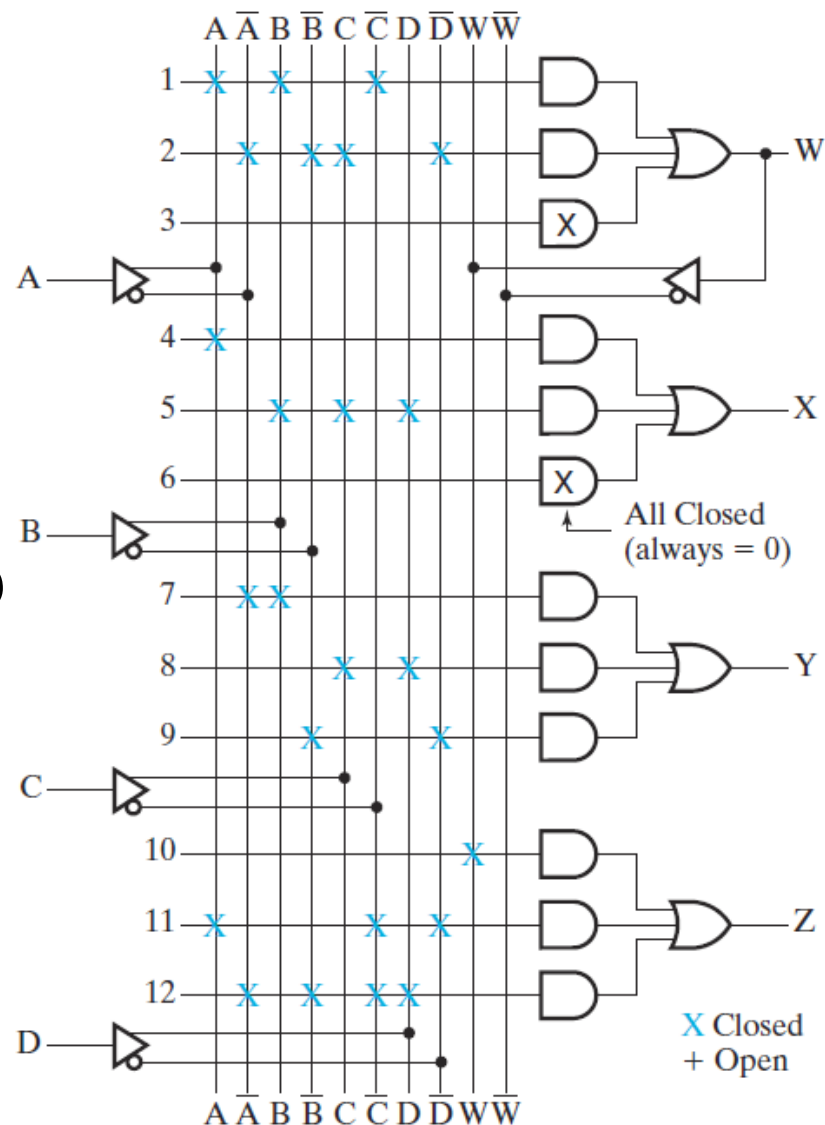
$$Z = \sum m(1, 2, 8, 12, 13)$$

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

$$X = A + BCD$$

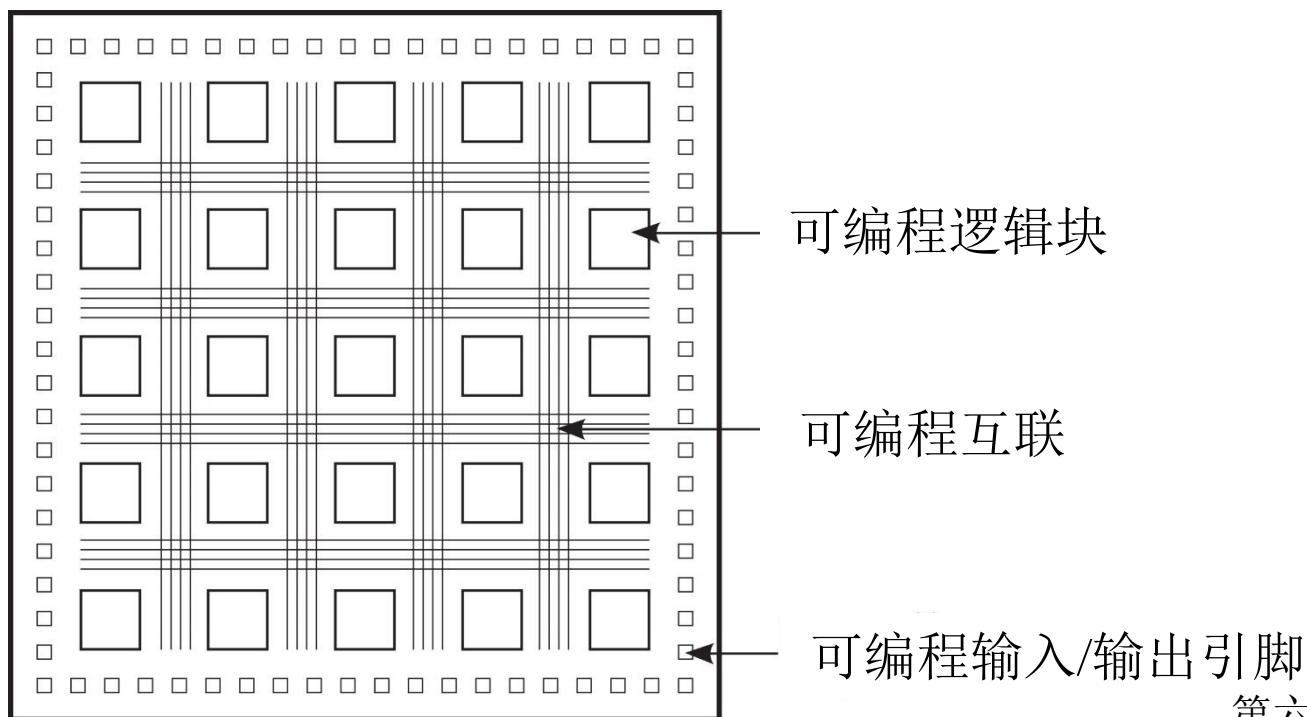
$$Y = \bar{A}B + CD + \bar{B}\bar{C}$$

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



1.4 可编程逻辑器件

- 现场可编程门阵列（FPGA）
 - 包含可编程逻辑块、可编程互联和可编程输入/输出引脚等三个可编程部件
 - 可采用反熔丝、闪存、SRAM等方式进行配置



1.4 可编程逻辑器件

■ 现场可编程门阵列（FPGA）

■ 可编程逻辑块

- 采用查找表（Look-Up Table, LUT）实现组合逻辑函数
- $2^k \times 1$ 的存储器记录带有 k 个变量的函数的真值表



香农展开式定理

$$f(x_1, x_2, \dots, x_k) = x_k \cdot f(x_1, x_2, \dots, x_{k-1}, 1) + \bar{x}_k \cdot f(x_1, x_2, \dots, x_{k-1}, 0)$$



2输入查找表



多路复用器查找表

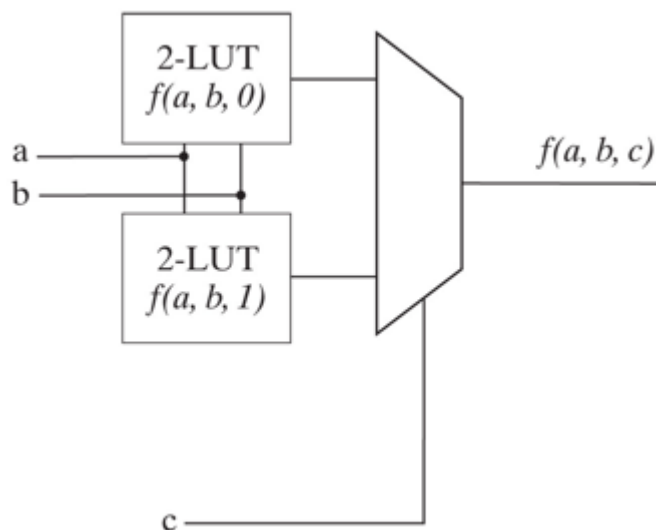
1.4 可编程逻辑器件

- 现场可编程门阵列（FPGA）

- 可编程逻辑块

- 例：用多路复用器实现布尔函数

$$F(A, B, C) = \sum m(3, 5, 6, 7)$$

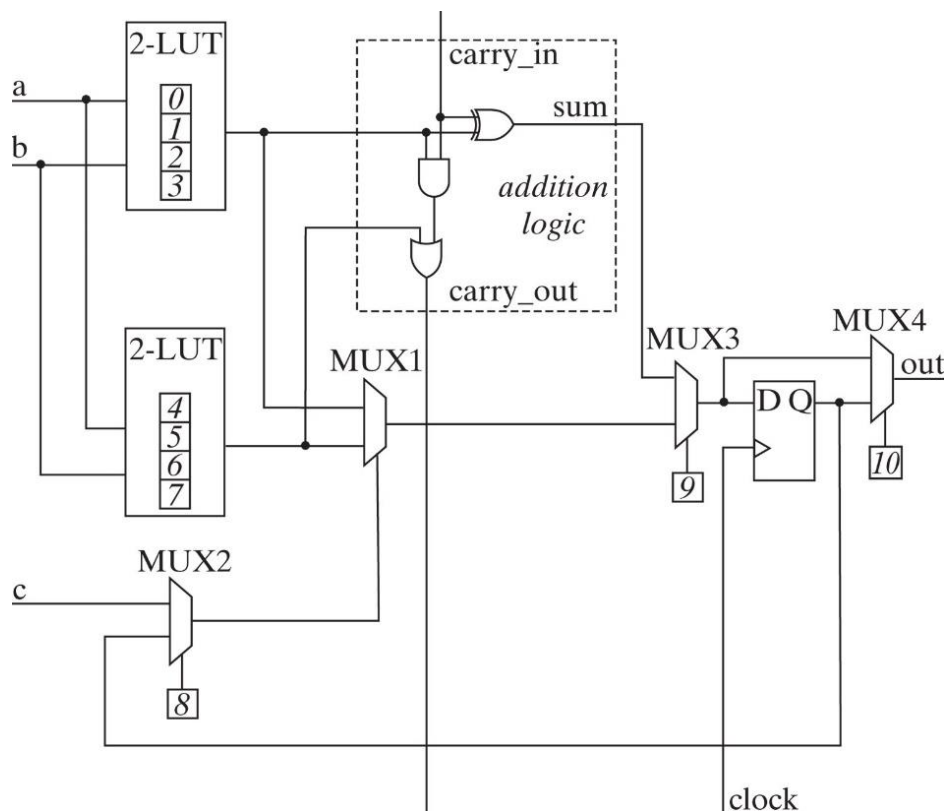


1.4 可编程逻辑器件

■ 现场可编程门阵列（FPGA）

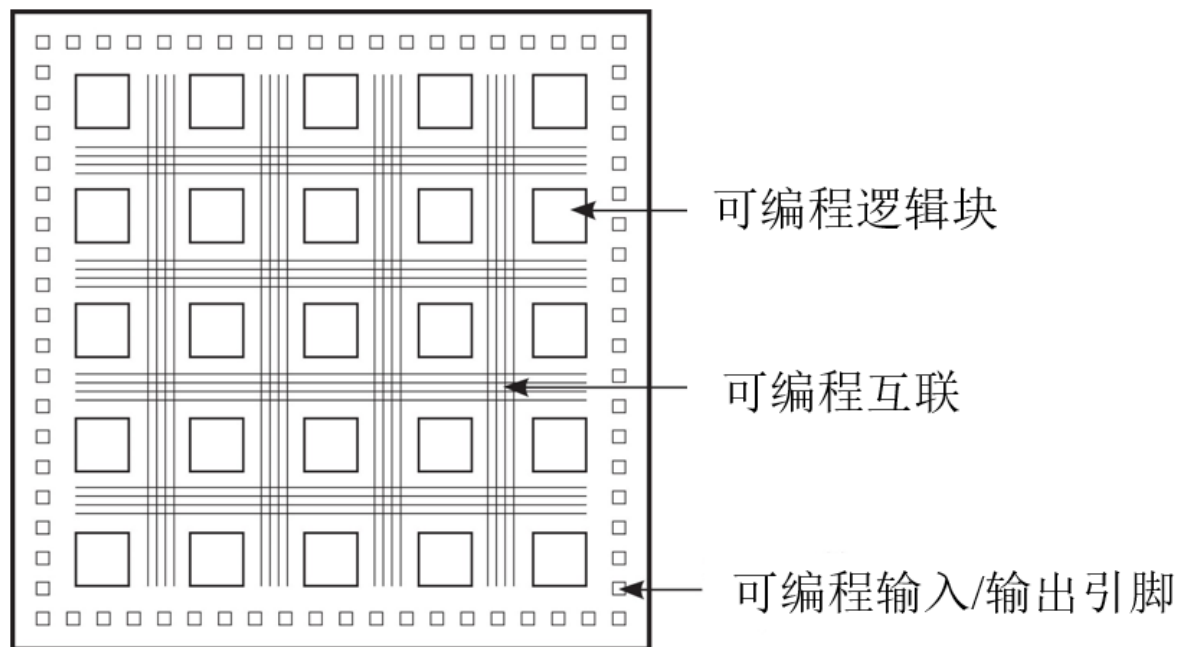
■ 可编程逻辑块

■ 例：用多路复用器实现布尔函数



1.4 可编程逻辑器件

- 现场可编程门阵列（FPGA）
 - 可编程逻辑块
 - 逻辑块之间的连接
 - 可编程的输入/输出（I/O）引脚



提纲

- 1 数字硬件实现
- 2 寄存器及传输**
- 3 多寄存器传输
- 4 寄存器传输控制
- 5 小结

2.1 寄存器与加载使能

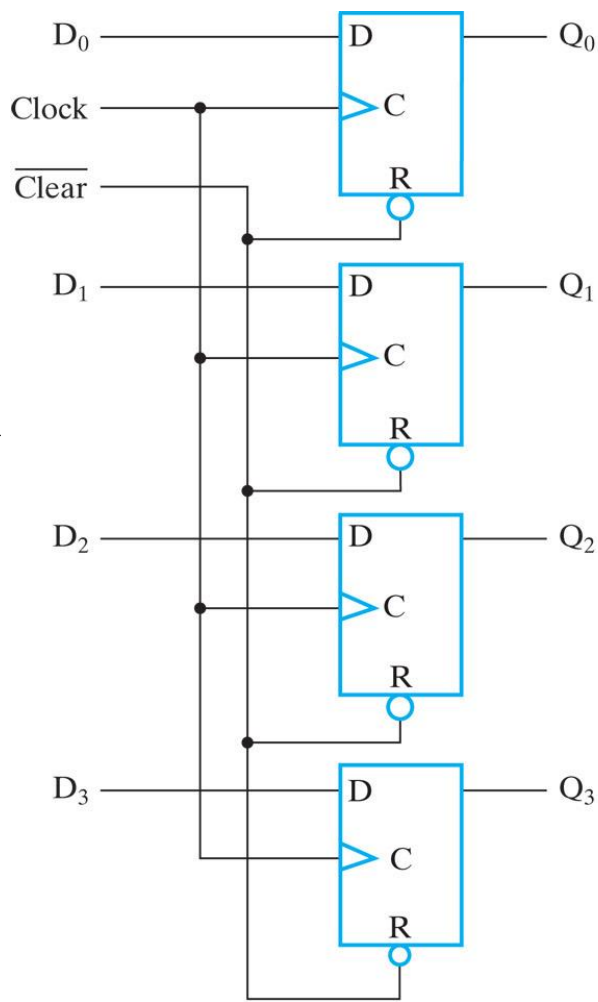
■ 寄存器的概念

- 寄存器是一组用于完成特定数据处理任务的触发器以及附加的组合门电路。
 - 触发器：锁存数据，一个触发器存储一位二进制
 - 门电路：数据处理
- 寄存器的主要作用是在数据处理过程中保持信息。对于通用计算机，用于临时保存没有保存在内部存储器中的数据。

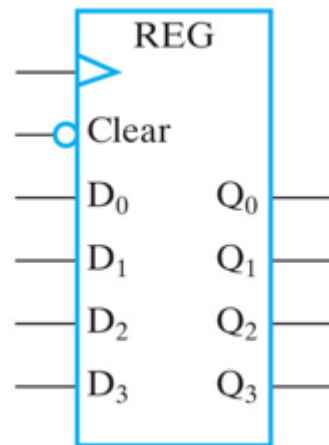
2.1 寄存器与加载使能

■ 例：4位寄存器

逻辑图



符号

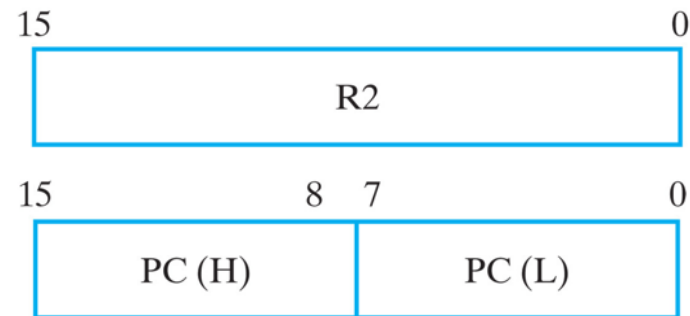
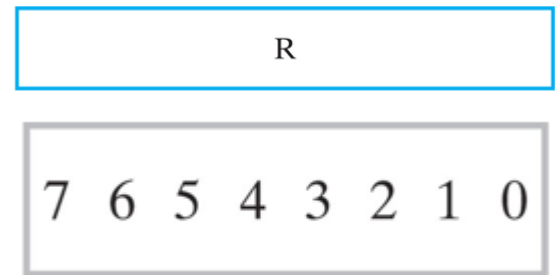


2.1 寄存器与加载使能

- 寄存器的符号表示
 - 地址寄存器（AR）
 - 保存存储单元地址
 - 程序计数器（PC）
 - 存放下一条指令所在单元的地址
 - 指令寄存器（IR）
 - 临时放置从内存里面取得的程序指令
 - 其他：寄存器2（R2）

2.1 寄存器与加载使能

- n 位寄存器中的单个触发器通常从0到 $n-1$ 编号
 - 小端格式：最低位0位在最右端
 - 大端格式：最低位0位在最左端
 - 例如16位程序计数器， $PC(L)$ 或 $PC(7:0)$ 表示低字节 $PC(H)$ 或 $PC(15:8)$ 表示高字节



2.1 寄存器与加载使能

- 加载（loading）操作
 - 将新信息传送至寄存器的过程
 - 可以用一个单独的控制信号来控制时钟脉冲对寄存器施加影响

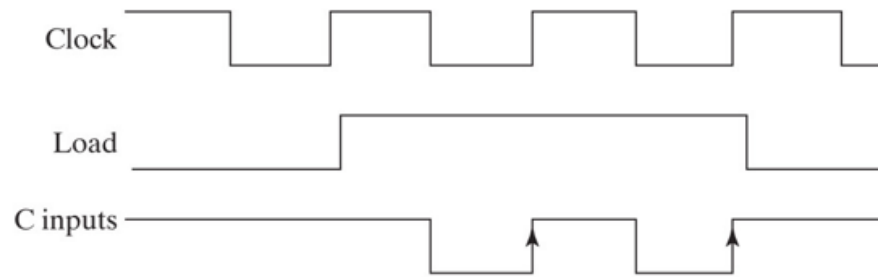


$$C\text{输入} = \overline{\text{Load}} + \text{Clock}$$

Load	C输入
1	Clock
0	1

门控时钟

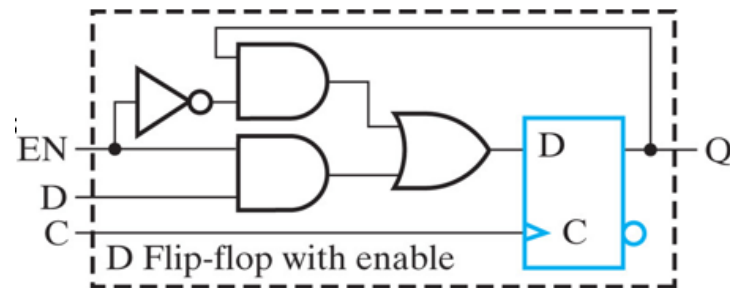
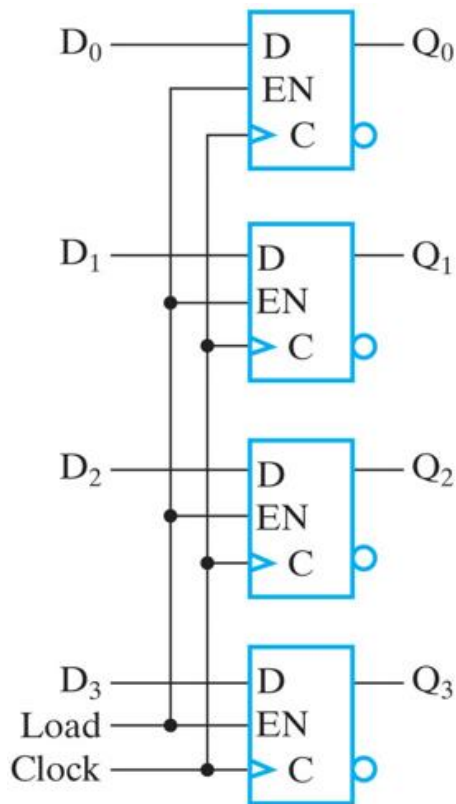
定时图



2.1 寄存器与加载使能

■ 并行加载寄存器

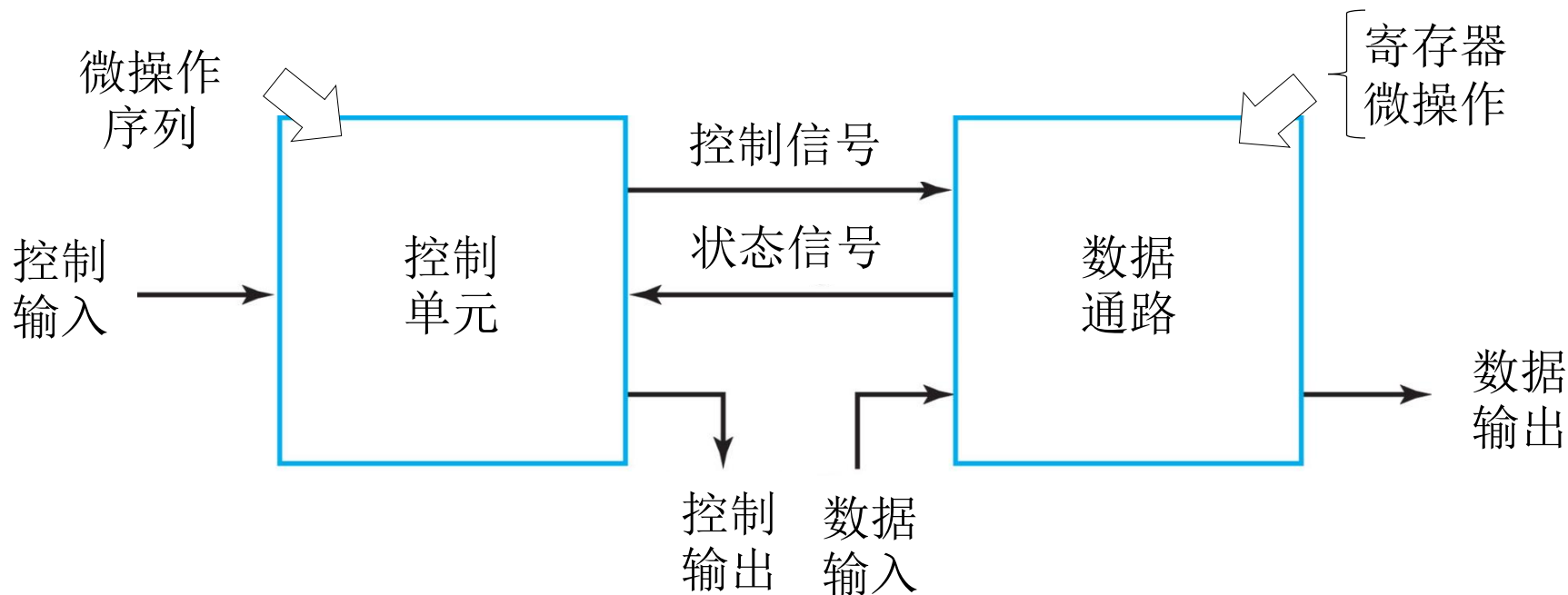
- 如果寄存器中所有位都是在公用时钟脉冲下同时加载的，称为并行加载



带并行加载的4
位寄存器

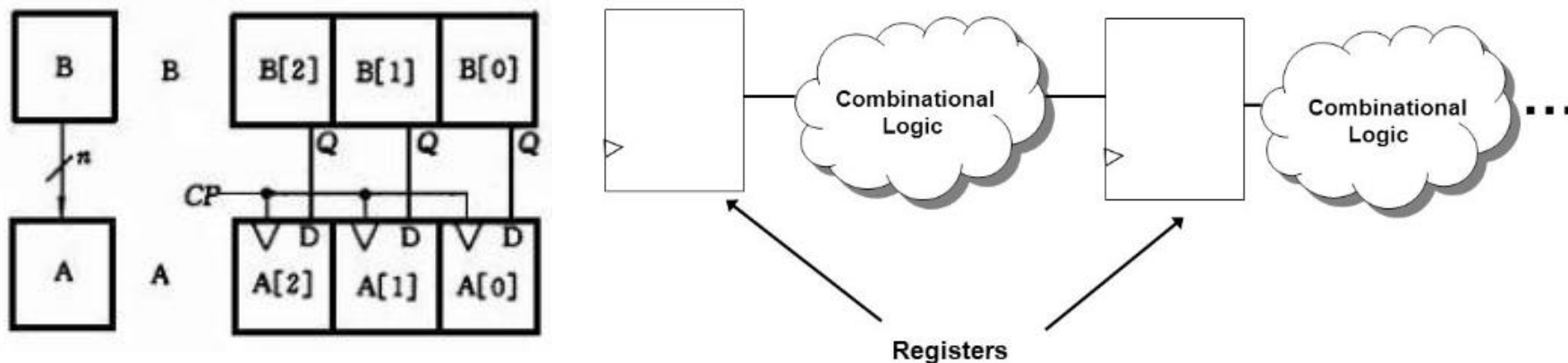
2.2 寄存器传输

- 一般数字系统由**数据通路**和**控制单元**两种类型的模块所构成
 - 数据通路：数据处理操作
 - 控制单元：操作的执行顺序



2.2 寄存器传输

- 一个寄存器可以完成一个或多个基本操作
 - 加载、计数、加法、减法、移位等
- 寄存器传输是指在寄存器之间、寄存器与存储器之间通过数据处理逻辑传输信息



2.2 寄存器传输

- 寄存器传输操作是对寄存器中存储的数据进行移动和处理，由以下三个基本方面进行描述：
 - 系统中的寄存器组
 - 对寄存器中存储数据执行的操作（微操作）
 - 系统中操作执行顺序的控制
- 符号表示 $R2 \leftarrow R1$
 - 将寄存器1（源寄存器）的内容传输到寄存器2（目的寄存器）
 - 只改变目的寄存器2的内容，不改变寄存器1

2.2 寄存器传输

- 寄存器传输语言（RTL）
 - 描述寄存器并定义对寄存器存储内容进行的各种操作

符号	含义	示例
字母（或带有数字）	一个寄存器	$AR, R2, IR$
圆括号	寄存器的一部分	$R2(1), R2(7:0)$
箭头	数据的传输	$R2 \leftarrow R1$
逗号	隔开同时进行的操作	$R2 \leftarrow R1, R1 \leftarrow R2$
方括号	定义存储器地址	$DR \leftarrow M[AR]$

2.2 寄存器传输

- 寄存器传输语言（RTL）
 - 与VHDL和Verilog的对比

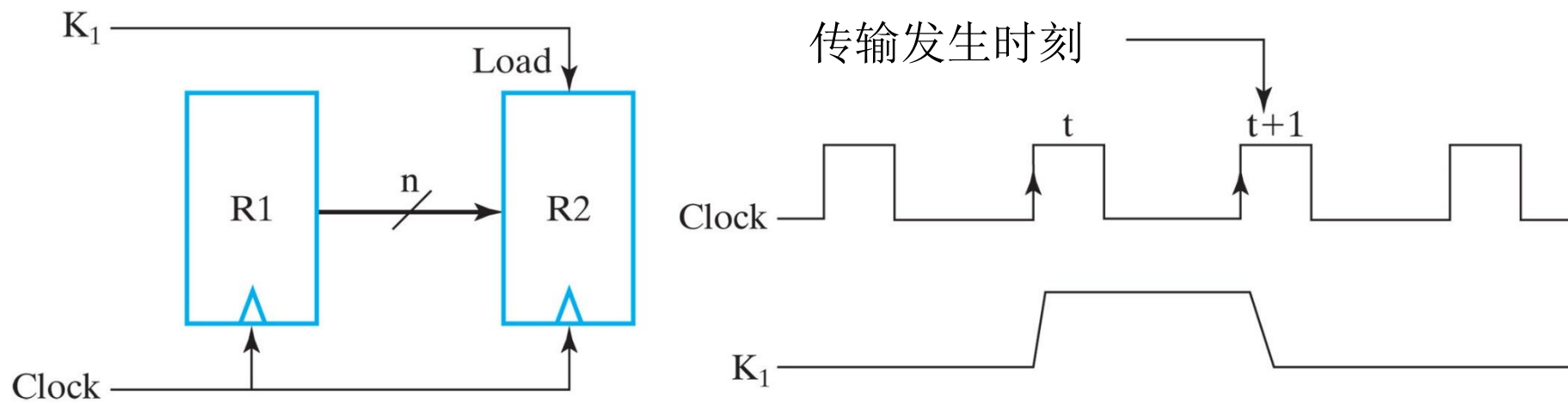
Operation	Text RTL	VHDL	Verilog
Combinational assignment	=	<= (concurrent)	assign = (nonblocking)
Register transfer	←	<= (concurrent)	<= (nonblocking)
Addition	+	+	+
Subtraction	−	−	−
Bitwise AND	∧	and	&
Bitwise OR	∨	or	
Bitwise XOR	⊕	xor	^
Bitwise NOT	− (overline)	not	~
Shift left (logical)	Sl	sll	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

2.2 寄存器传输

- 每一条寄存器传输语句都预先假定存在可以执行传输操作的逻辑和硬件结构

if ($K_1 = 1$) then ($R2 \leftarrow R1$)

简写为 $K_1 : R2 \leftarrow R1$



2.3 微操作

■ 传输微操作

- 将二进制数据从一个寄存器传输到另一个寄存器

} 不改变二进制的值

■ 算术微操作

- 对寄存器中的数据进行算术操作

■ 逻辑微操作

- 对寄存器中的数据进行位操作

} 产生新的二进制的值

■ 移位微操作

- 对寄存器中的数据进行移位

2.3 微操作

■ 传输微操作

- 将二进制数据从一个寄存器（源寄存器）传输到另一个寄存器（目的寄存器）
- 不改变二进制数据的值

$$R2 \leftarrow R1$$

2.3 微操作

■ 算术微操作

- 加法、减法、递增、递减、求补等基本算术运算

符号名称	描述
$R0 \leftarrow R1 + R2$	寄存器 $R1$ 的值与寄存器 $R2$ 的值相加，结果传输给 $R0$
$R2 \leftarrow \overline{R2}$	对寄存器 $R2$ 的值求反码
$R2 \leftarrow \overline{R2} + 1$	对寄存器 $R2$ 的值求补码
$R0 \leftarrow R1 + \overline{R2} + 1$	寄存器 $R1$ 的值加上寄存器 $R2$ 值的补码，结果传输给 $R0$
$R1 \leftarrow R1 + 1$	寄存器 $R1$ 的值加1
$R1 \leftarrow R1 - 1$	寄存器 $R1$ 的值减1

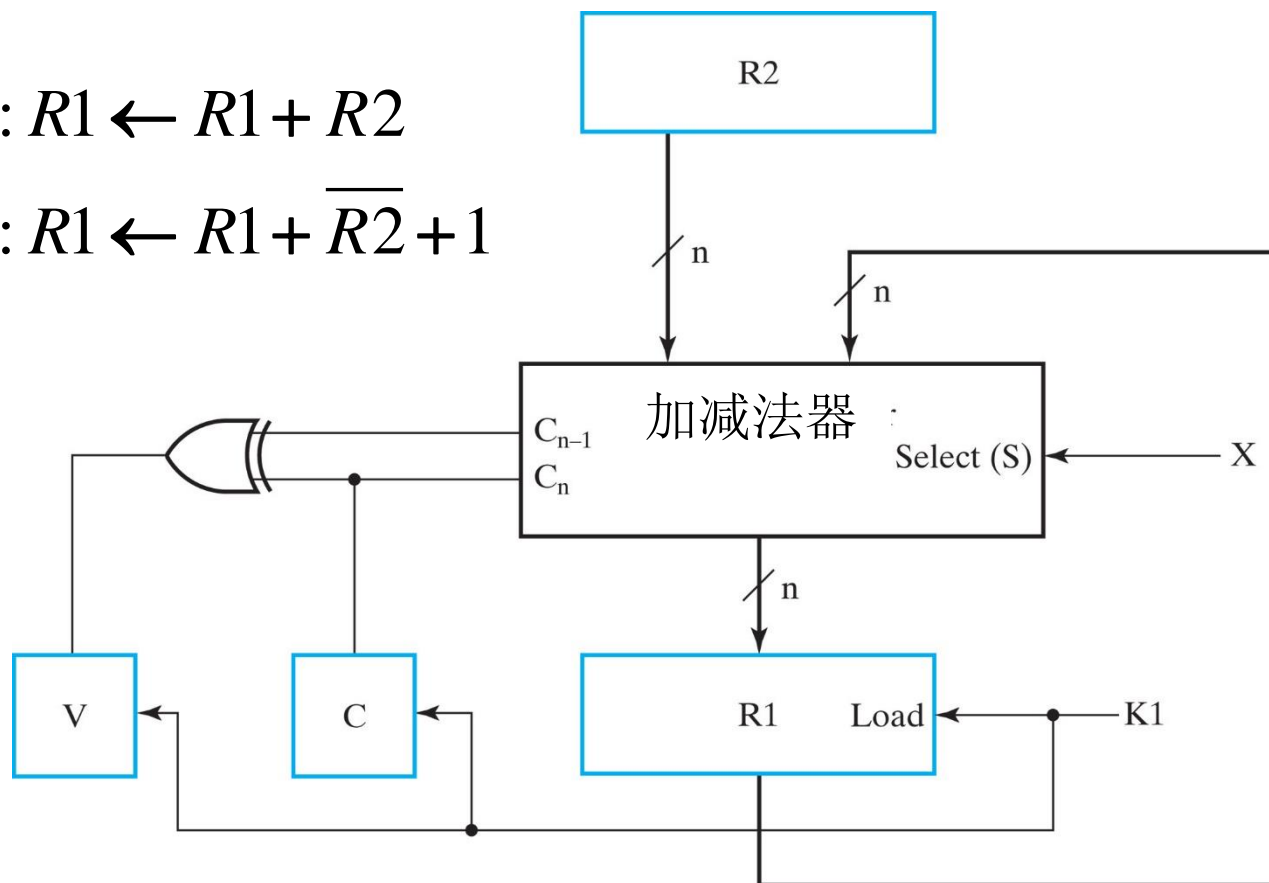
2.3 微操作

■ 算术微操作

■ 加减法器的逻辑和硬件结构

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

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



2.3 微操作

■ 逻辑微操作

■ 对寄存器数据的按位逻辑运算

符号名称	描述
$R0 \leftarrow \overline{R1}$	逻辑按位取反
$R0 \leftarrow R1 \wedge R2$	逻辑按位与
$R0 \leftarrow R1 \vee R2$	逻辑按位或
$R0 \leftarrow R1 \oplus R2$	逻辑按位异或

$(K_1 + K_2): R1 \leftarrow R2 + R3, R4 \leftarrow R5 \vee R6$

或操作 加法微操作 逻辑或微操作

2.3 微操作

■ 逻辑微操作

■ 掩蔽清除 (masking out)

10101101	10101011	$R1$	(数据)
00000000	11111111	$R2$	(掩码)
00000000	10101011	$R1 \leftarrow R1 \wedge R2$	

2.3 微操作

■ 移位微操作

- 源寄存器的内容进行向右或向左横向移动
 - 左移：向最高位移动
 - 右移：向最低位移动

示例			
类型	符号名称	源寄存器 $R2$	移位后：目的寄存器
Shift left	$R1 \leftarrow sl\ R2$	10011110	00111100
Shift right	$R1 \leftarrow sr\ R2$	11100101	01110010

2.3 微操作

- 移位微操作

- 移入位

- 左移微操作的寄存器最右端的位

- 右移微操作的寄存器最左端的位

- 移出位

- 左移微操作的源寄存器最左端的位

- 右移微操作的源寄存器最右端的位

源寄存器R2

移位后：目的寄存器

10011110

00111100

2.4 微操作实现

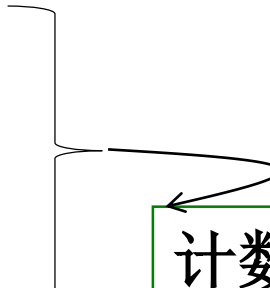
■ 寄存器使用方式

- 专用逻辑：由一个源寄存器实现的组合逻辑，从而可以把微操作当作寄存器的一部分
- 共享逻辑：针对目的寄存器组来实现微操作的组合逻辑，实现多个目的寄存器的共享

2.4 微操作实现

■ 寄存器微操作

- 基于多路复用器的传输
- 移位寄存器
- 行波计数器
- 同步二进制计数器
- 其他类型计数器
 - BCD码计数器
 - 任意计数序列



计数器：在时钟脉冲的激励下，能遍历预先规定好的状态序列的一种寄存器

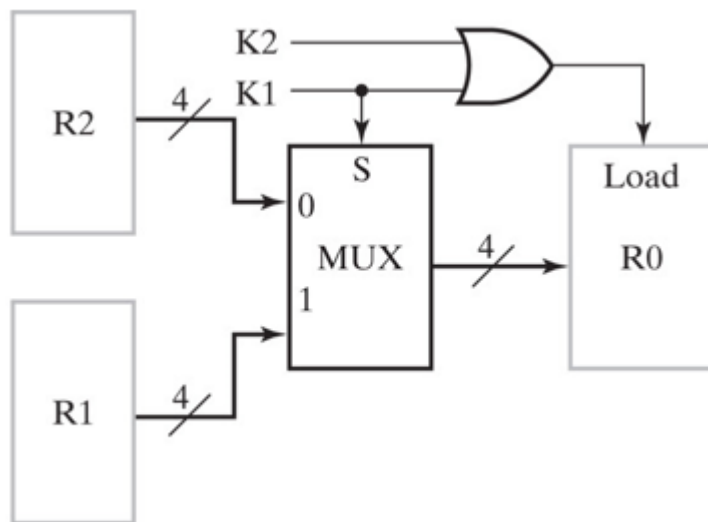
2.4 微操作实现

■ 基于多路复用器的传输

- 利用同一寄存器在不同时刻接收多个不同源的数据

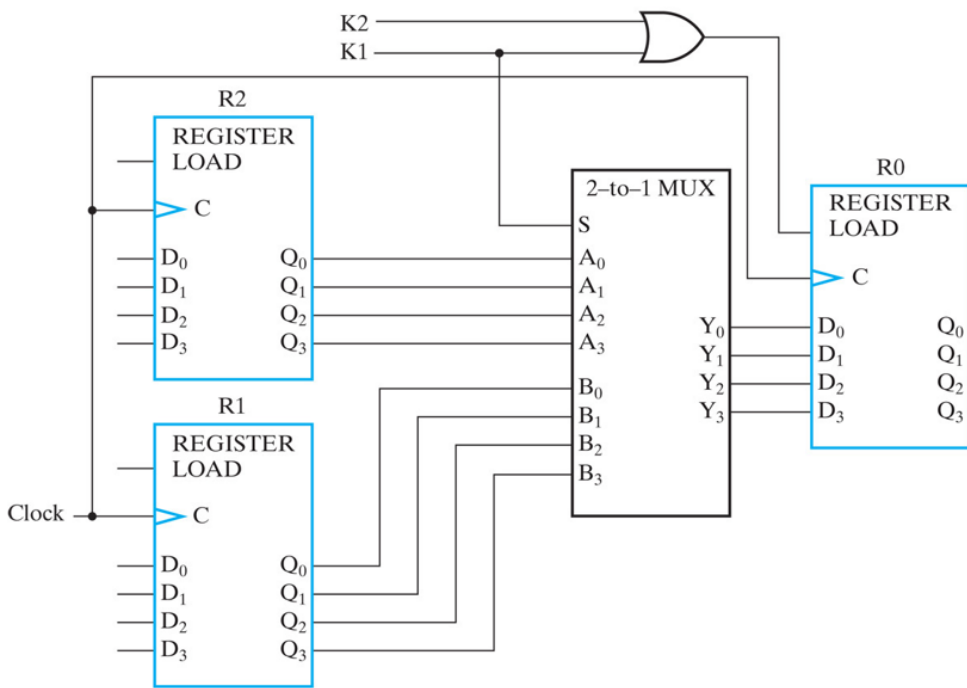
if ($K_1 = 1$) then ($R0 \leftarrow R1$) else if ($K_2 = 1$) then ($R0 \leftarrow R2$)

$\hookrightarrow K_1 : R0 \leftarrow R1 \quad \bar{K}_1 K_2 : R0 \leftarrow R2$

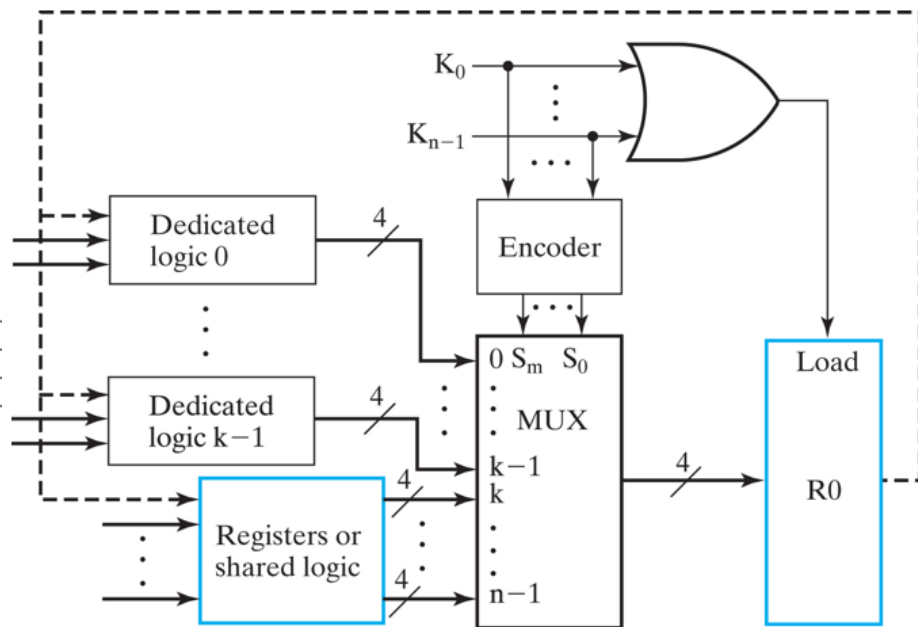


2.4 微操作实现

■ 基于多路复用器的传输



4位寄存器电路框图

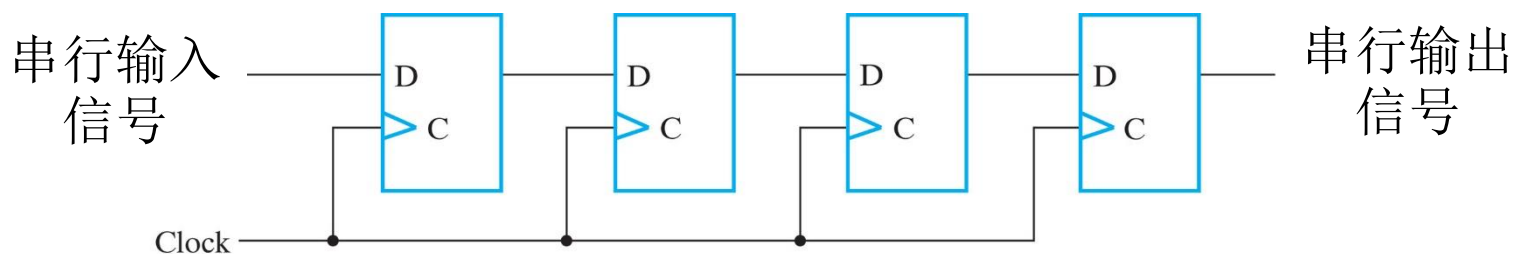


具有 n 个源的多路复用器

2.4 微操作实现

■ 移位寄存器

- 具有单向或双向移动存储数据功能的寄存器
- 每一个触发器的输出连接下一个触发器的输入，使用相同时钟脉冲输入触发移位操作



(a) Logic diagram



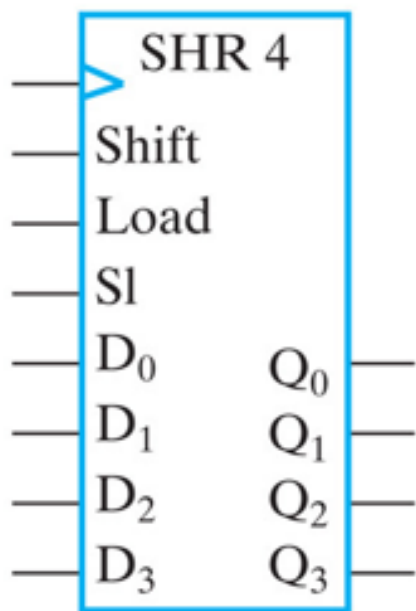
(b) Symbol

2.4 微操作实现

■ 移位寄存器

■ 具有并行加载功能的移位寄存器

- 通过使移位寄存器的所有触发器输出可访问，能够将串行移位操作进入的信息从触发器的输出端并行读出



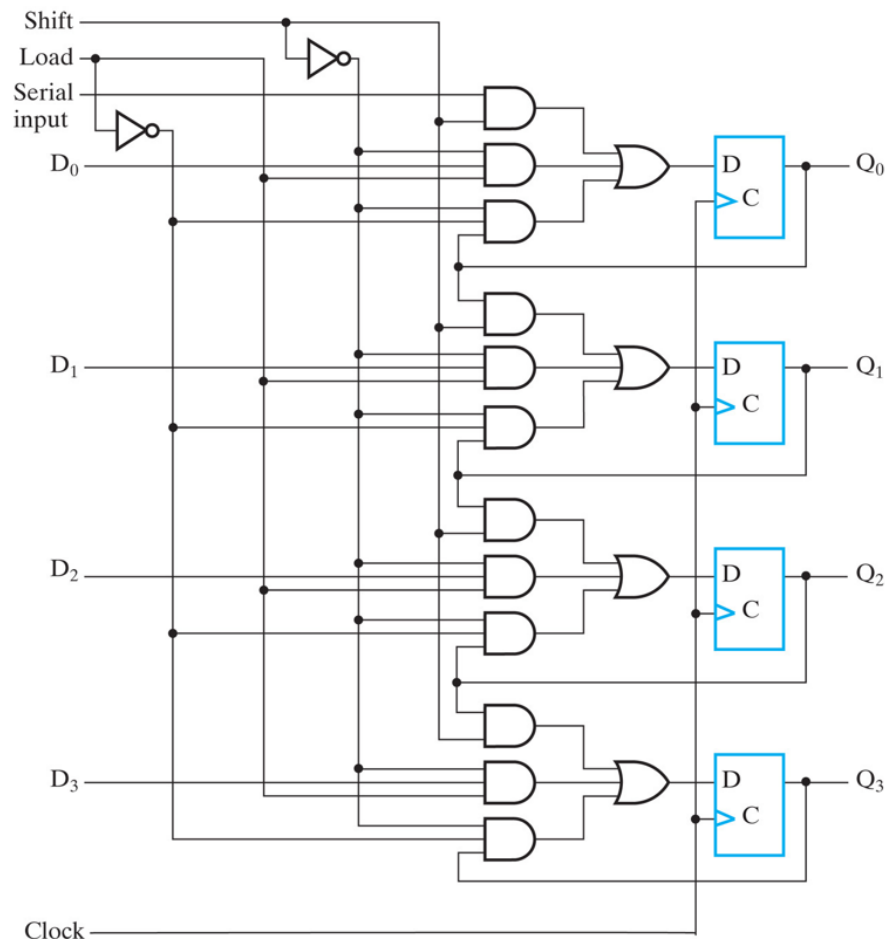
$$\text{Shift} : Q \leftarrow s1Q$$
$$\text{Shift} \cdot \text{Load} : Q \leftarrow D$$

Shift	Load	操作
0	0	没有变化
0	1	加载并行数据
1	×	左移

2.4 微操作实现

■ 移位寄存器

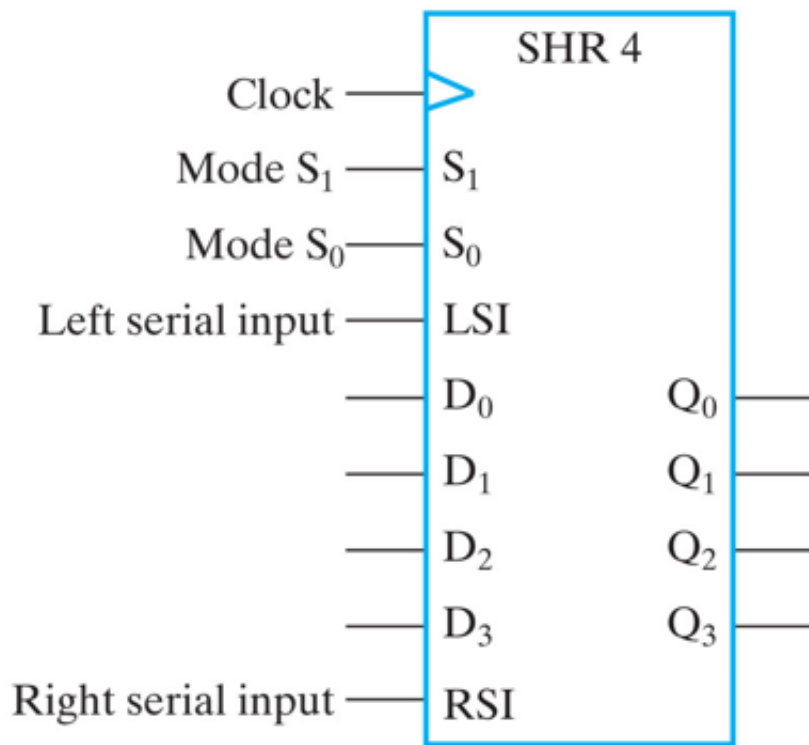
■ 具有并行加载功能的移位寄存器



2.4 微操作实现

■ 移位寄存器

■ 双向移位寄存器



$$\bar{S}_1 \cdot S_0 : Q \leftarrow \text{sl}Q$$

$$S_1 \cdot \bar{S}_0 : Q \leftarrow \text{sr}Q$$

$$S_1 \cdot S_0 : Q \leftarrow D$$

S1	S0	寄存器操作
0	0	保持不变
0	1	向左移位
1	0	向右移位
1	1	并行加载

2.4 微操作实现

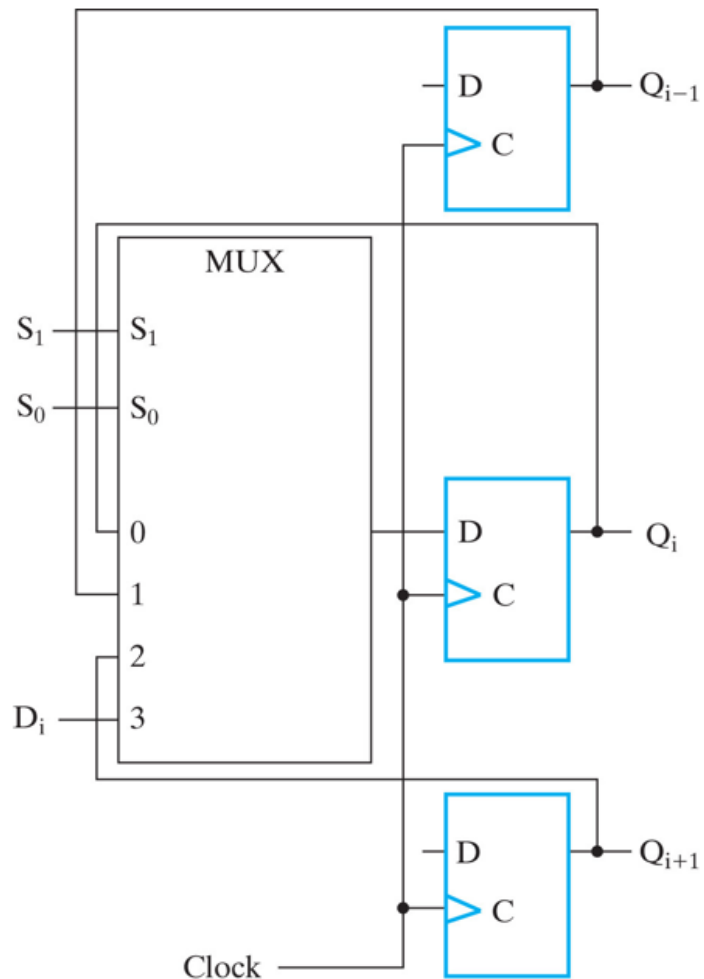
■ 移位寄存器

■ 双向移位寄存器

$$\bar{S}_1 \cdot S_0 : Q \leftarrow \mathfrak{sl} Q$$

$$S_1 \cdot \bar{S}_0 : Q \leftarrow \text{sr}Q$$

$$S_1 \cdot S_0 : Q \leftarrow D$$



2.4 微操作实现

■ 计数器（counter）

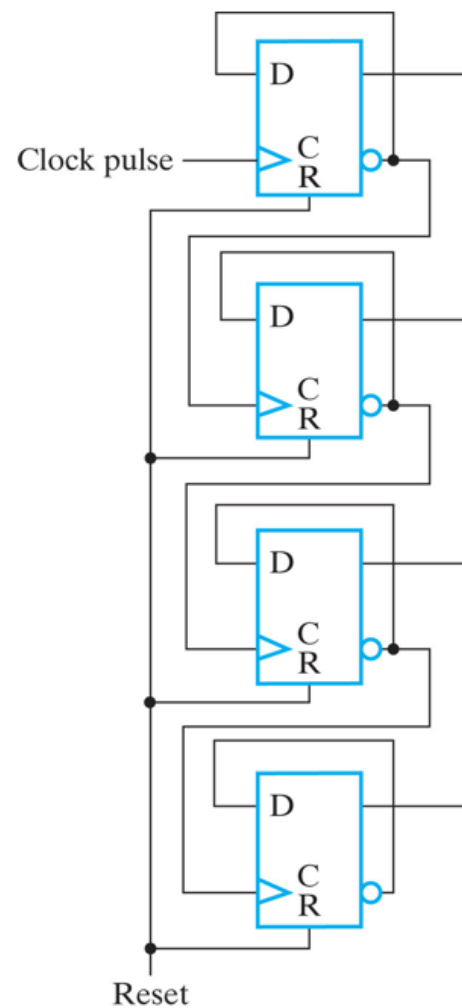
- 能够在输入脉冲序列的激励下遍历指定状态序列的寄存器。 n 位二进制计数器的计数范围为 $0\sim 2^n-1$
- 分为两种类型
 - 行波计数器：加载到触发器输入端的值不是公用时钟脉冲，而是其他触发器的输出信号
 - 同步计数器：所有触发器的输入端都是加载的公用时钟脉冲

2.4 微操作实现

■ 计数器（counter）

■ 例：4位行波计数器

向上计数序列				向下计数序列			
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

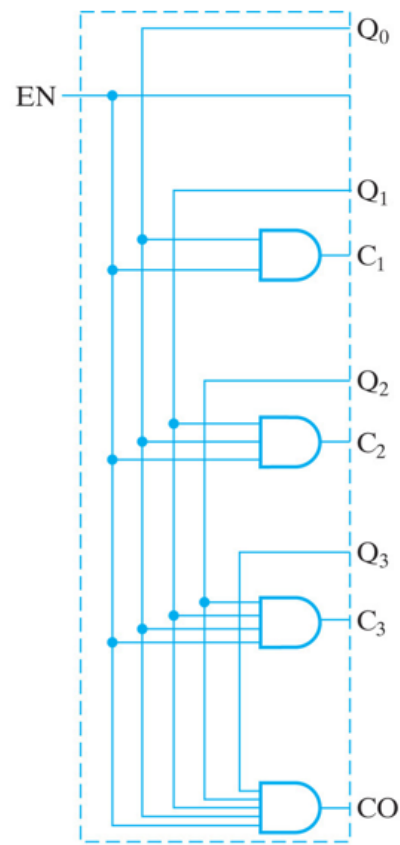
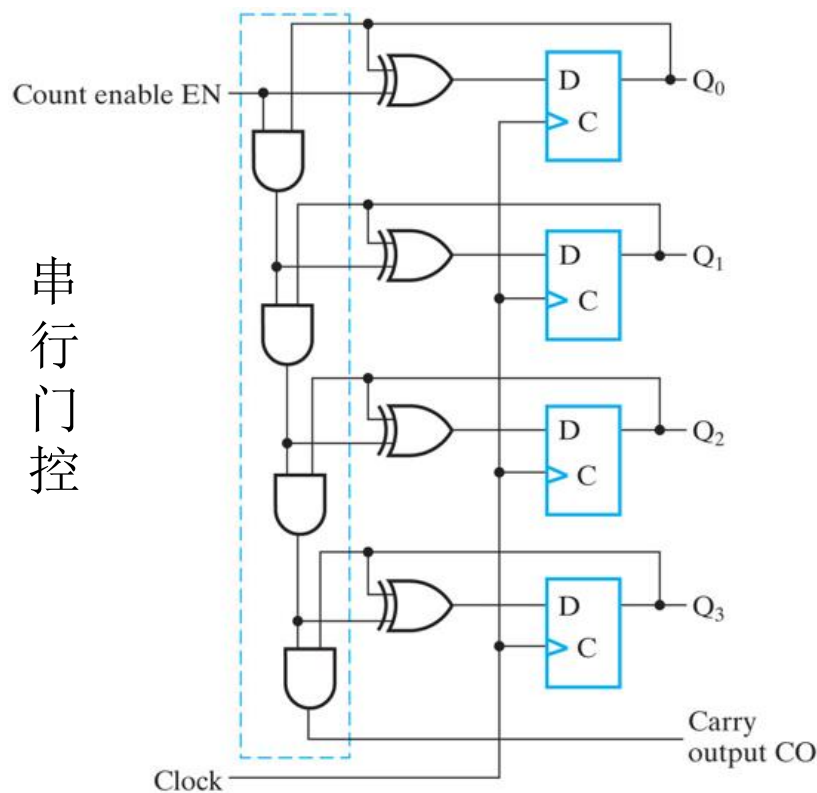


2.4 微操作实现

■ 计数器（counter）

■ 同步二进制计数器

① 串行和并行计数器

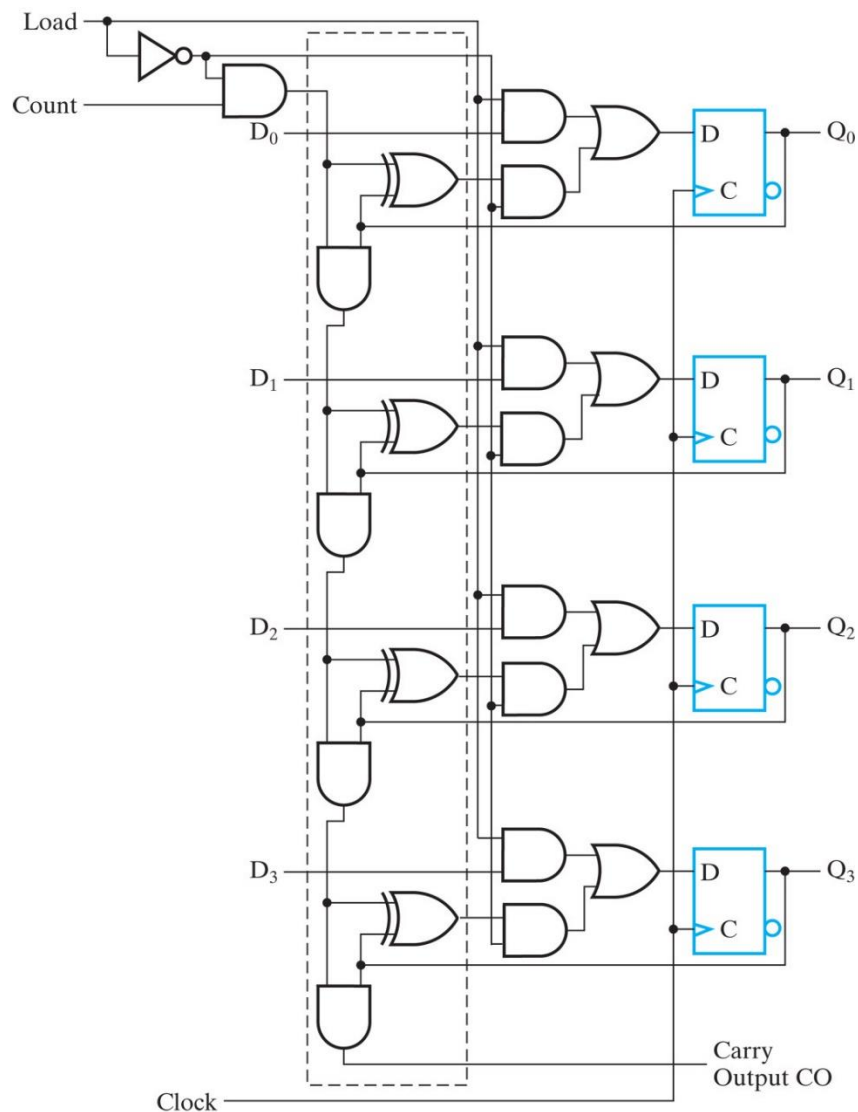


2.4 微操作实现

■ 计数器（counter）

■ 同步二进制计数器

- ② 双向二进制计数器
- ③ 具有并行加载功能的二进制计数器



2.4 微操作实现

■ 计数器（counter）

- 任意计数序列：能够反复遍历6个状态的计数器

$$DA = A \oplus B \quad DB = C \quad DC = \bar{B}\bar{C}$$

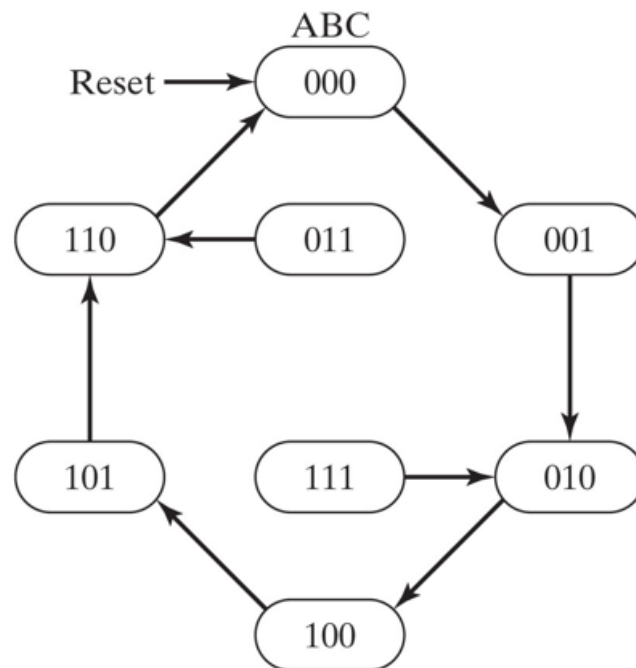
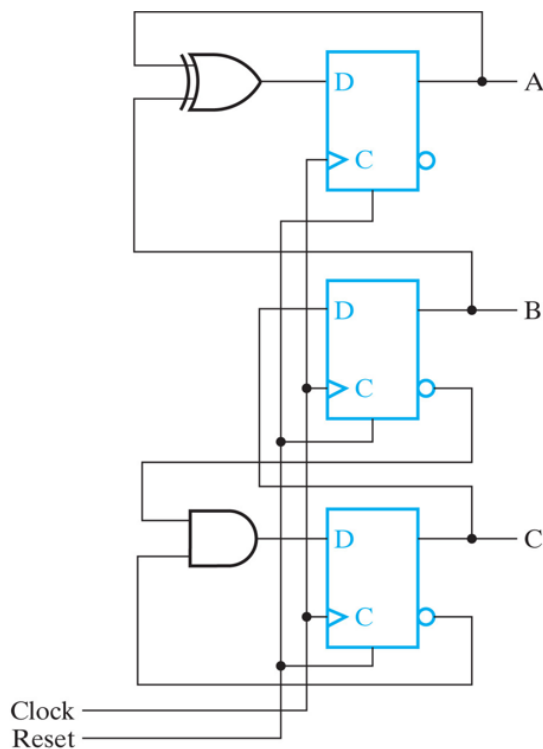
当前状态			下一状态		
A	B	C	DA = A(t + 1)	DB = B(t + 1)	DC = C(t + 1)
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0

2.4 微操作实现

■ 计数器（counter）

- 任意计数序列：能够反复遍历6个状态的计数器

$$DA = A \oplus B \quad DB = C \quad DC = \bar{B}\bar{C}$$



2.5 寄存器单元设计

- 将一位的迭代组合电路单元与一个触发器连接起来就构成了一个具有两个状态的时序电路，称为寄存器单元
- 将一个寄存器单元复制 n 份组合在一起，构成具有某种或者几种相关操作功能的 n 位寄存器

2.5 寄存器单元设计

- 例：寄存器A具有如下寄存器传输功能

$$\text{AND} : A \leftarrow A \wedge B$$

$$\text{EXOR} : A \leftarrow A \oplus B$$

$$\text{OR} : A \leftarrow A \vee B$$

当前状态

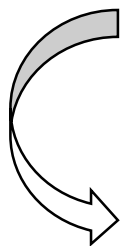
下一状态A(t+1)

	(AND = 0) (EXOR = 0) (OR = 1) (OR = 1) (EXOR = 1) (EXOR = 1) (AND = 1) (AND = 1) (OR = 0) (B = 0) (B = 1) (B = 0) (B = 1) (B = 0) (B = 1)						
0	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1

2.5 寄存器单元设计

- 例：寄存器A具有如下寄存器传输功能

$$D_i = A(t+1)_i = \text{AND} \cdot A_i \cdot B_i + \text{EXOR} \cdot (A_i \bar{B}_i + \bar{A}_i B_i) \\ + \text{OR} \cdot (A_i + B_i) + \overline{\text{AND}} \cdot \overline{\text{EXOR}} \cdot \overline{\text{OR}} \cdot A_i$$



$$C_1 = \text{OR} + \text{AND} + \overline{\text{EXOR}}$$

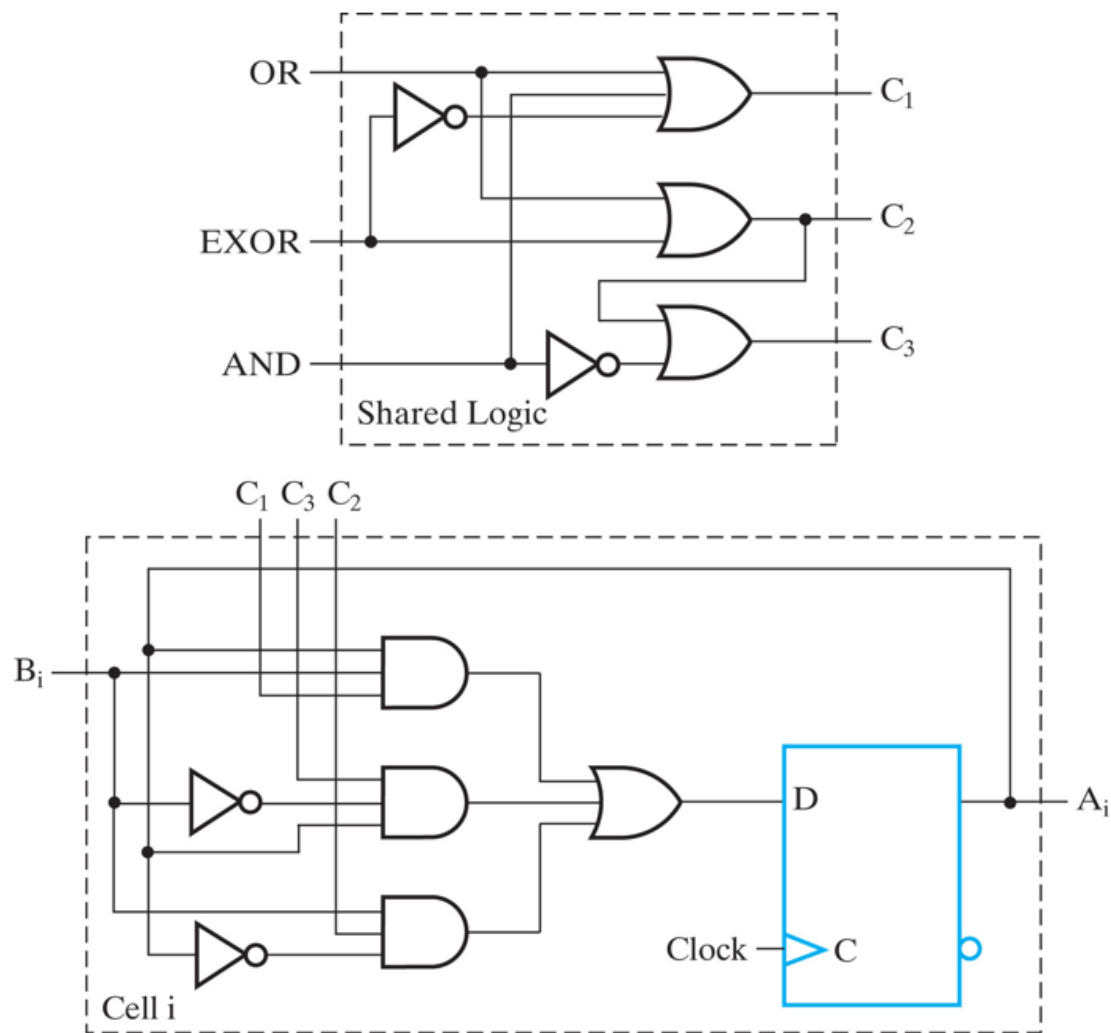
$$C_2 = \text{OR} + \text{EXOR}$$

$$C_3 = C_2 + \overline{\text{AND}}$$

$$D_i = C_1 A_i B_i + C_3 A_i \bar{B}_i + C_2 \bar{A}_i B_i$$

2.5 寄存器单元设计

- 例：寄存器A具有如下寄存器传输功能



提纲

- 1 数字硬件实现
- 2 寄存器及传输
- 3 多寄存器传输**
- 4 寄存器传输控制
- 5 小结

3.1 传输类型

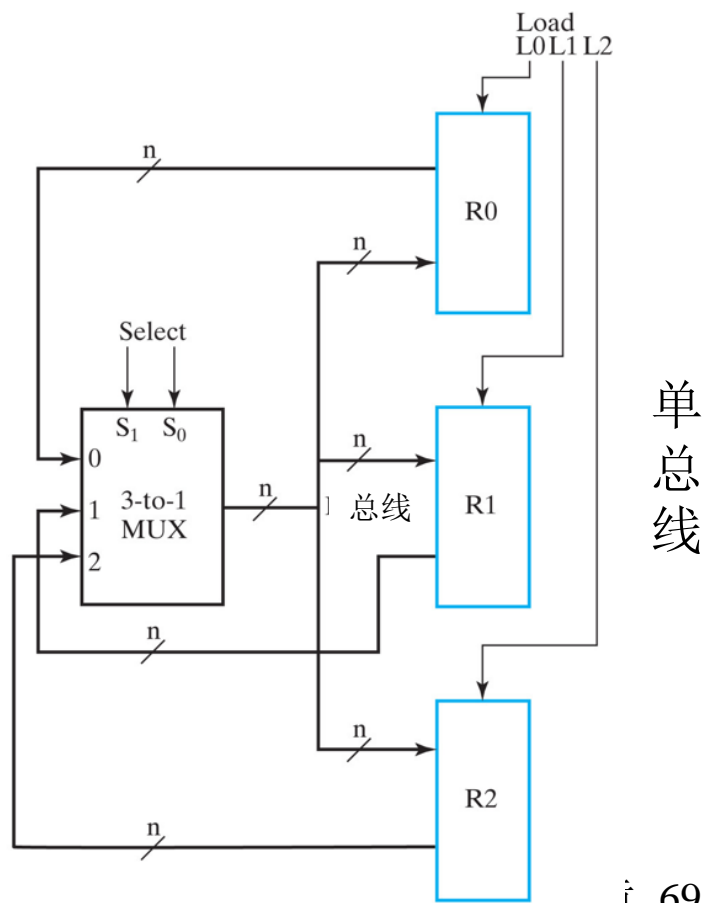
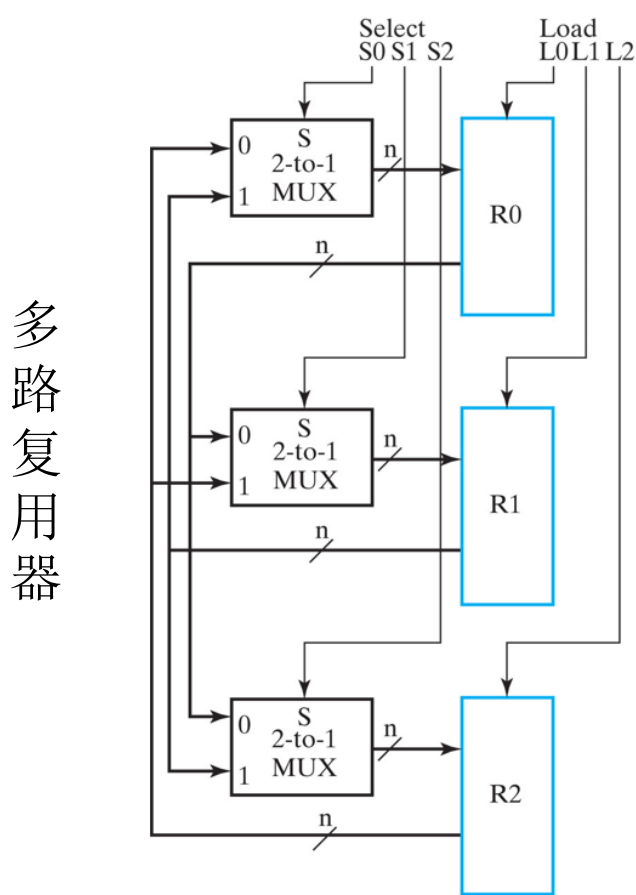
- 数字系统中多个寄存器之间传输数据时可采用**多路复用器**和**总线**两种方式
 - 多路复用器：共享逻辑实现的不同时刻寄存器接收不同来源数据
 - 优点：逻辑简单
 - 缺点：逻辑电路的大小和数量巨大

3.1 传输类型

- 数字系统中多个寄存器之间传输数据时可采用**多路复用器**和**总线**两种方式
 - 总线：一组通用连线作为共享传输的通路。通常情况下，如果一组多路复用器的输出作为公共通路为多个目的寄存器共享，那么这些输出线就称为总线。
 - 优点：电路简单
 - 缺点：逻辑复杂

3.1 传输类型

- 数字系统中多个寄存器之间传输数据时可以采用**多路复用器**和**总线**两种方式

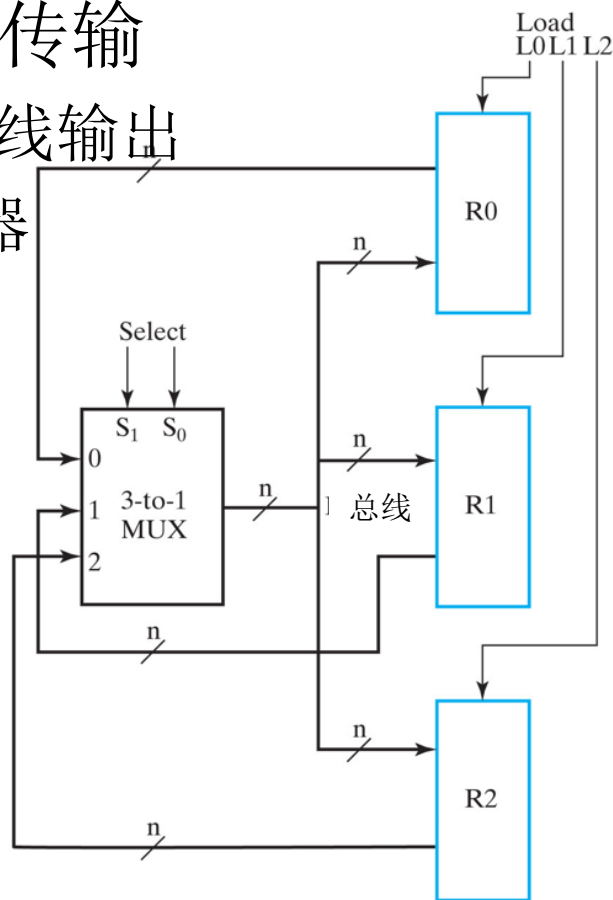


3.2 总线构建方式

① 两态总线

- 采用一对控制输入信号实现一条总线在寄存器之间进行数据传输
- Select: 哪个寄存器作为总线输出
- Load: 哪个作为目的寄存器

Register Transfer	Select		Load		
	S1	S0	L2	L1	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		



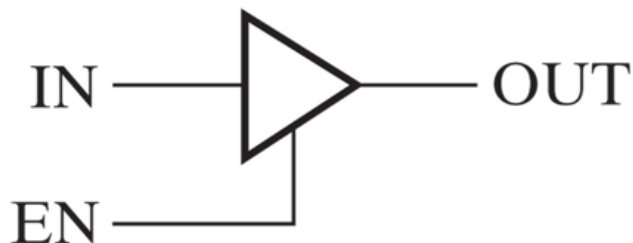
单总线

3.2 总线构建方式

② 三态总线

- 采用三态缓冲器取代多路复用器，减少连接的数量

三态缓冲器：除了逻辑0和逻辑1，还可以提供称为高阻态的第三种输出值（Hi-Z），能够作为双向输入/输出。



逻辑符号

EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

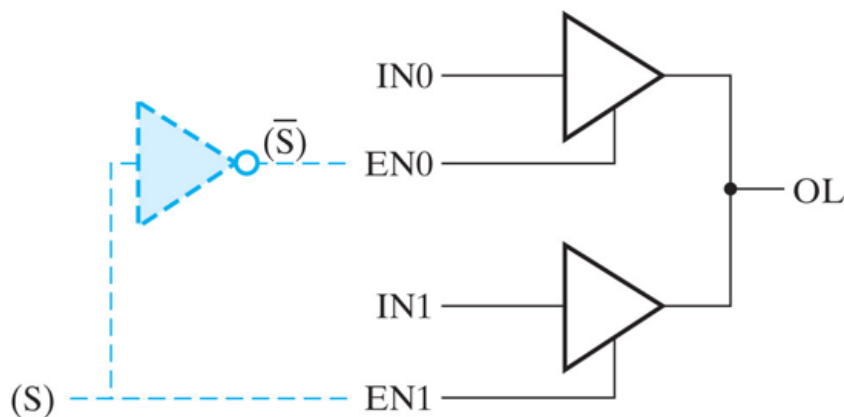
真值表

3.2 总线构建方式

② 三态总线

- 采用三态缓冲器取代多路复用器，减少连接的数量

三态缓冲器：除了逻辑0和逻辑1，还可以提供称为高阻态的第三种输出值（Hi-Z），能够作为双向输入/输出。



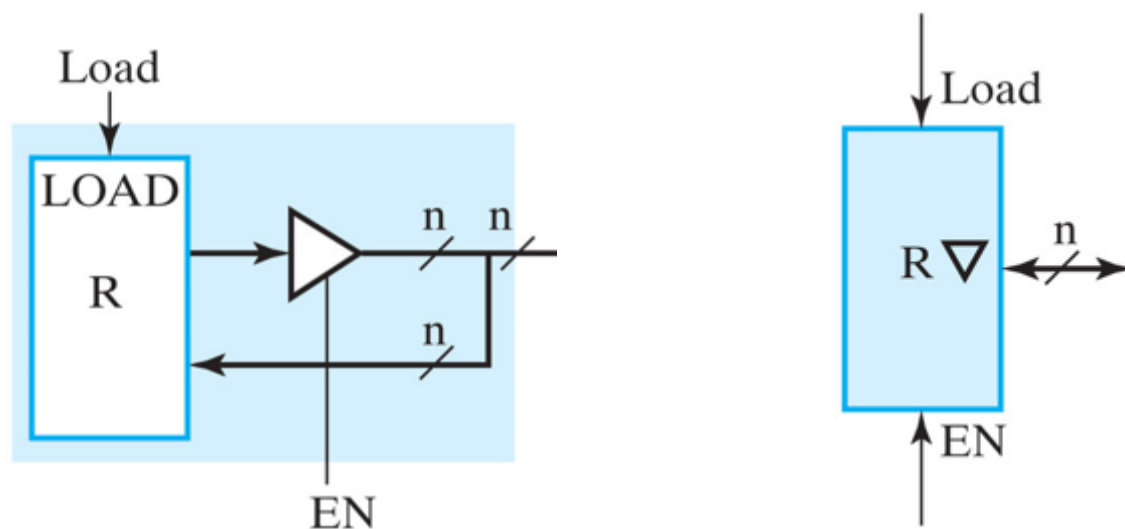
多重输出线

EN1	EN0	IN1	IN0	OL
0	0	X	X	Hi-Z
(S) 0	(S-bar) 1	X	0	0
0	1	X	1	1
1	0	0	X	0
1	0	1	X	1
1	1	0	0	0
1	1	1	1	1
1	1	0	1	0
1	1	1	0	1

3.2 总线构建方式

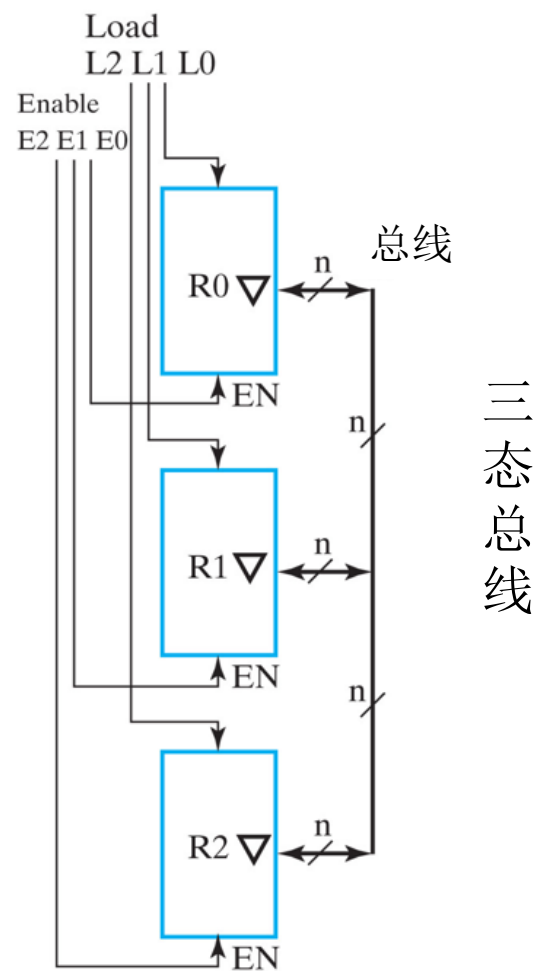
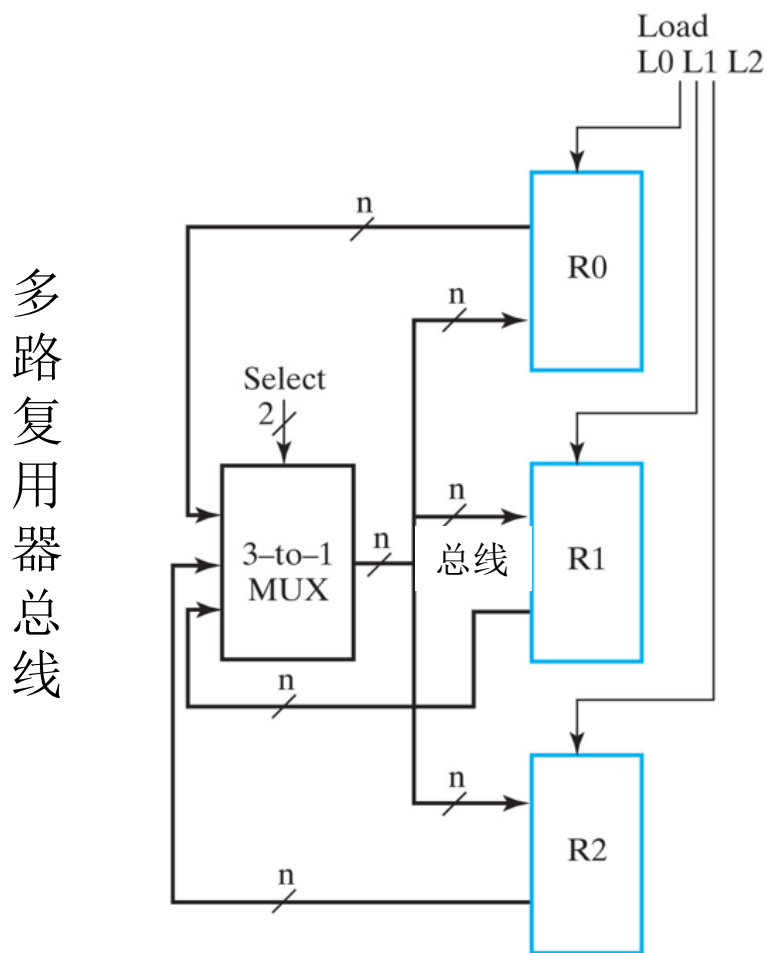
② 三态总线

- 采用三态缓冲器取代多路复用器，减少连接的数量
- 通过将多个三态缓冲器的输出连接在一起形成一位总线，其中的信号可以双向传播



3.2 总线构建方式

② 三态总线

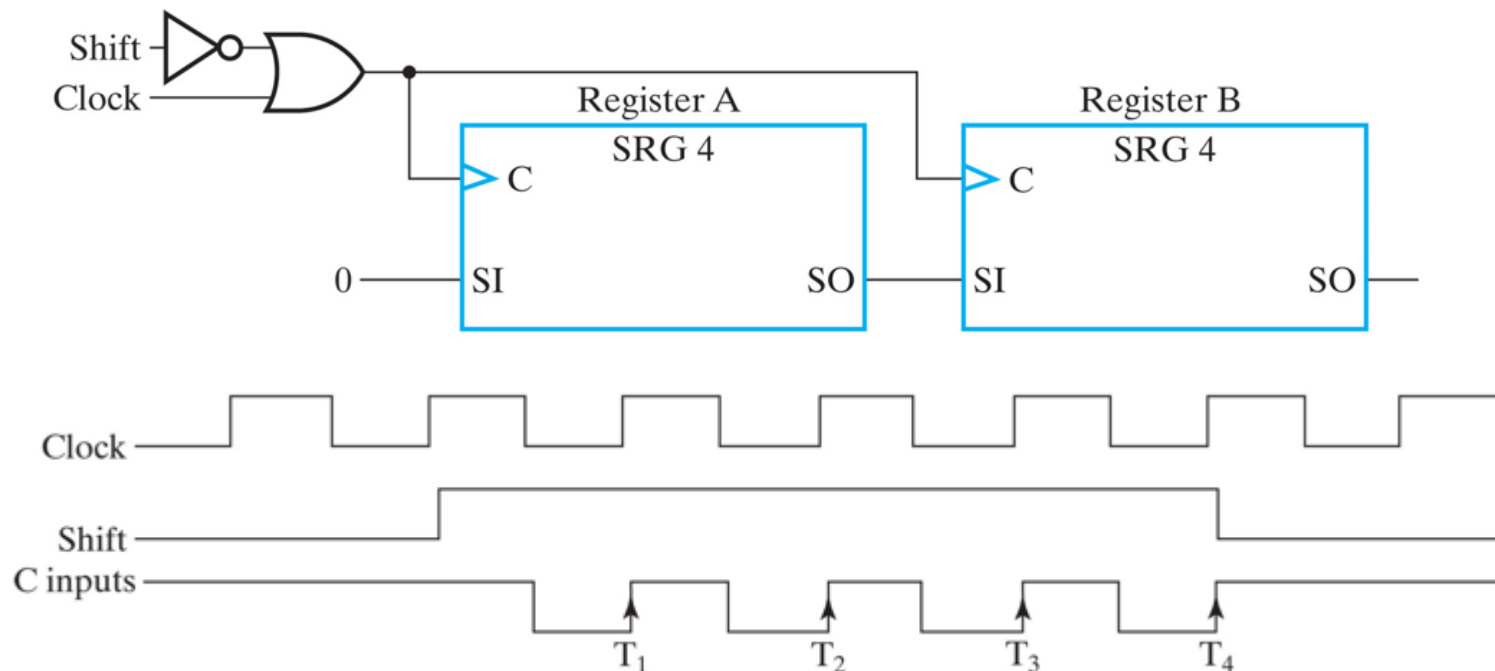


提纲

- 1 数字硬件实现
- 2 寄存器及传输
- 3 多寄存器传输
- 4 寄存器传输控制**
- 5 小结

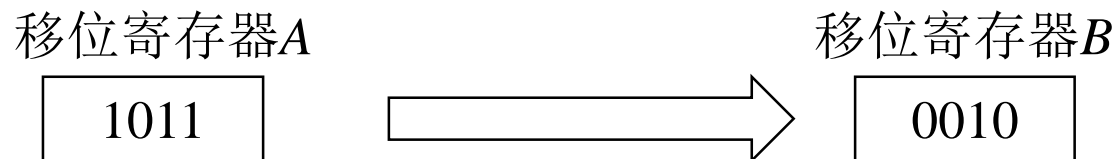
4.1 串行传输

- 并行传输：所有数据同时传输
- 串行传输：每次只传输一位数据
 - 通过移位寄存器可将寄存器A中的数据通过串行传输的方式传输至寄存器B



4.1 串行传输

■ 例



时钟脉冲	移位寄存器A				移位寄存器B			
Initial value	1	0	1	1	0	0	1	0
After T_1	0	1	0	1	1	0	0	1
After T_2	0	0	1	0	1	1	0	0
After T_3	0	0	0	1	0	1	1	0
After T_4	0	0	0	0	1	0	1	1

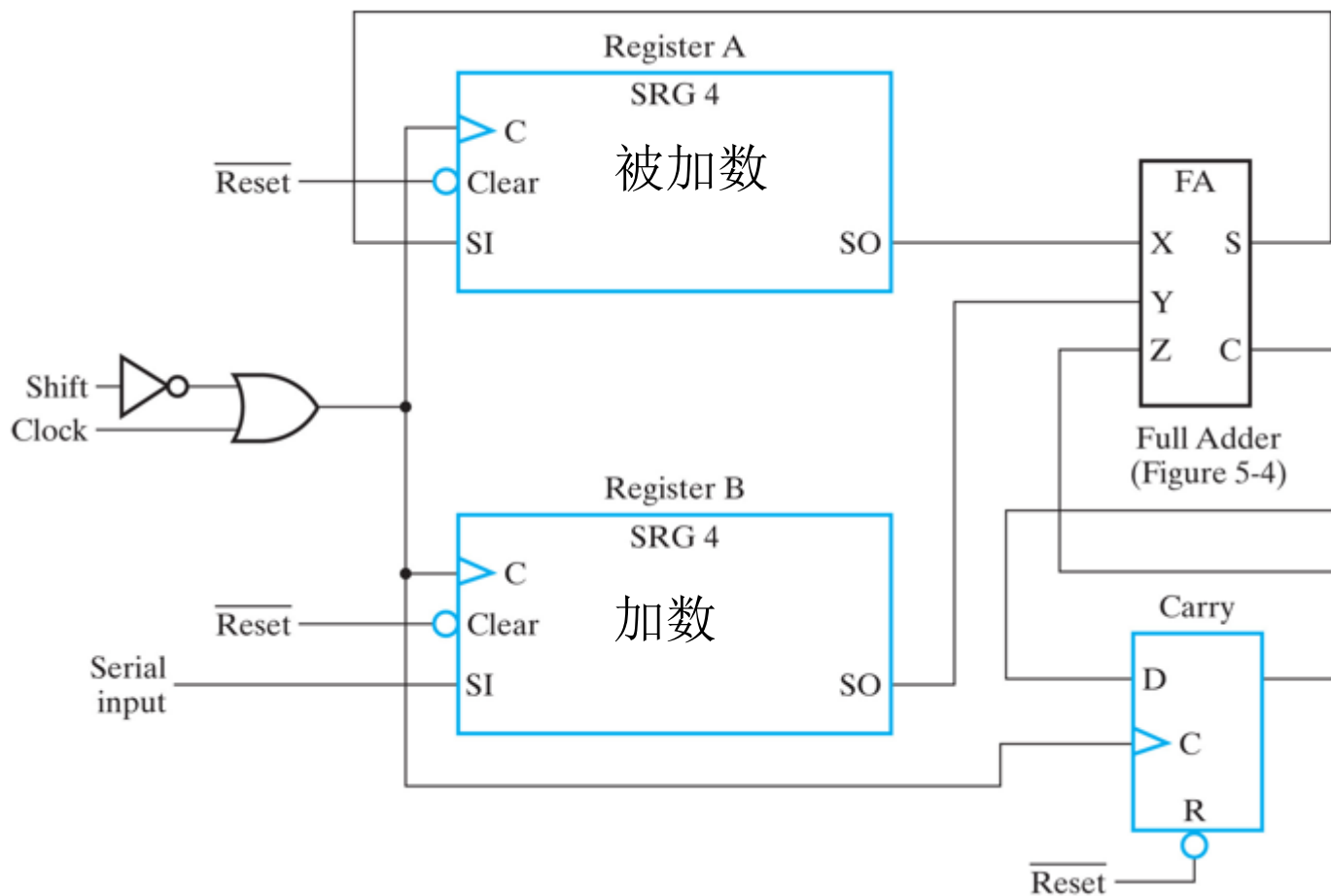
4.1 串行传输

■ 并行vs串行

- 并行模式下寄存器中的值可以任意改变，并且所有位都可以在一个时钟脉冲周期内并行传输；处理速度快
- 串行模式下寄存器只有一个串行输入端和一个串行输出端，数据一次只能传输一位；处理速度慢，但只需较少的硬件资源即可完成相应操作

4.2 串行加法

- 借助一个全加器实现加法操作



4.3 寄存器传输系统

- 控制单元用于决定微操作执行顺序的控制信号，而其本身也是一种时序电路
 - 可编程系统：处理器的部分输入形成指令序列。执行一条指令也就是激活所需要在数据通路中执行的微操作序列。
 - 不可编程系统：不依赖于指令序列，仅通过当前输入和数据通路中的状态位决定所需执行的操作

4.3 寄存器传输系统

■ 传输控制系统的一般设计过程

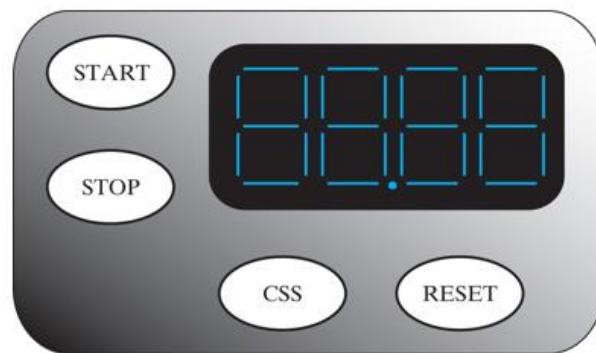
- ① 系统说明书
- ② 定义外部数据和控制信号
- ③ 画出状态机图
- ④ 定义内部控制和状态信号
- ⑤ 画出模块图
- ⑥ 设计寄存器传输逻辑
- ⑦ 设计控制单元逻辑
- ⑧ 检查系统正确性

4.3 寄存器传输系统

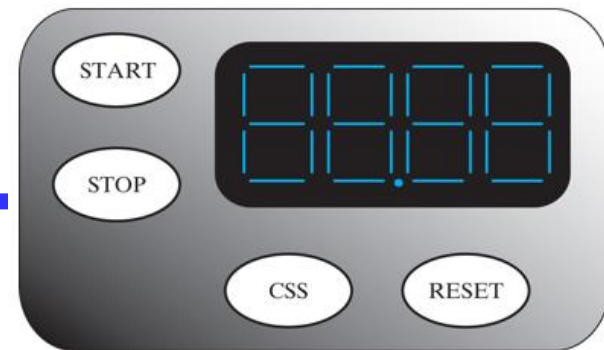
■ 例：码表

① 功能相对单一的跑表，用于短距离计时

- 计时上限99.99秒
- 将最短时间存储在寄存器中
- START：计时器清零
- STOP：停止计时并显示
- CSS：比较当前的最短计时；
将最小值记录在寄存器中；
将最小值计时显示于LCD
- RESET：寄存器值设为最大99.99



4.3 寄存器传输系统

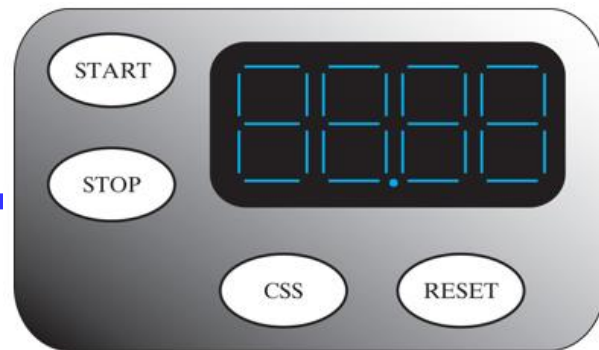


■ 例：码表

② 系统所需外部控制输入信号、外部数据输出信号、寄存器等

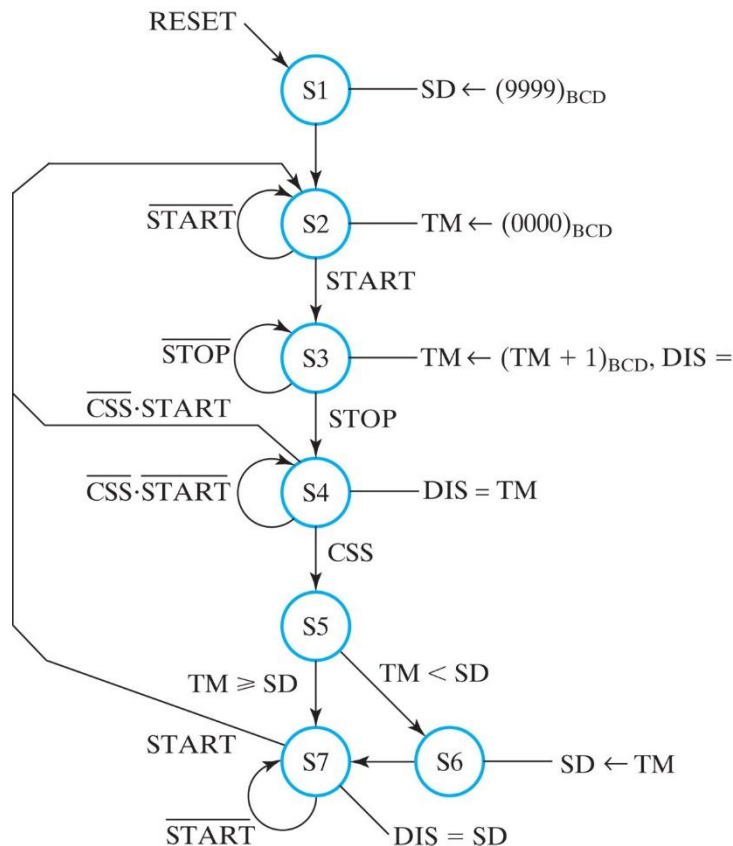
TM	符号	功能	类型
4位数字 BCD计数器	START	Initialize timer to 0 and start timer	Control input
	STOP	Stop timer and display timer	Control input
	CSS	Compare, store, and display shortest dash time	Control input
	RESET	Set shortest value to 10011001	Control input
	B ₁	Digit 1 data vector a, b, c, d, e, f, g to display	Data output vector
SD	B ₀	Digit 0 data vector a, b, c, d, e, f, g to display	Data output vector
	DP	Decimal point to display (= 1)	Data output
	B ₋₁	Digit -1 data vector a, b, c, d, e, f, g to display	Data output vector
	B ₋₂	Digit -2 data vector a, b, c, d, e, f, g to display	Data output vector
	B	The 29-bit display input vector (B ₁ , B ₀ , DP, B ₋₁ , B ₋₂)	Data output vector
	TM	4-Digit BCD counter	16-Bit register
	SD	Parallel load register	16-Bit register

4.3 寄存器传输系统



■ 例：码表

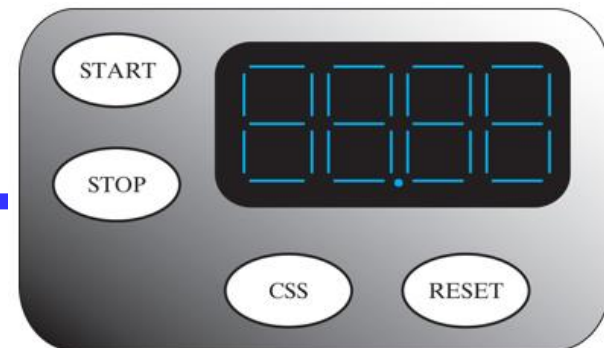
③ 系统状态机图



Moore型状态机

- 系统上电或RESET进入S1
- 状态S2时，TM同步复位为0
- START为1时，状态进入S3
- STOP为1时，状态进入S4
- 状态S5时，比较TM与SD值
- 状态S6时，赋值
- 状态S7时，显示

4.3 寄存器传输系统



■ 例：码表

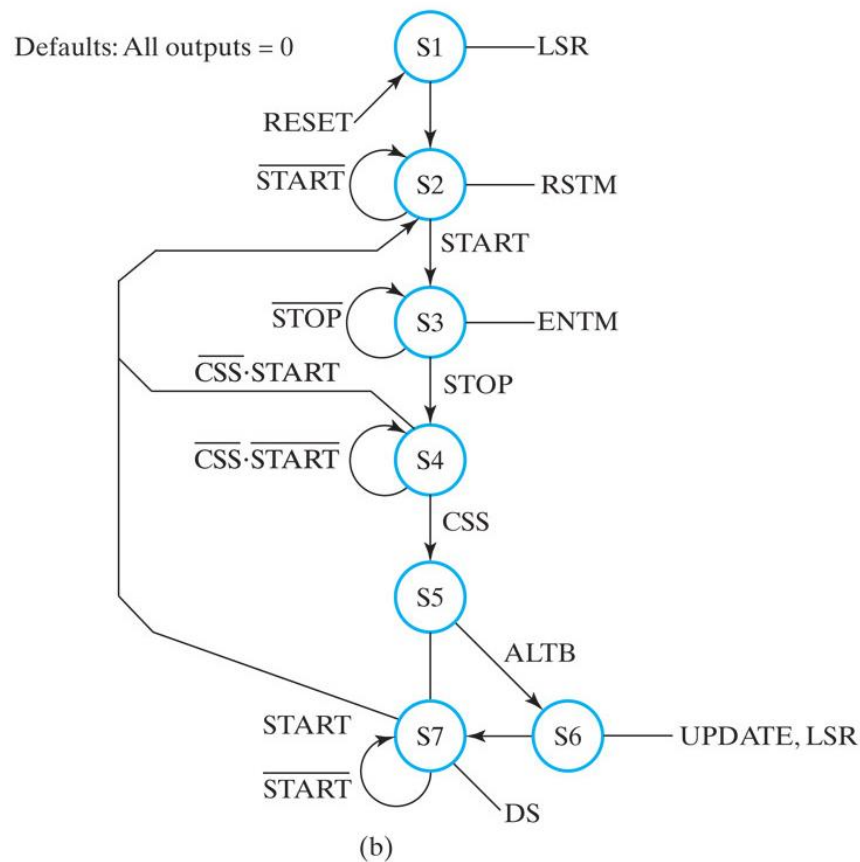
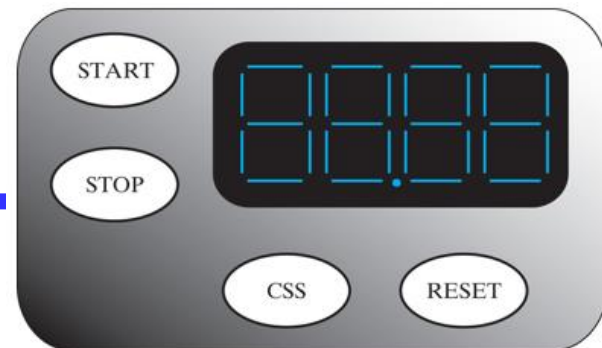
④ 定义控制和状态信号

操作或状态	控制或状态信号	0和1的含义
$TM \leftarrow (0000)_{BCD}$	RSTM	1: Reset TM to 0 (synchronous reset) 0: No reset of TM
$TM \leftarrow (TM + 1)_{BCD}$	ENTM	1: BCD count up TM by 1, 0: hold TM value
$SD \leftarrow (9999)_{BCD}$	UPDATE LSR	0: Select 1001100110011001 for loading SD 1: Enable load SD, 0: disable load SD
$SD \leftarrow TM$	UPDATE LSR	1: Select TM for loading SD Same as above
DIS = TM DIS = SD	DS	0: Select TM for DIS 1: Select SD for DIS
TM < SD TM ≥ SD	ALTB	1: TM less than SD 0: TM greater than or equal to SD

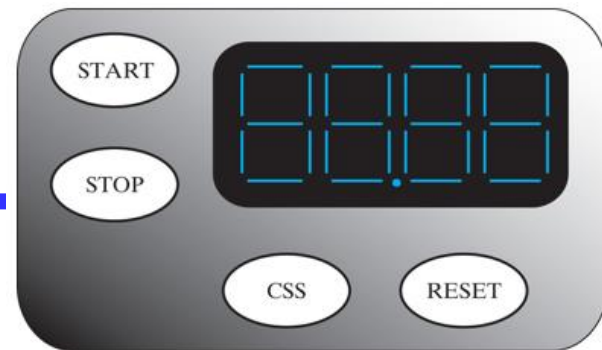
4.3 寄存器传输系统

■ 例：码表

④ 定义控制和状态信号

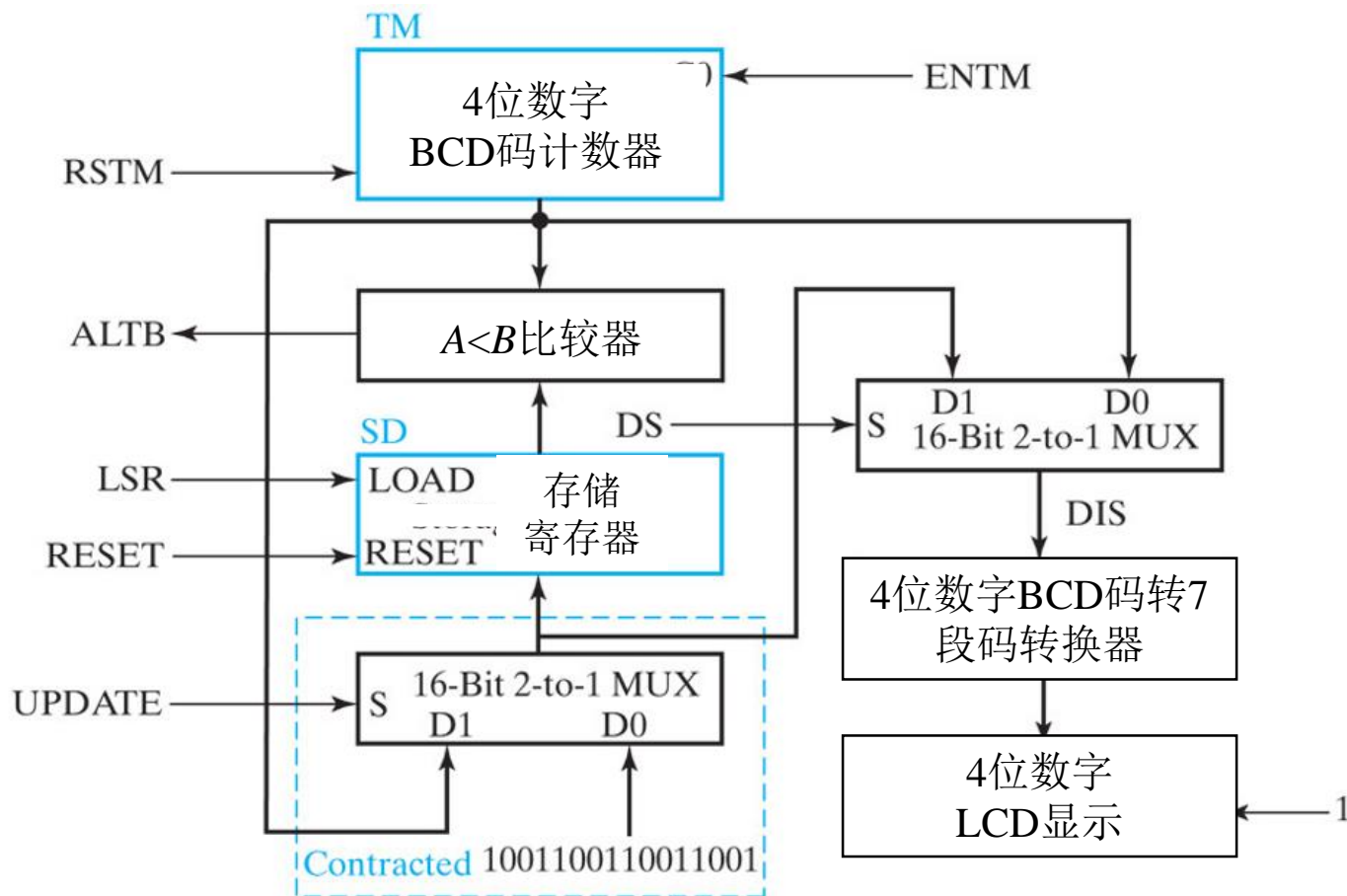


4.3 寄存器传输系统

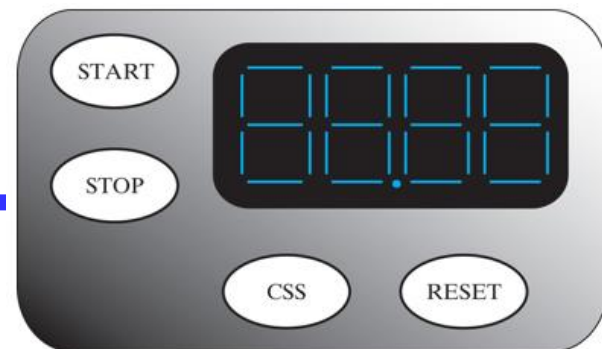


■ 例：码表

⑤ 画出模块图



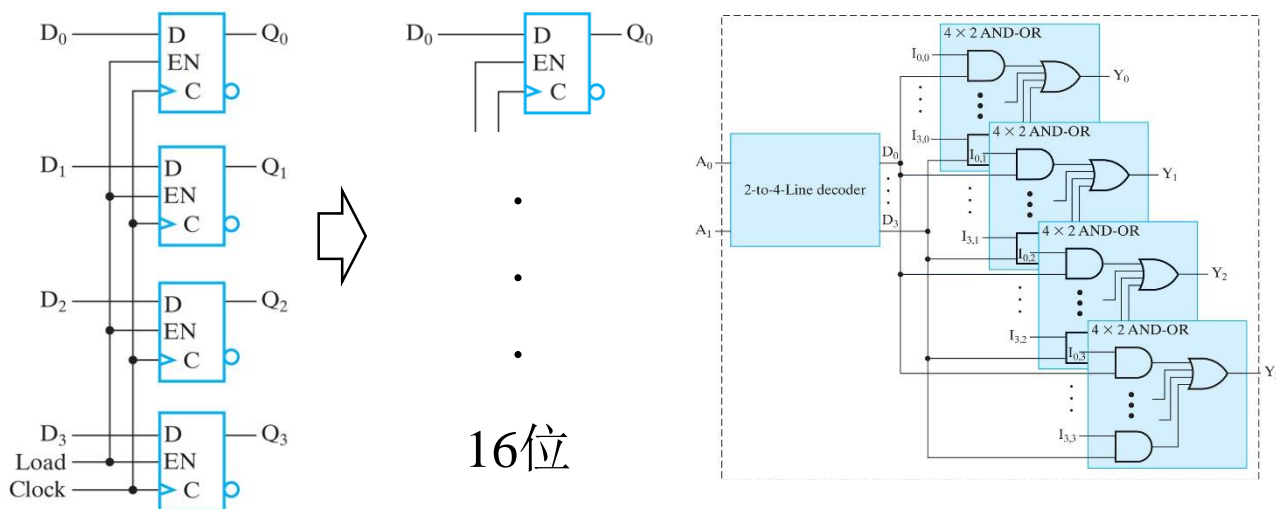
4.3 寄存器传输系统



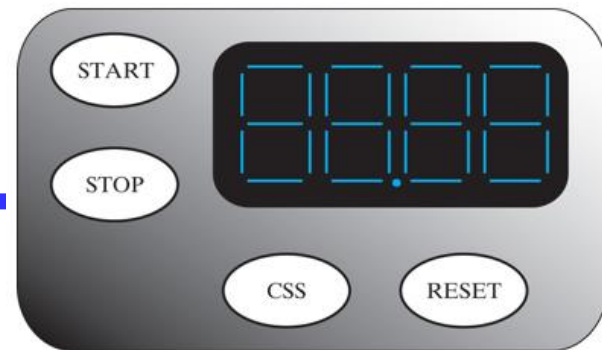
■ 例：码表

⑥ 设计寄存器

- 4位BCD码计数器：4个1位的BCD码计数器级联构成
- 16位并行加载功能的寄存器
- 16位2-1多路复用器

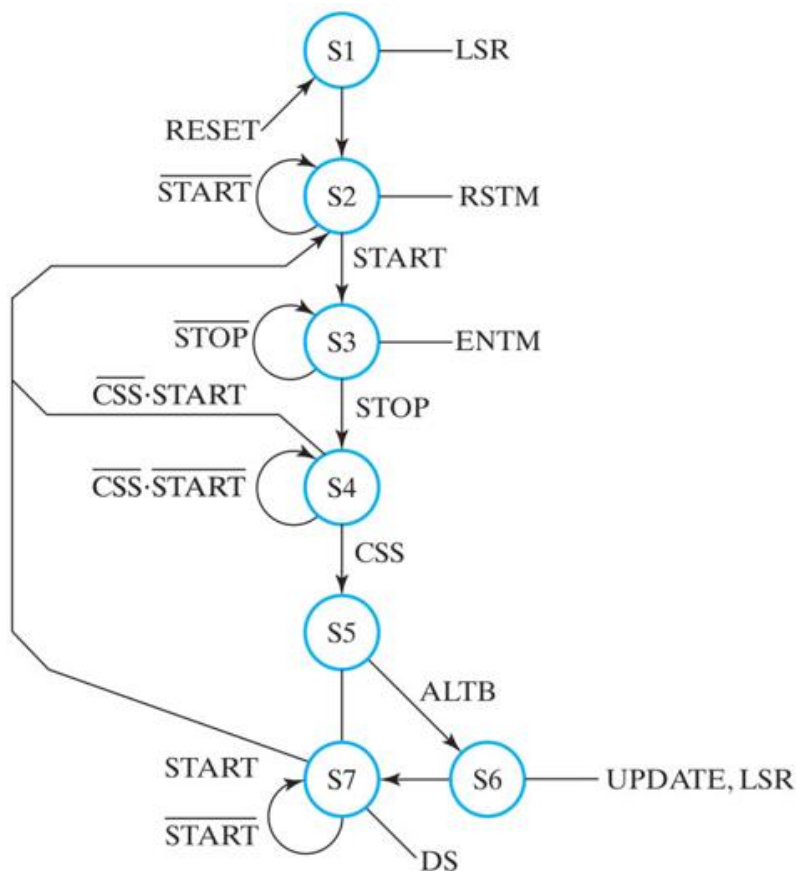


4.3 寄存器传输系统



■ 例：码表

⑦ 设计控制单元



$$\text{LSR} = S1 + S6$$

$$\text{RSTM} = S2$$

$$\text{ENTM} = S3$$

$$\text{UPDATE} = S6$$

$$\text{DS} = S7$$

4.4 微程序控制

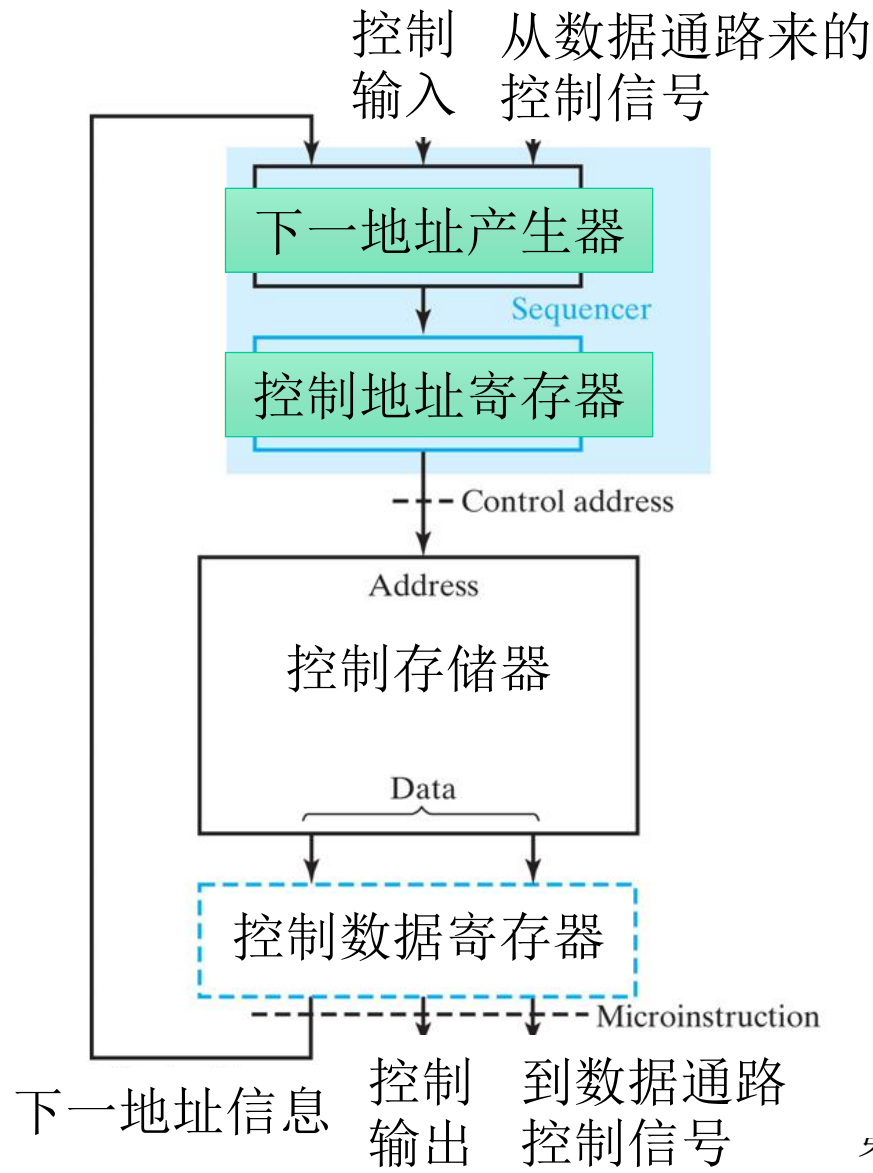
- 用存储在存储器中的字（word）作为控制单元的二进制控制值称为微程序控制
 - 每个字包含一条微指令，用于指定系统中的一条或者多条微操作
 - 一系列微指令组成微程序
 - 控制单元和数据通路所执行的微操作存储在ROM给定地址的字当中
 - 系统启动时可以加载到RAM中

4.4 微程序控制

- 微程序存储在控制存储器
 - 控制地址寄存器：指定微指令存储地址
 - 控制数据寄存器：保存当前执行的微指令
 - 下一地址产生器：生成下一条微指令地址
 - 序列发生器：下一地址生成器与控制地址寄存器的组合

4.4 微程序控制

■ 一般组织结构



提纲

- 1 数字硬件实现
- 2 寄存器及传输
- 3 多寄存器传输
- 4 寄存器传输控制
- 5 小结**

小结

- CMOS晶体管模型
- 寄存器传输
- 寄存器微操作
 - 加载、计数、加法、减法、移位
- 多寄存器传输
- 寄存器传输控制
 - 串行传输
 - 设计方法