Chapter 5
Registers and Register Transfers

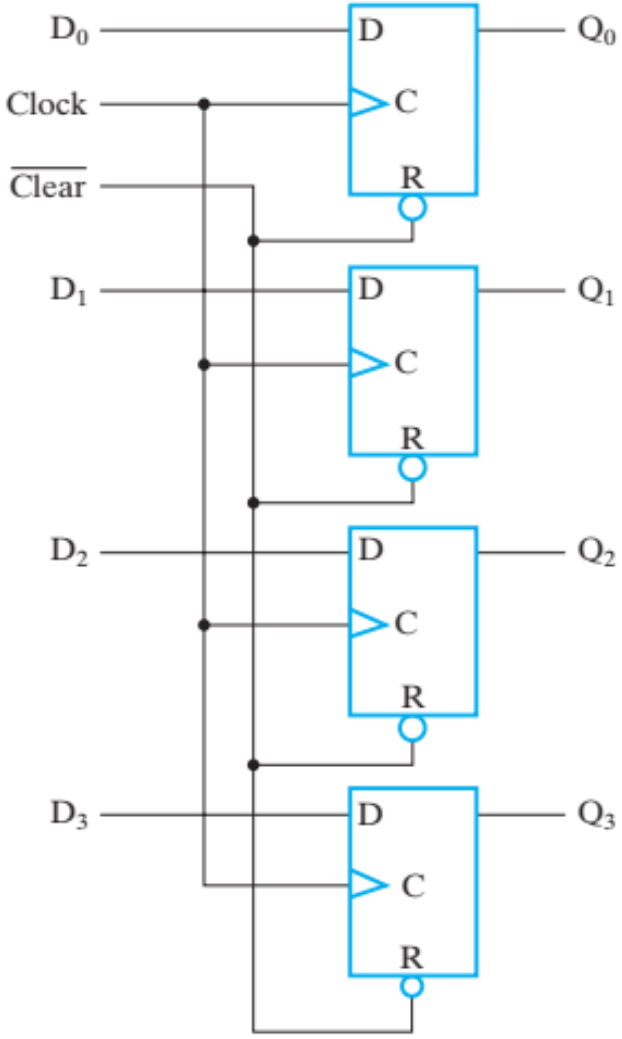
M. Morris Mano, Charles R. Kime. (2015). *Logic and computer design fundamentals* (5th ed.). Pearson.

Contents

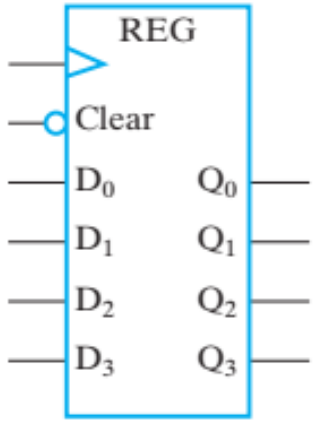
1. Registers and Load Enable

- A register includes a set of flip-flops. Since each flip-flop is capable of storing one bit of information, an *n*-bit register, composed of *n* flip-flops, is capable of storing *n* bits of binary information.
- By the broadest definition, a register consists of a set of flipflops, together with gates that implement their state transitions.
- The transfer of new information into a register is referred to as *loading* the register. If all the bits of the register are loaded simultaneously with a common clock pulse, we say that the loading is done in parallel.

Register with Parallel Load Clock Gating



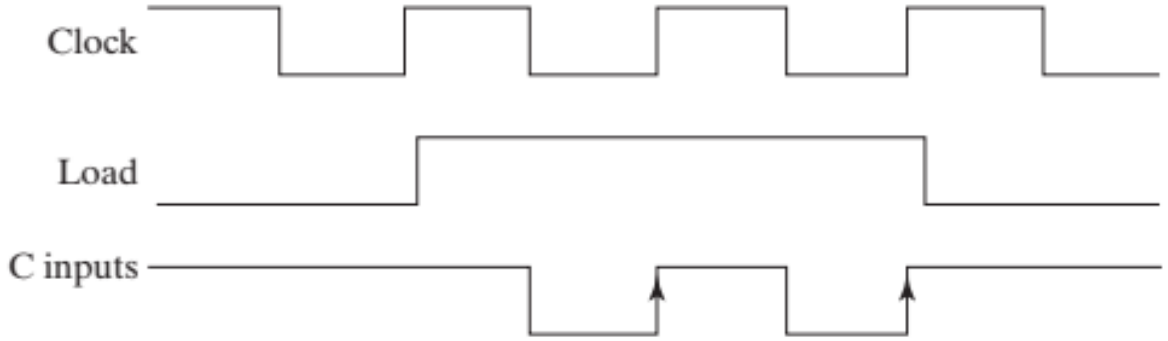
(a) Logic diagram



(b) Symbol

(c) Load control input

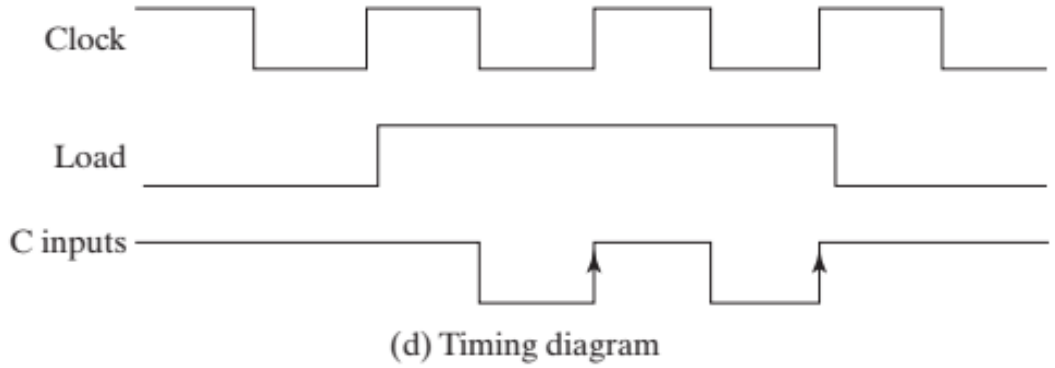
Since the clock is turned on and off at the register C inputs by the use of a logic gate, the technique is referred to as *clock gating*.



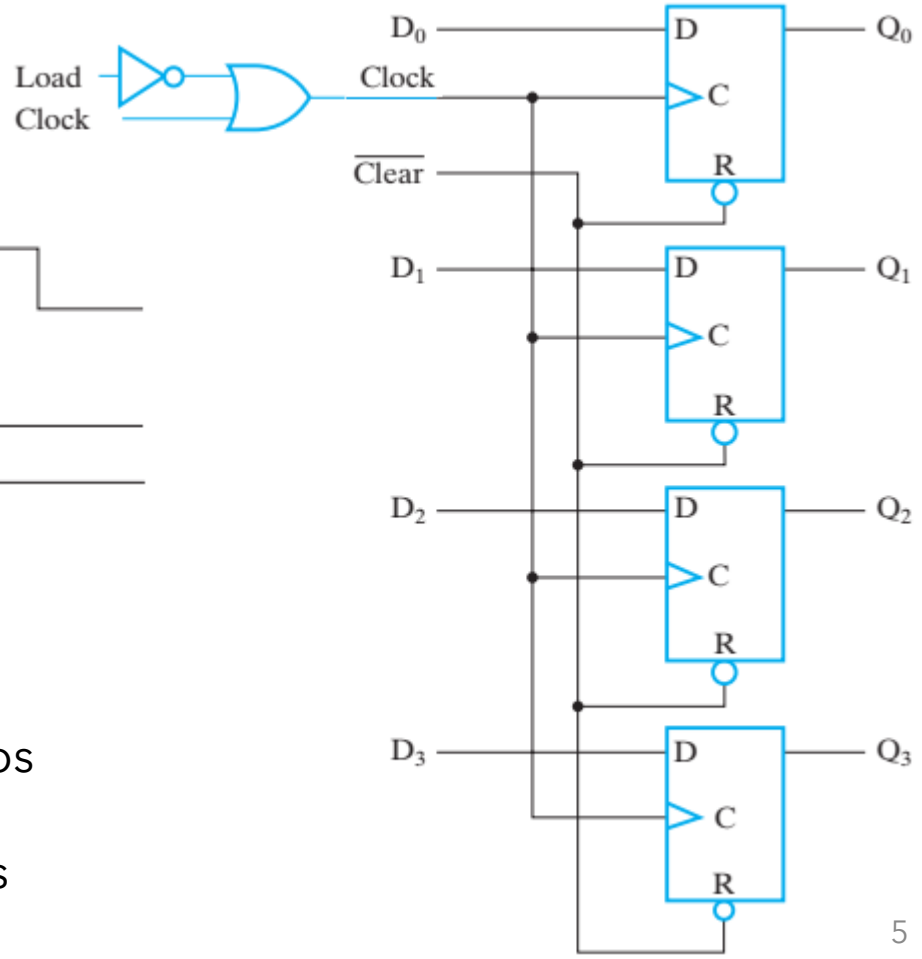
(d) Timing diagram

Register with Parallel Load Clock Gating

Since the clock is turned on and off at the register C inputs by the use of a logic gate, the technique is referred to as *clock gating*.

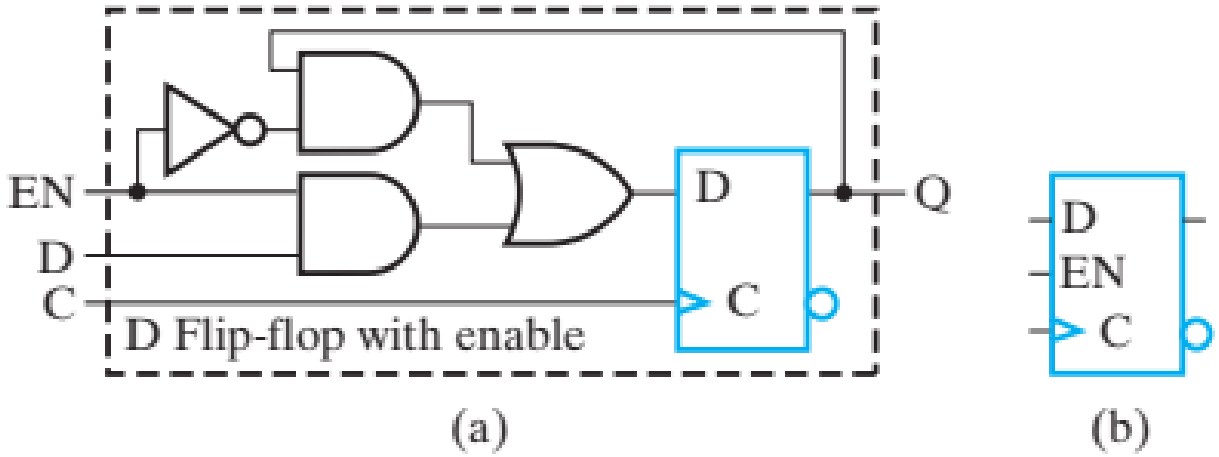


(d) Timing diagram



Inserting gates in the clock pulse path produces different propagation delays between *Clock* and the inputs of flip-flops with and without clock gating. If the clock signals arrive at different flip-flops or registers at different times, *clock skew* is said to exist.

Register with Parallel Load
D Flip-Flop with Enable



The signal *EN* selects between the data bit *D* entering the cell and the value *Q* at the output of the cell.

. EN = 1: *D* is selected and the cell is loaded.

. EN = 0: ***Q* is selected and the output is loaded back into the flip-flop, preserving its current state.**

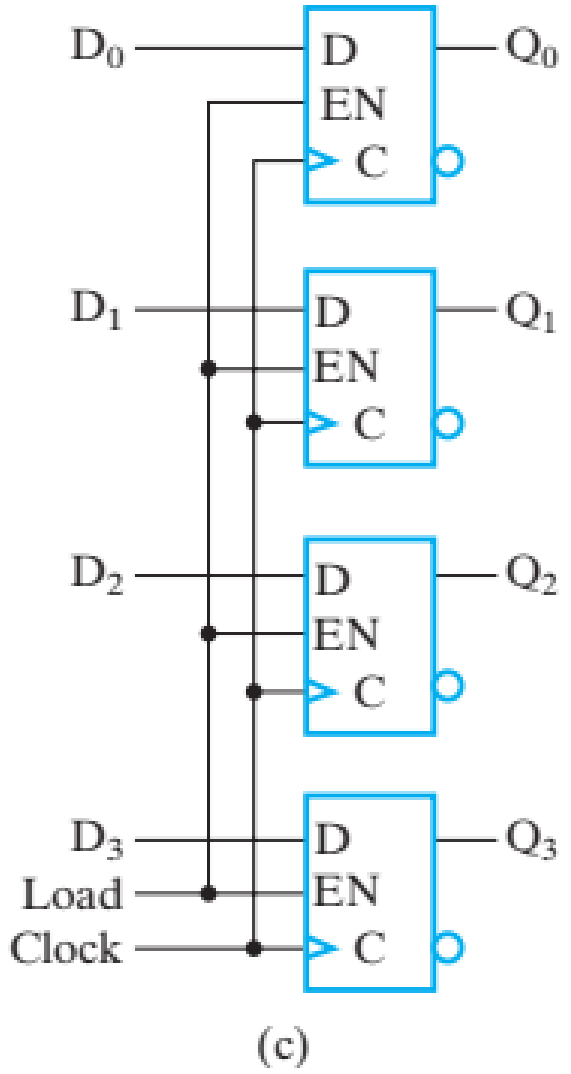


FIGURE 6-2
4-Bit Register with Parallel Load

2. Register Transfers

- The term Register Transfer refers to the availability of hardware logic circuits that can perform a given micro-operation and transfer the result of the operation to the same or another register.
- In most digital system designs, we partition the system into two types of modules:
  - A *datapath:* performs data-processing operations
  - A *control unit:* determines the sequence of those operations.

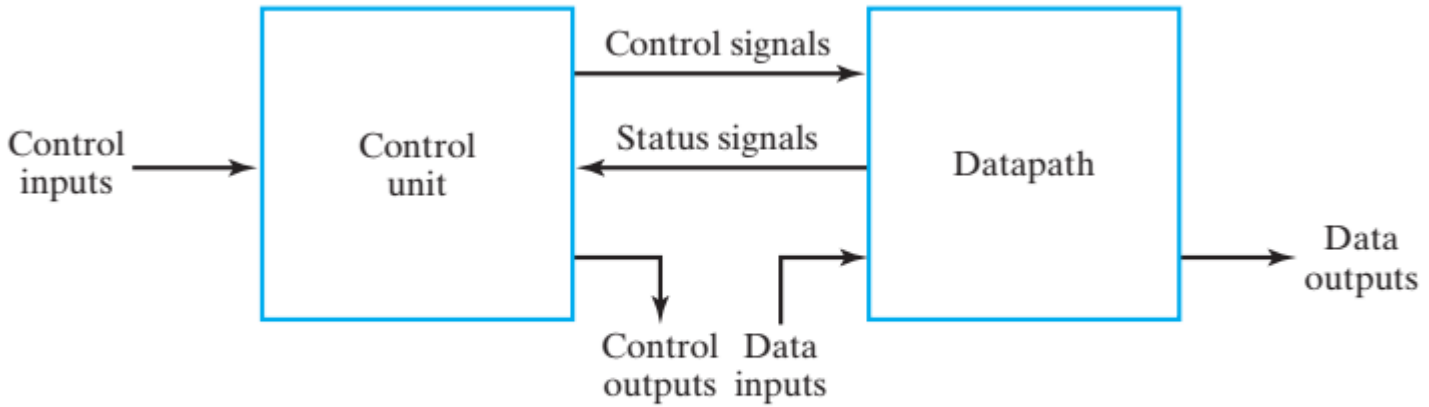


FIGURE 6-3
Interaction Between Datapath and Control Unit

2. Register Transfers

- Datapaths are defined by their registers and the operations performed on binary data stored in the registers.
- The registers are assumed to be basic components of the digital system. The movement of the data stored in registers and the processing performed on the data are referred to as *register transfer operations*.
- A register has the capability to perform one or more *elementary operations* such as load, count, add, subtract, and shift.
  - An elementary operation performed on data stored in registers is called a *microoperation*.

3. Register Transfer Operations

- We denote the registers in a digital system by uppercase letters.
- The individual flip-flops in an *n*-bit register are typically numbered in sequence from 0 to *n* - 1, starting with 0 in the least significant position and increasing toward the most significant position.
  - Since the 0 bit is on the right, this order can be referred to as *little-endian*.
  - The reverse order, with bit 0 on the left, is referred to as *big-endian*.

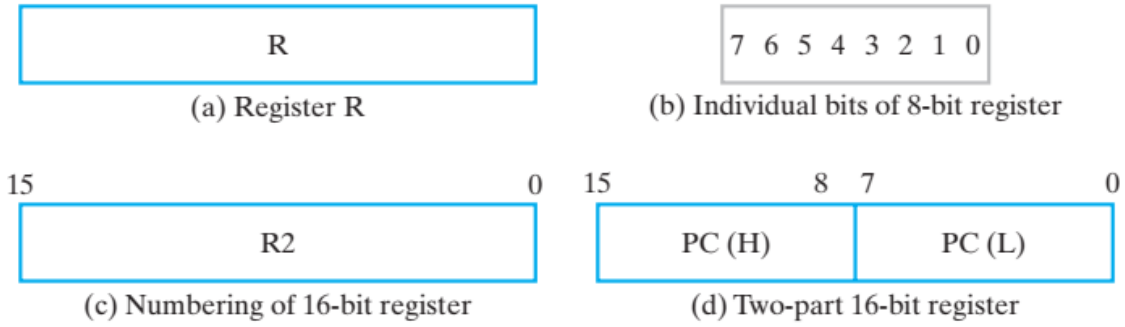


FIGURE 6-4
Block Diagrams of Registers

3. Register Transfer Operations

TABLE 6-1
Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	$AR, R2, DR, IR$
Parentheses	Denotes a part of a register	$R2(1), R2(7{:}0), AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2,\ R2 \leftarrow R1$
Square brackets	Specifies an address for memory	$DR \leftarrow M[AR]$

3. Register Transfer Operations

- Normally, we want a given transfer to occur not for every clock pulse, but only for specific values of the control signals. This can be specified by a *conditional statement*, symbolized by the *if-then* form

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

$$K_1\text{: } R2 \leftarrow R1$$

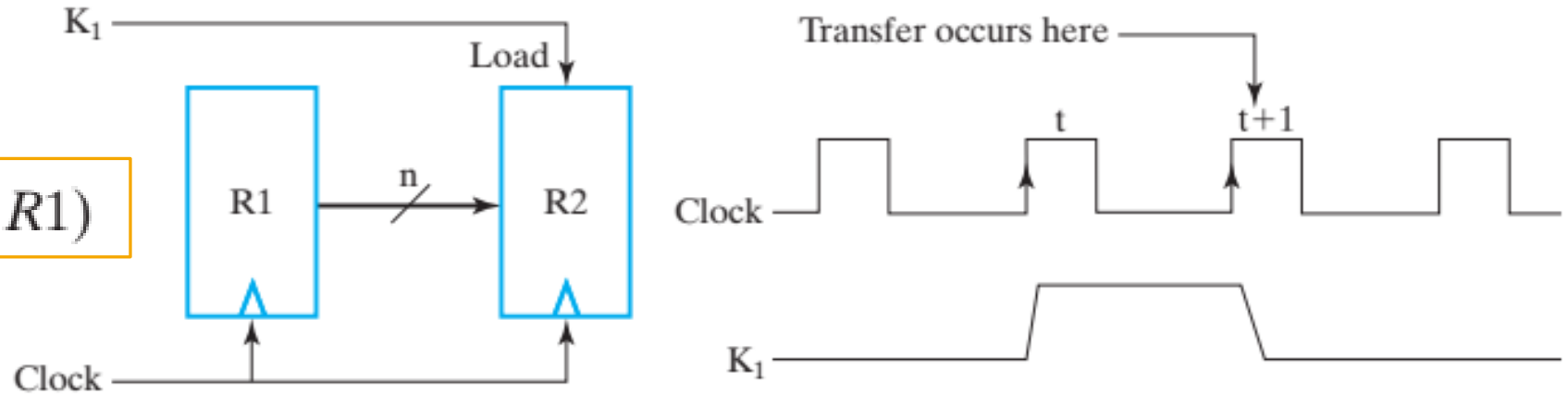


FIGURE 6-5
Transfer from $R1$ to $R2$ when $K_1 = 1$

4. Microoperations

A microoperation is an elementary operation performed on data stored in registers or in memory. The microoperations most often encountered in digital systems are of four types:

1. *Transfer* microoperations, which transfer binary data from one register to another.

   This type of microoperation does not change the binary data bits as they move from the source register to the destination register.

2. *Arithmetic* microoperations, which perform arithmetic operations on data in registers.
3. *Logic* microoperations, which perform bit manipulation on data in registers.
4. *Shift* microoperations, which shift data in registers.

Arithmetic Microoperations

TABLE 6-3
Arithmetic Microoperations

Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1s complement)
$R2 \leftarrow \overline{R2} + 1$	2s complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2s complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

- Multiplication and division are not listed in Table 6-3.
- Multiplication can be represented by the symbol * and division by /.
- These two operations are not included in the basic set of arithmetic microoperations because they are assumed to be implemented by sequences of basic microoperations.
- However, multiplication can be considered as a microoperation if implemented by a combinational circuit.

Arithmetic Microoperations

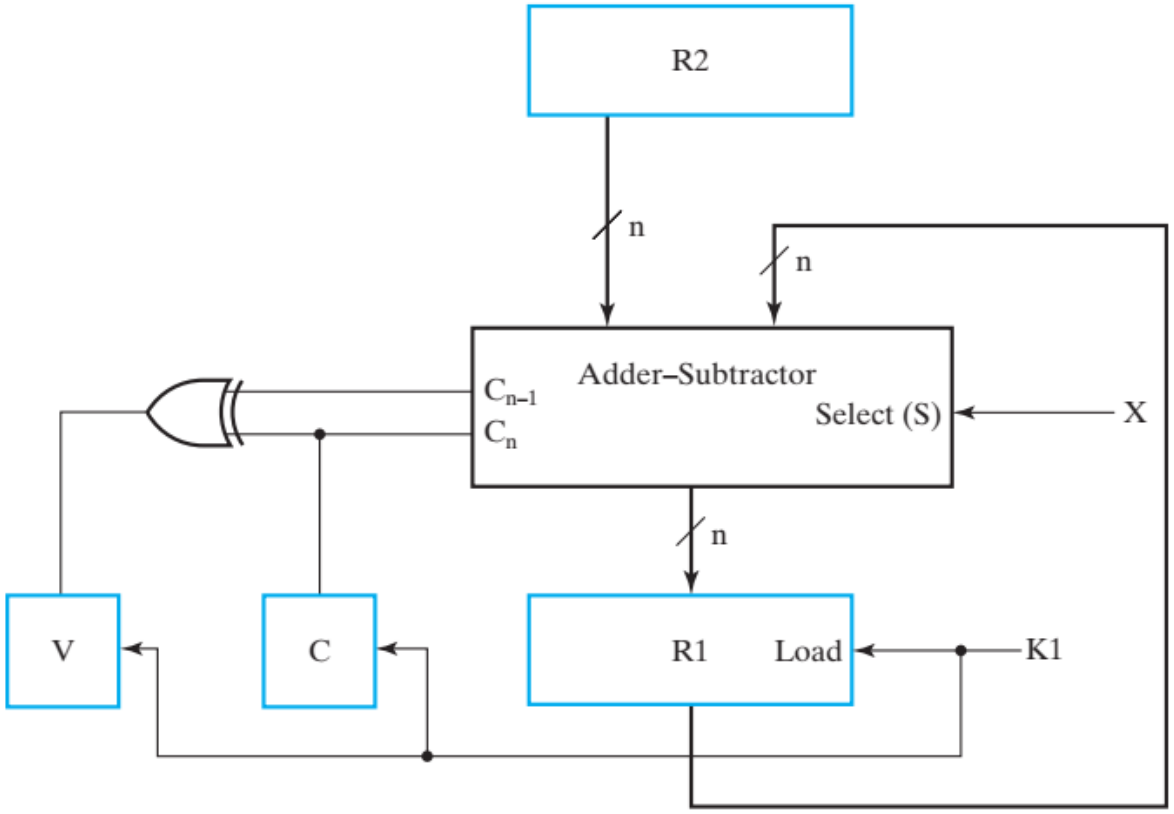


FIGURE 6-6
Implementation of Add and Subtract Microoperations

- The control variable X selects the operation.
  X = 0: Add
  X = 1: Subtract

- The control variable K1 loads the result into R1, the overflow to flip-flop V, the carry to flip-flip C.

Logic Microoperations

- Logic microoperations are useful in manipulating the bits stored in a register. These operations consider each bit in the register separately and treat it as a binary variable.

TABLE 6-4
Logic Microoperations

Symbolic Designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1s complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

Shift Microoperations

- Shift microoperations are used for lateral movement of data. The contents of a source register can be shifted either right or left.
  - A *left shift* is toward the most significant bit.
  - A *right shift* is toward the least significant bit.

TABLE 6-5
Examples of Shifts

		Eight-Bit Examples	
Type	**Symbolic Designation**	**Source *R2***	**After Shift: Destination *R1***
Shift left	$R1 \leftarrow \text{sl } R2$	10011110	00111100
Shift right	$R1 \leftarrow \text{sr } R2$	11100101	01110010

5. Microoperations on a Single Register

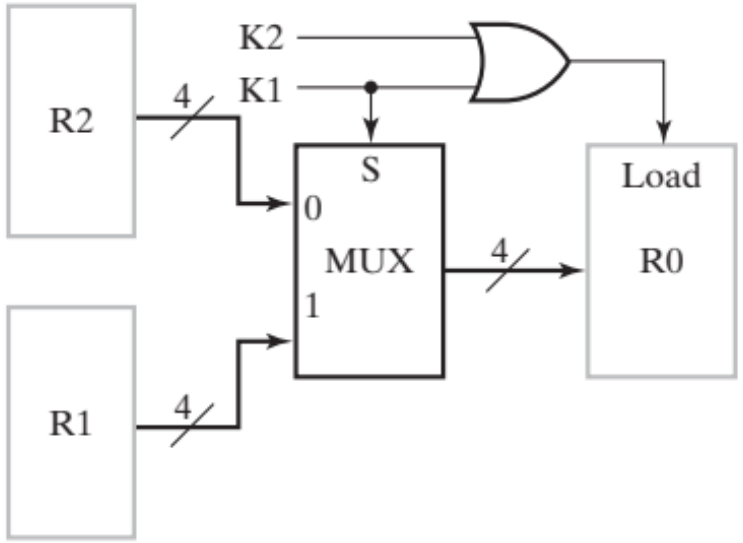
- Due to the close ties between a single set of storage elements and the microoperations, the combinational logic implementing the microoperations is assumed to be a part of the register or is shared by multiple destination registers.
- *Dedicated logic*: The combinational logic implementing the microoperations is a part of the register.
- *Shared logic*: The combinational logic implementing the microoperations is shared my multiple destination registers.

5. Microoperations on a Single Register

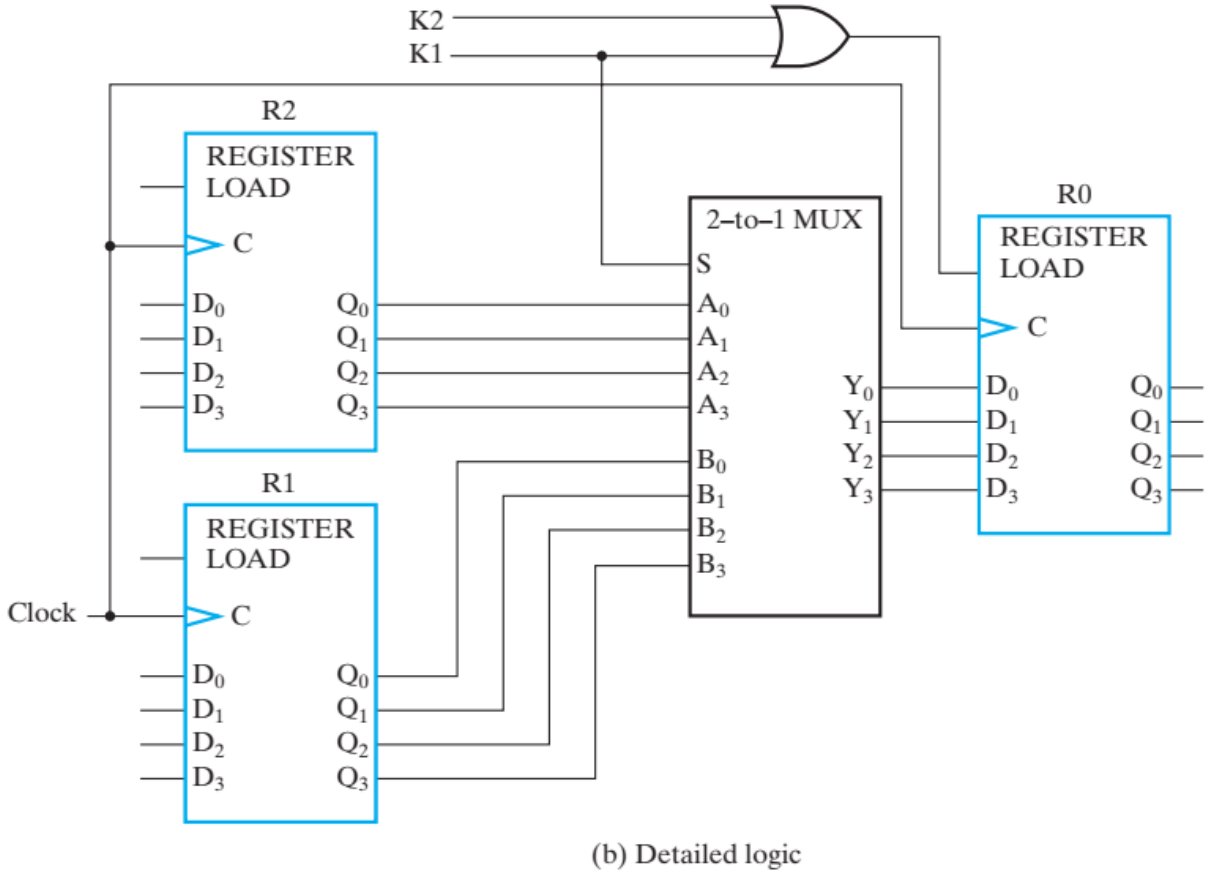
- Multiplexer-Based Transfers
  - There are occasions when a register receives data from two or more different sources at different times.

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

$$K_1: R0 \leftarrow R1, \quad \overline{K}_1 K_2: R0 \leftarrow R2$$



(a) Block diagram



(b) Detailed logic

FIGURE 6-7
Use of Multiplexers to Select Between Two Registers

Multiplexer-Based Transfers

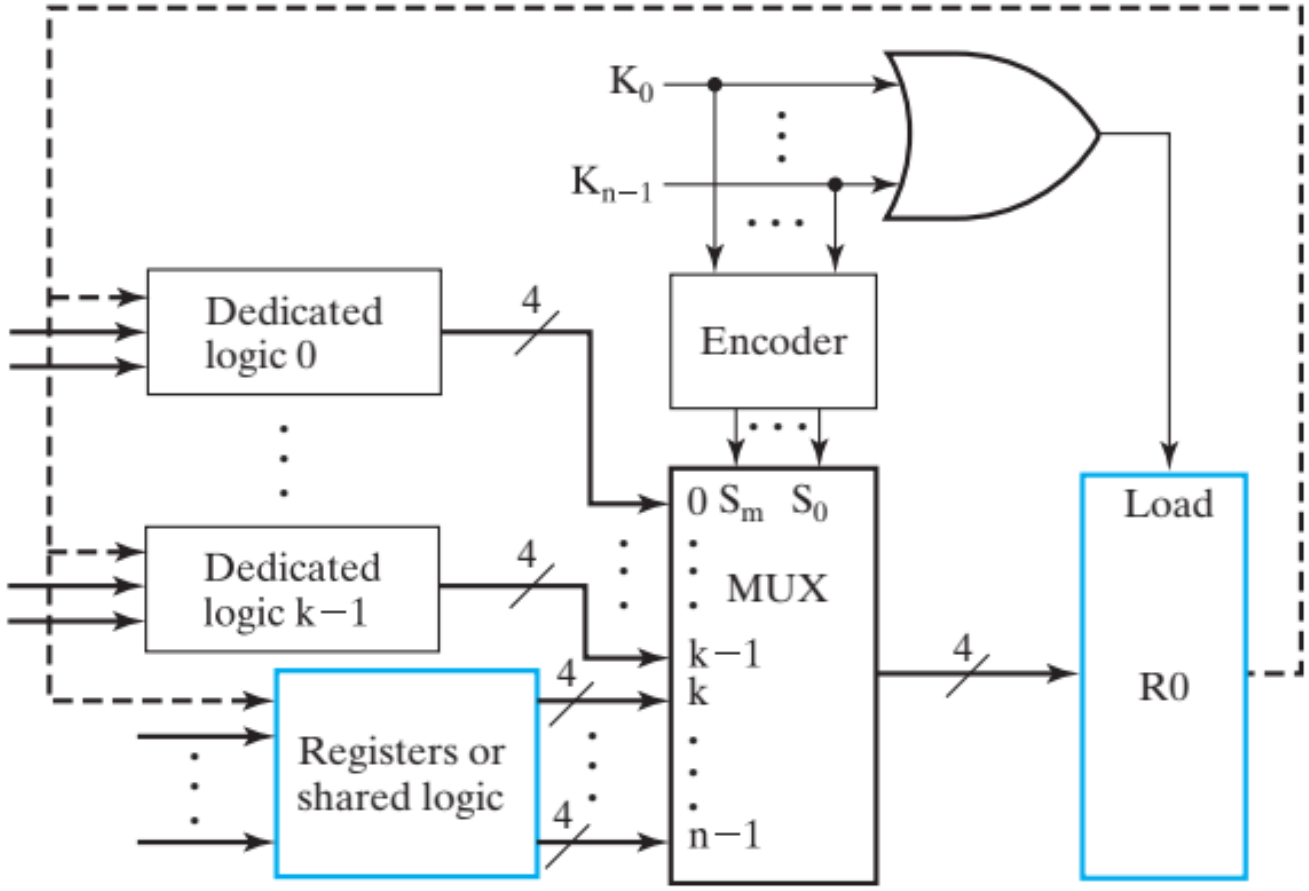
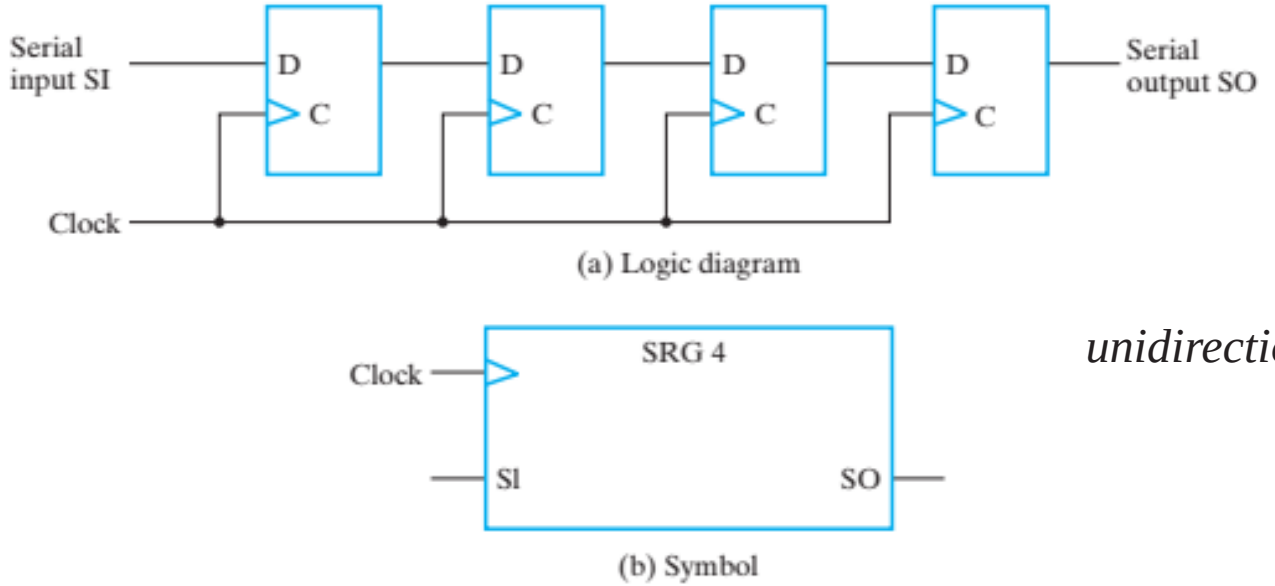


FIGURE 6-8
Generalization of Multiplexer Selection for *n* Sources

Shift Registers

- A register capable of shifting its stored bits laterally in one or both directions is called a *shift register*.



unidirectional shift register

FIGURE 6-9
4-Bit Shift Register

Shift Register with Parallel Load

$$Shift\colon Q \leftarrow \text{sl}\, Q$$

$$\overline{Shift} \cdot Load\colon Q \leftarrow D$$

TABLE 6-6
Function Table for the Register of Figure 6-10

Shift	Load	Operation
0	0	No change (Hold)
0	1	Load parallel data
1	×	Shift left (down) from Q_0 to Q_3

unidirectional shift register

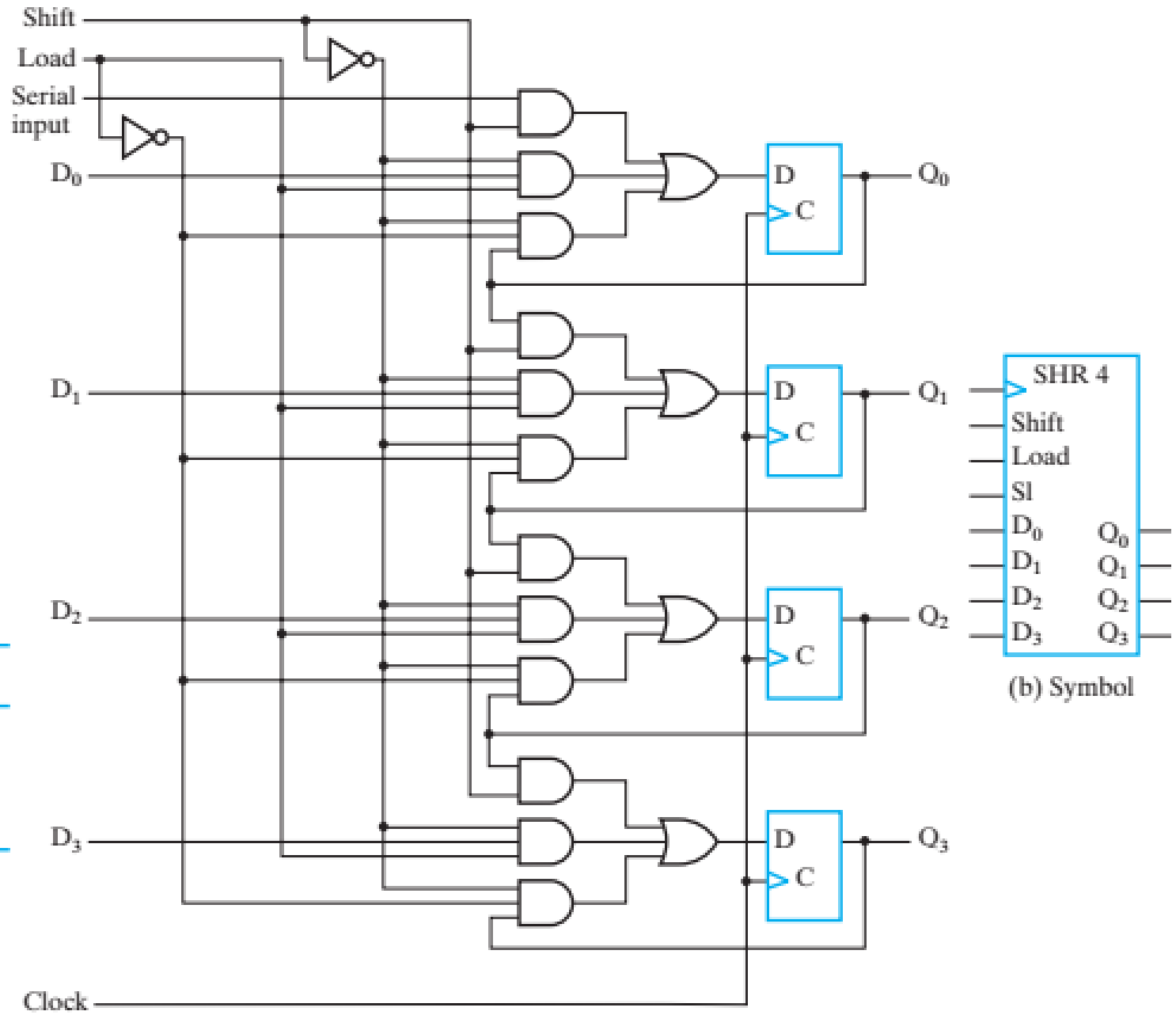


FIGURE 6-10
Shift Register with Parallel Load

Bidirectional Shift Register

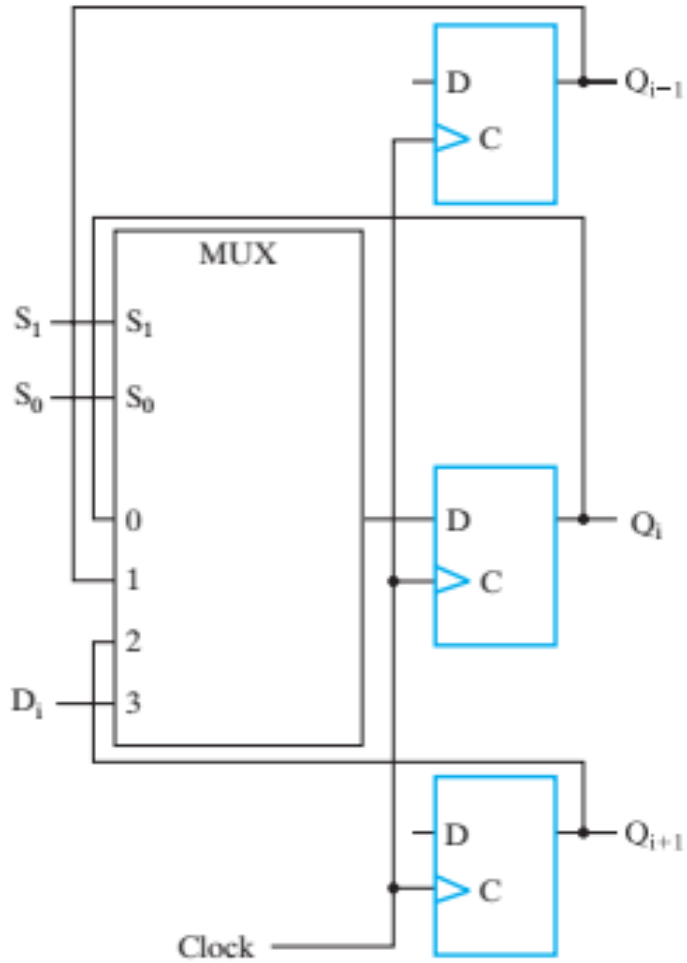
$\overline{S}_1 \cdot S_0$: $Q \leftarrow \text{sl}\, Q$

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

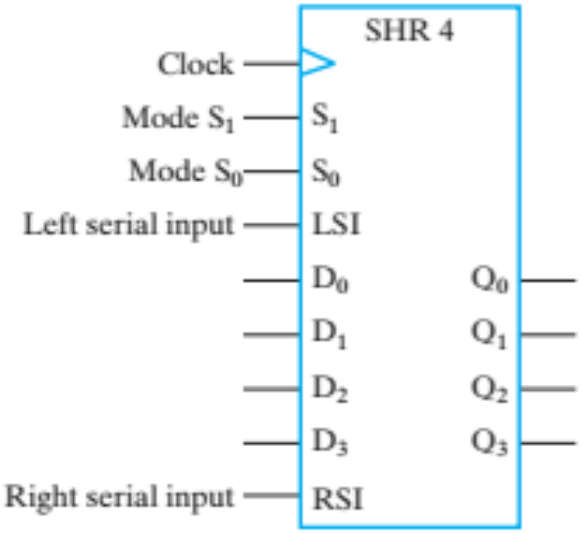
$S_1 \cdot S_0$: $Q \leftarrow D$

TABLE 6-7
Function Table for the Register of Figure 6-11

Mode Control		Register Operation
S_1	S_0	
0	0	No change (Hold)
0	1	Shift left
1	0	Shift right
1	1	Parallel load



(a) Logic diagram of one typical stage



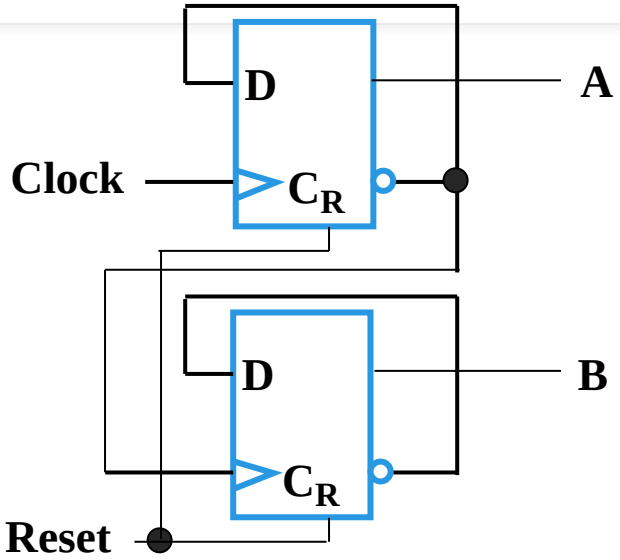
(b) Symbol

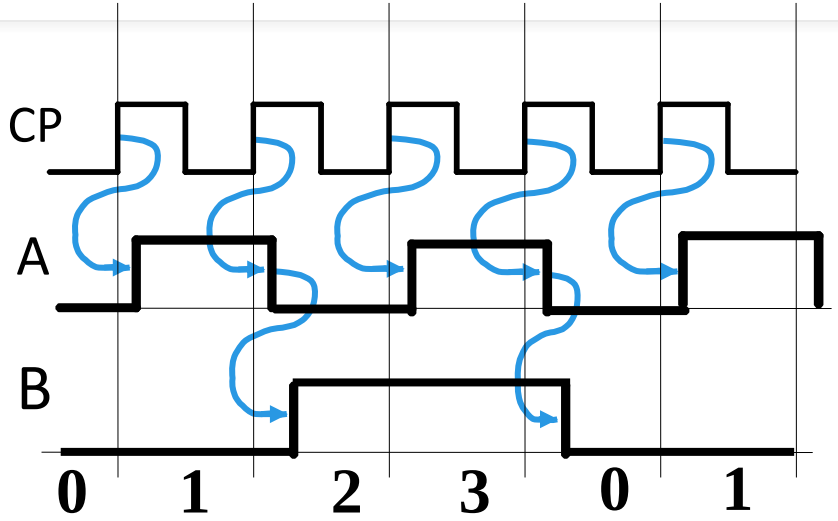
FIGURE 6-11
Bidirectional Shift Register with Parallel Load

Counters

- A register that goes through a prescribed sequence of distinct states upon the application of a sequence of input pulses is called a *counter*.
  - A counter that follows the binary number sequence is called a *binary counter*. An *n*-bit binary counter consists of *n* flip-flops and can count in binary from 0 through 2^n - 1.
- Counters are available in two categories: ripple counters and synchronous counters.
  - Ripple counter: the flip-flop output transitions serve as the sources for triggering the changes in other flip-flops. The C inputs of some of the flip-flops are triggered not by the common clock pulse, but rather by the transitions that occur on other flip-flop outputs.
  - Synchronous counter: the C inputs of all flip-flops receive the common clock pulse, and the change of state is determined from the present state of the counter.

Ripple Counters





- These circuits are called *ripple counters* because each edge sensitive transition (positive in the example) causes a change in the next flip-flop's state.
- The advantage of ripple counters is their simple hardware. Unfortunately, they are asynchronous circuits and, with added logic, can become circuits with delay dependence and unreliable operation.

Ripple Counters

Upward Counting Sequence

Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

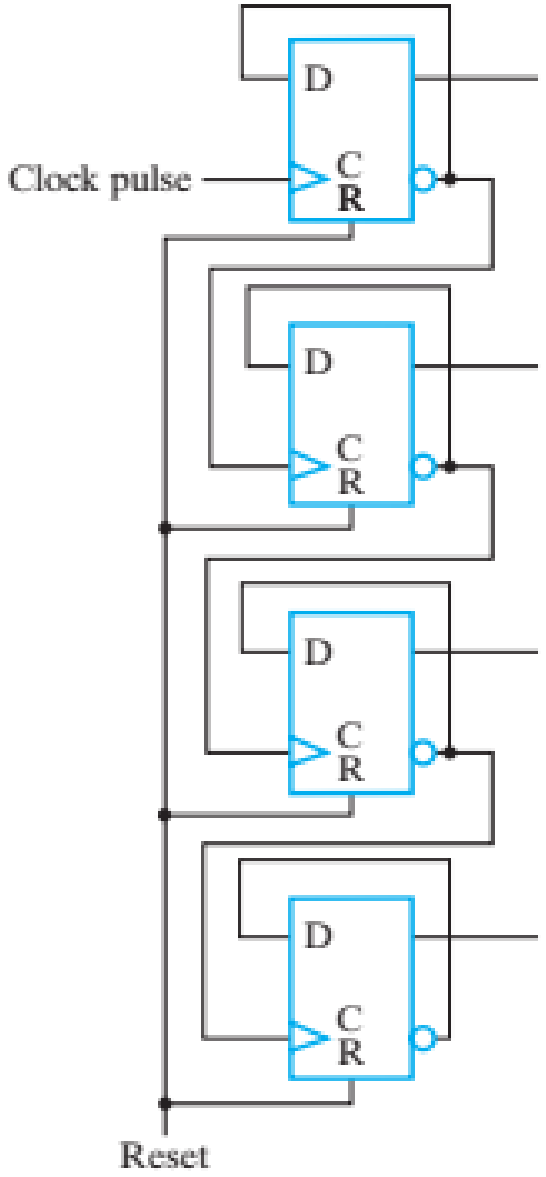


FIGURE 6-12
4-Bit Ripple Counter

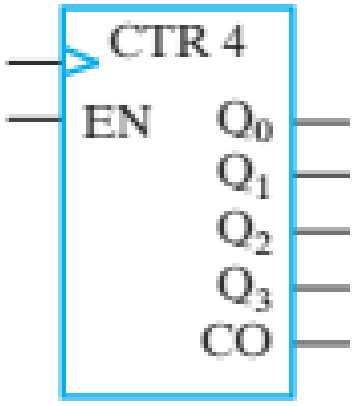
Ripple Counters

Downward Counting Sequence			
Q_3	Q_2	Q_1	Q_0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

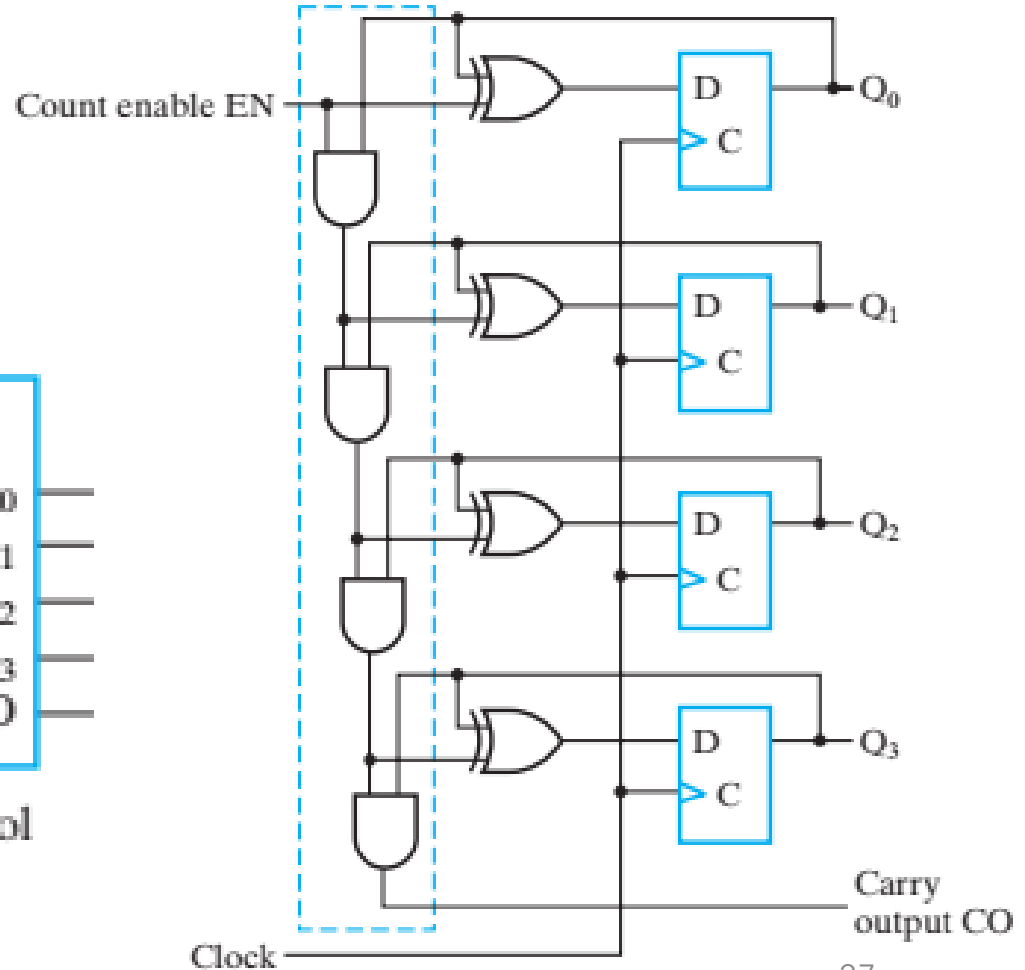
?

Synchronous Binary Counters

- Synchronous counters, in contrast to ripple counters, have the clock applied to the C inputs of all flip-flops.
  - Thus, the common clock pulse triggers all flip-flops simultaneously rather than one at a time, as in a ripple counter.



(c) Symbol



(a) Logic diagram—serial gating