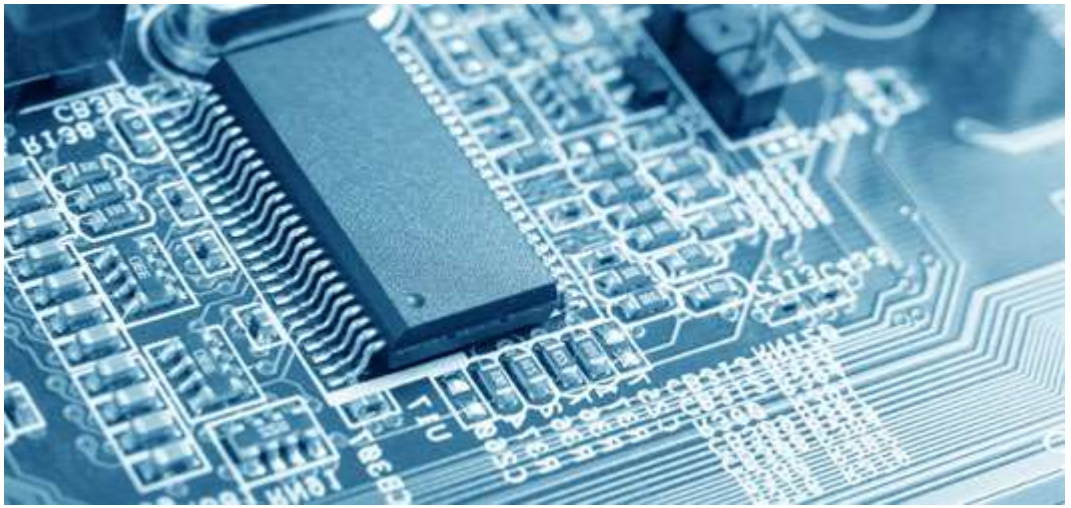
NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY



ROBT206 – Microcontrollers with Lab

Lecture 17-18 – Registers and Microoperations

27-29 March, 2018

Topics

Today's Topics

- Registers and load enable
- Register transfer operations
- Microoperations - arithmetic, logic, and shift
- Microoperations on a single register
  - Multiplexer-based transfers
  - Shift registers

Registers

- Register – a collection of binary storage elements
- In theory, a register is sequential logic which can be defined by a state table
- More often, think of a register as storing a vector of binary values
- Frequently used to perform simple data storage and data movement and processing operations

Register Design Models

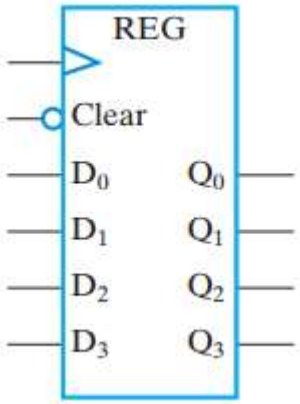
- Due to the large numbers of states and input combinations as *n* becomes large, the state diagram/state table model is not feasible!
- What are methods we can use to design registers?
  - Add predefined combinational circuits to registers
    - Example: To count up, connect the register flip-flops to an incrementor
  - Design individual cells using the state diagram/state table model and combine them into a register
    - A 1-bit cell has just two states
    - Output is usually the state variable

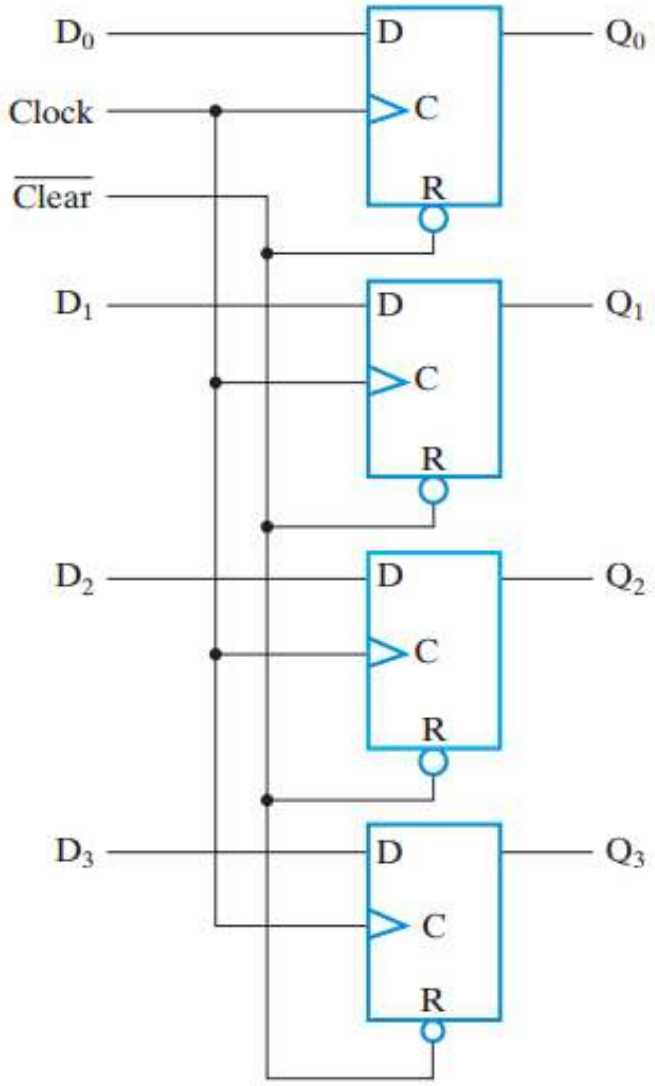
Register Storage

- Expectations:
  - A register can store information for multiple clock cycles
  - To "store" or "load" information should be controlled by a signal
- Reality:
  - A D flip-flop register loads information on every clock cycle
- Realizing expectations:
  - Use a signal to block the clock to the register,
  - Use a signal to control feedback of the output of the register back to its inputs, or
  - Use other SR or JK flip-flops, that for (0,0) applied, store their state
- Load is a frequent name for the signal that controls register storage and loading
  - Load = 1: Load the values on the data inputs
  - Load = 0: Store the values in the register

Register with Parallel Load

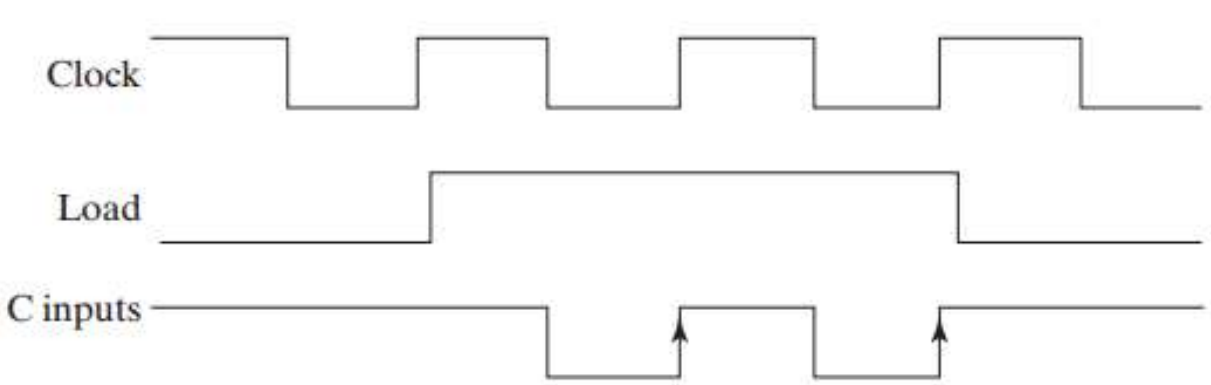
- Register: Group of Flip-Flops
- Ex: D Flip-Flops
- Holds a Word (Nibble) of Data
- Loads in Parallel on Clock Transition
- Asynchronous Clear (Reset)





Clock Gating

- The *Load* signal enables the clock signal to pass through if 1 and prevents the clock signal from passing through if 0.
- Example: For Positive Edge-Triggered or Negative Pulse Master-Slave Flip-flop:

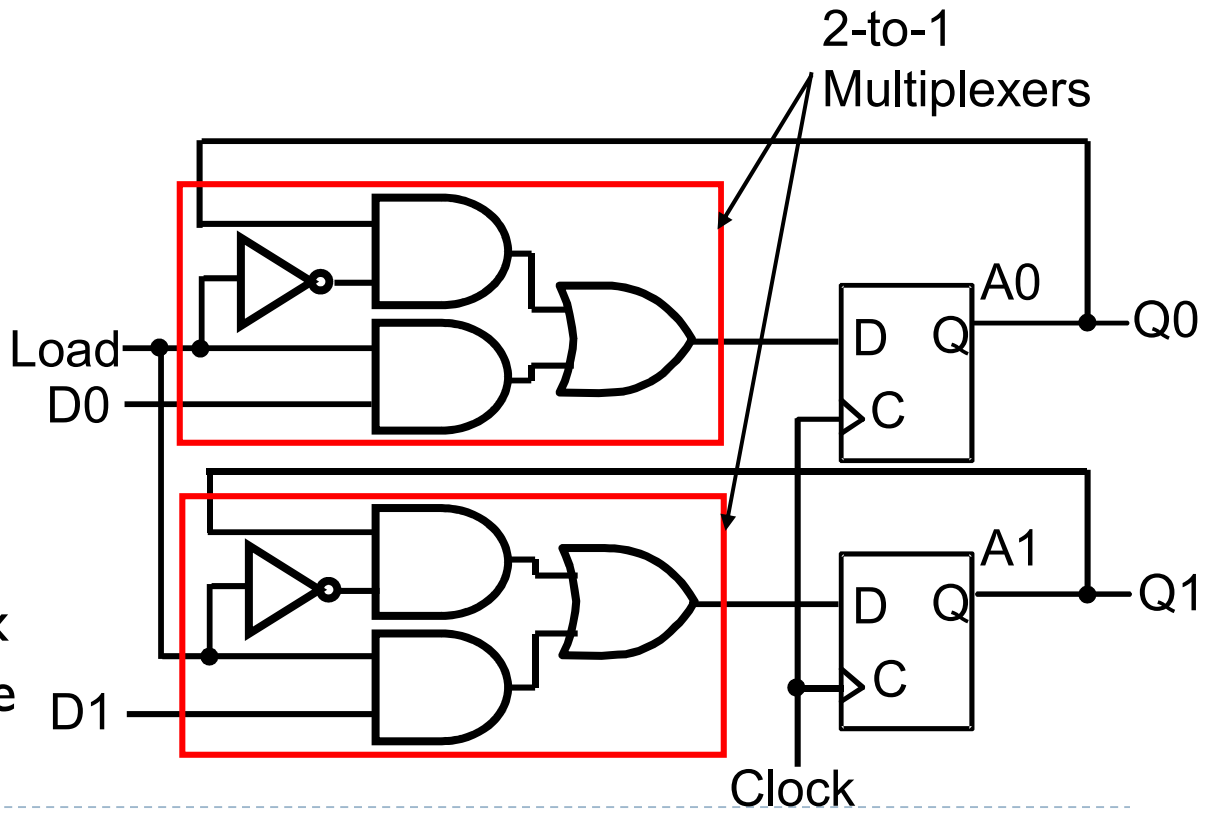


$$C \text{ inputs} = \overline{Load} + Clock$$

Registers with Load-Controlled Feedback

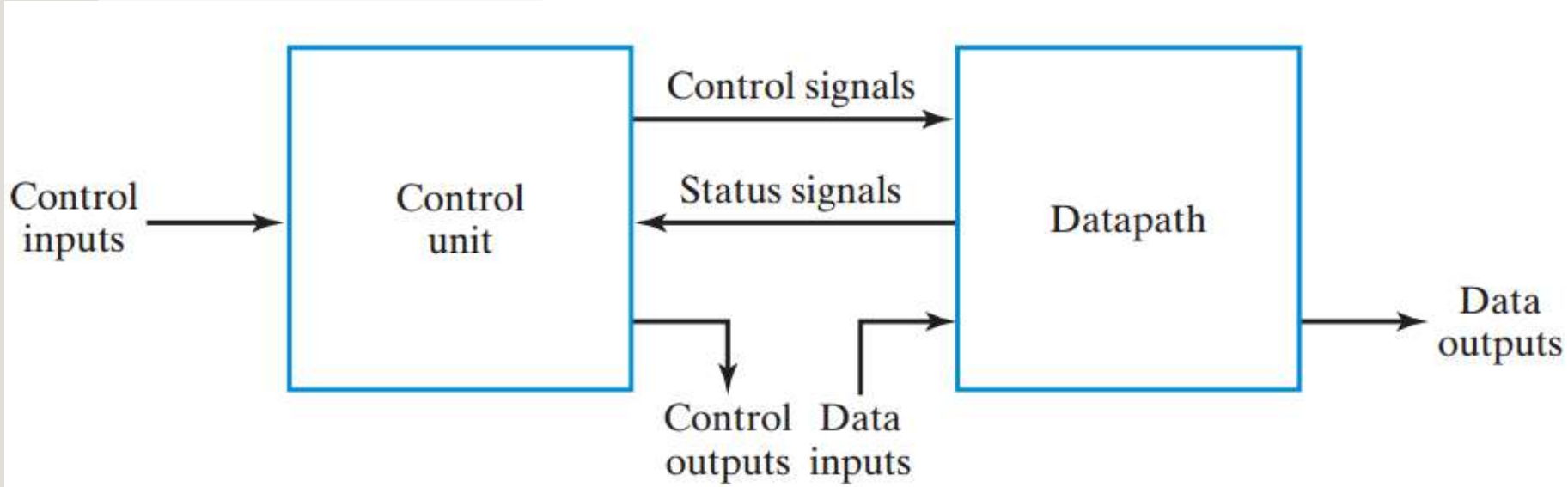
- A more reliable way to selectively load a register:
  - Run the clock continuously, and
  - Selectively use a load control to change the register contents.

- Example: 2-bit register with Load Control:
- For Load = 0, loads register contents (hold current values)
- For Load = 1, loads input values (load new values)
- Hardware more complex than clock gating, but free of timing problems



Register Transfer

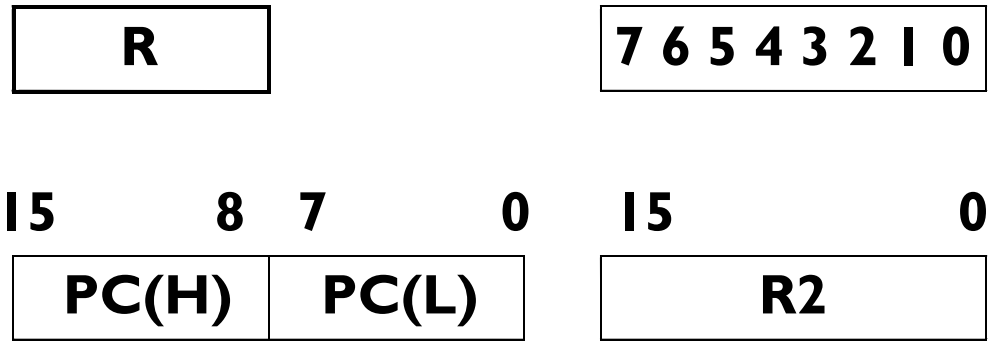
- Most digital systems can be split to:
- **datapath** – performs data performing operations
- **control unit –** determines the sequence of those operations
- Datapaths are defined by their registers and the operations performed on binary data stored in the registers



Register Transfer Operations

- **Register Transfer Operations** – The movement and processing of data stored in registers
- Three basic components:
  - set of registers
  - operations
  - control of operations
- Elementary Operations **-- load, count, shift, add, bitwise "OR", etc.**
  - Elementary operations called ***microoperations***
- A microoperation is usually performed in parallel on a vector of bits during one clock cycle

Register Notation



- Letters and numbers – denotes a register (ex. R2, PC, IR)
- Parentheses ( ) – denotes a range of register bits (ex. R1(1), PC(7:0), PC(L))
- Arrow (←) – denotes data transfer (ex. R1 ← R2, PC(L) ← R0)
- Comma – separates parallel operations R1 ← R2, R2 ← R1
- Brackets [ ] – Specifies a memory address

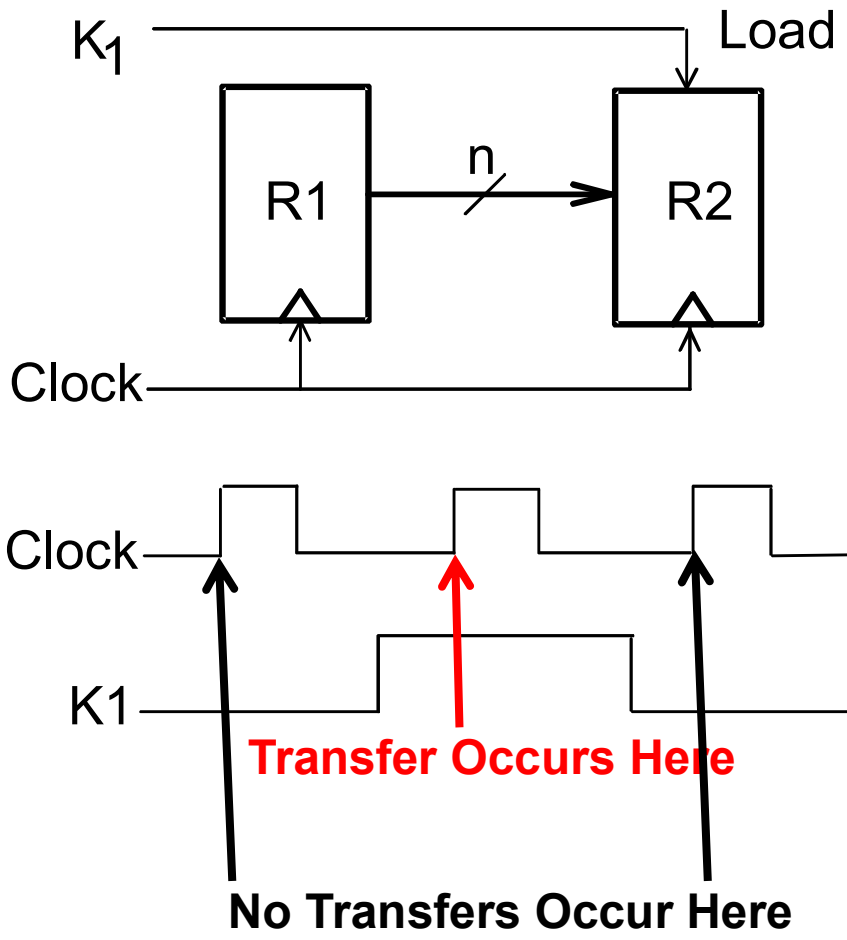
  (ex. R0 ← M[AR], R3 ← M[PC])

Conditional Transfer

- If (K1 =1) then (R2 ← R1) is shortened to

  K1: (R2 ← R1)

  where K1 is a control variable specifying a conditional execution of the microoperation.



Microoperations

- Logical Groupings:
  - Transfer - move data from one register to another
  - Arithmetic - perform arithmetic on data in registers
  - Logic - manipulate data or use bitwise logical operations
  - Shift - shift data in registers

Arithmetic operations

+ Addition

– Subtraction

* Multiplication

/ Division

Logical operations

$\vee$ Logical OR

$\wedge$ Logical AND

$\oplus$ Logical Exclusive OR

$\overline{\ }$ NOT

Example Microoperations

- Add the content of R1 to the content of R2 and place the result in R1.

  $R1 \leftarrow R1 + R2$

- Multiply the content of R1 by the content of R6 and place the result in PC.

  $PC \leftarrow R1 * R6$

- Exclusive OR the content of R1 with the content of R2 and place the result in R1.

  $R1 \leftarrow R1 \oplus R2$

Example Microoperations (Continued)

- Take the 1's Complement of the contents of R2 and place it in the PC.

  $PC \leftarrow \overline{R2}$

- On condition K1 OR K2, the content of R1 is Logic bitwise ORred with the content of R3 and the result placed in R1.

  (K1 + K2): $R1 \leftarrow R1 \vee R3$

- NOTE: "+" (as in $K_1 + K_2$) and means "OR." In $R1 \leftarrow R1 + R3$, + means "plus."

Control Expressions

- The control expression for an operation appears to the left of the operation and is separated from it by a colon
- Control expressions specify the logical condition for the operation to occur
- Control expression values of:
  - Logic "1" -- the operation occurs.
  - Logic "0" -- the operation is does not occur.

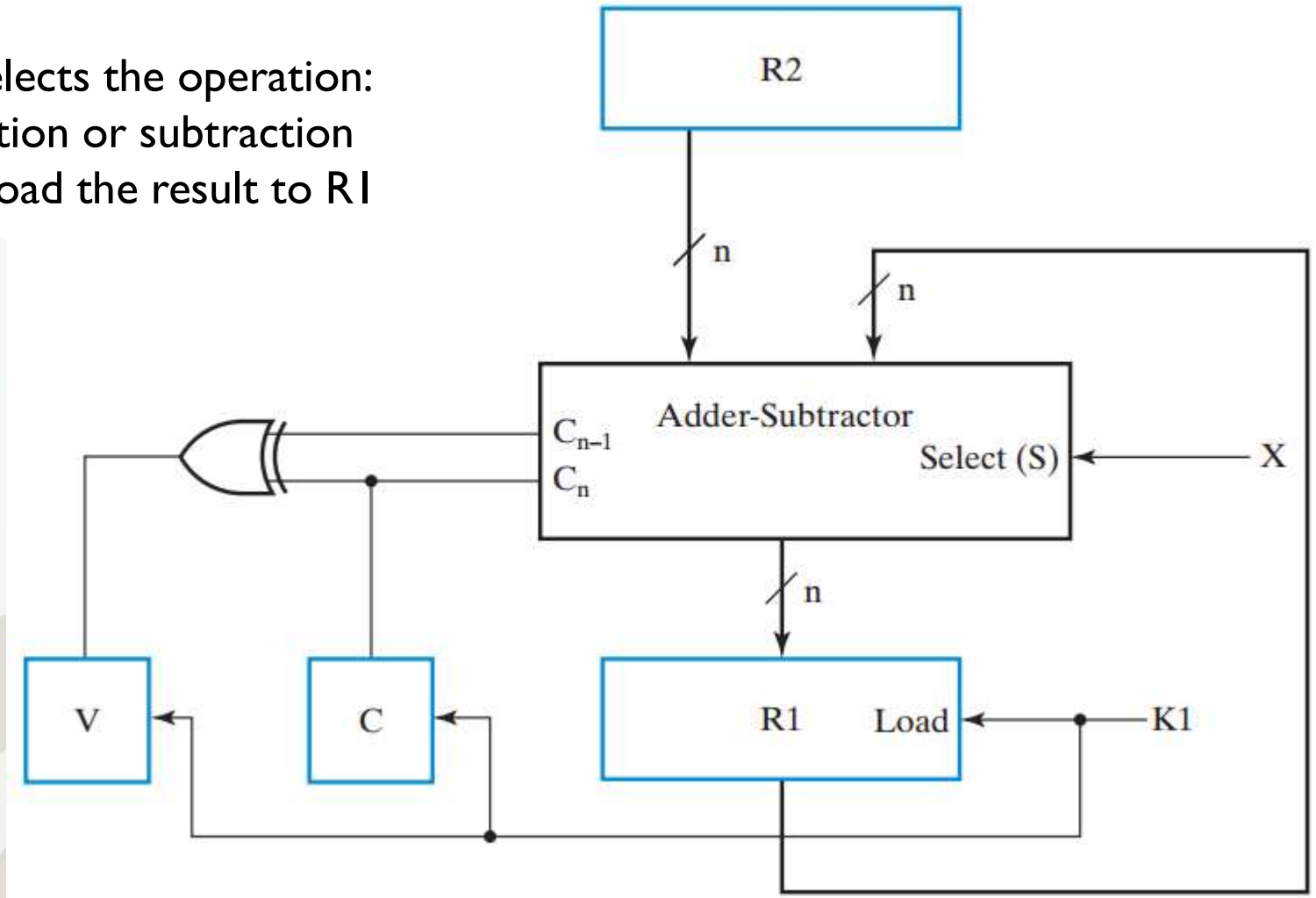
- Example:

  $\overline{X}\,K1 : R1 \leftarrow R1 + R2$

  $X\,K1 : R1 \leftarrow R1 + \overline{R2} + 1$
- Variable K1 enables the add or subtract operation.
- If X =0, then $\overline{X}$ =1 so $\overline{X}\,K1 = 1$, activating the addition of R1 and R2.
- If X = 1, then X K1 = 1, activating the addition of R1 and the two's complement of R2 (subtract).

Control Expressions

- X selects the operation: addition or subtraction
- K1 load the result to R1



Arithmetic Microoperations

Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Addition
$R0 \leftarrow \overline{R1}$	Ones Complement
$R0 \leftarrow \overline{R1} + 1$	Two's Complement
$R0 \leftarrow R2 + \overline{R1} + 1$	R2 minus R1 (2's Comp)
$R1 \leftarrow R1 + 1$	Increment (count up)
$R1 \leftarrow R1 - 1$	Decrement (count down)

- Note that any register may be specified for source 1, source 2, or destination.
- These simple microoperations operate on the whole word

Logical Microoperations

Symbolic Designation	Description
$R0 \leftarrow \overline{R1}$	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)

Logical Microoperations

- Let R1 = 10101010,
  and R2 = 11110000
- Then after the operation, R0 becomes:

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

Shift Microoperations

- Let R2 = 11001001
- Then after the operation, R1 becomes:

Symbolic Designation	Description
R1 ← sl R2	Shift Left
R1 ← sr R2	Shift Right

R1	Operation
10010010	R1 ← sl R2
01100100	R1 ← sr R2

- Note: These shifts "zero fill". Sometimes a separate flip-flop is used to provide the data shifted in, or to "catch" the data shifted out.

Register Transfer Structures

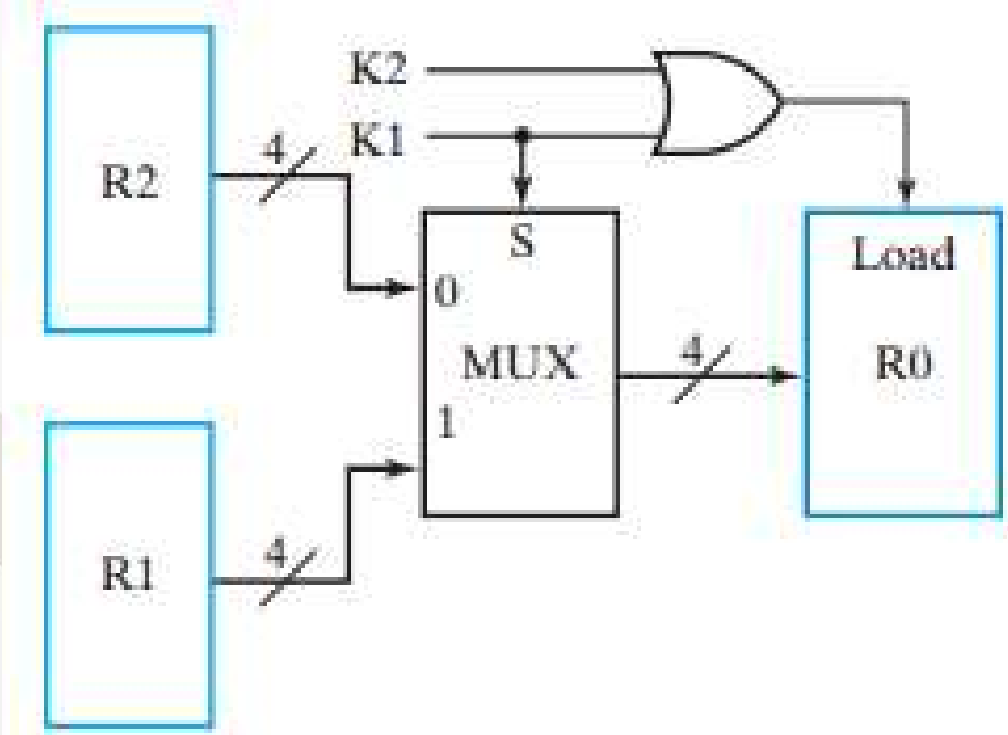
- **Multiplexer-Based Transfers** - Multiple inputs are selected by a multiplexer dedicated to the register
- **Bus-Based Transfers** - Multiple inputs are selected by a shared multiplexer driving a bus that feeds inputs to multiple registers
- **Three-State Bus** - Multiple inputs are selected by 3-state drivers with outputs connected to a bus that feeds multiple registers
- **Other Transfer Structures** - Use multiple multiplexers, multiple buses, and combinations of all the above

Multiplexer-Based Transfers

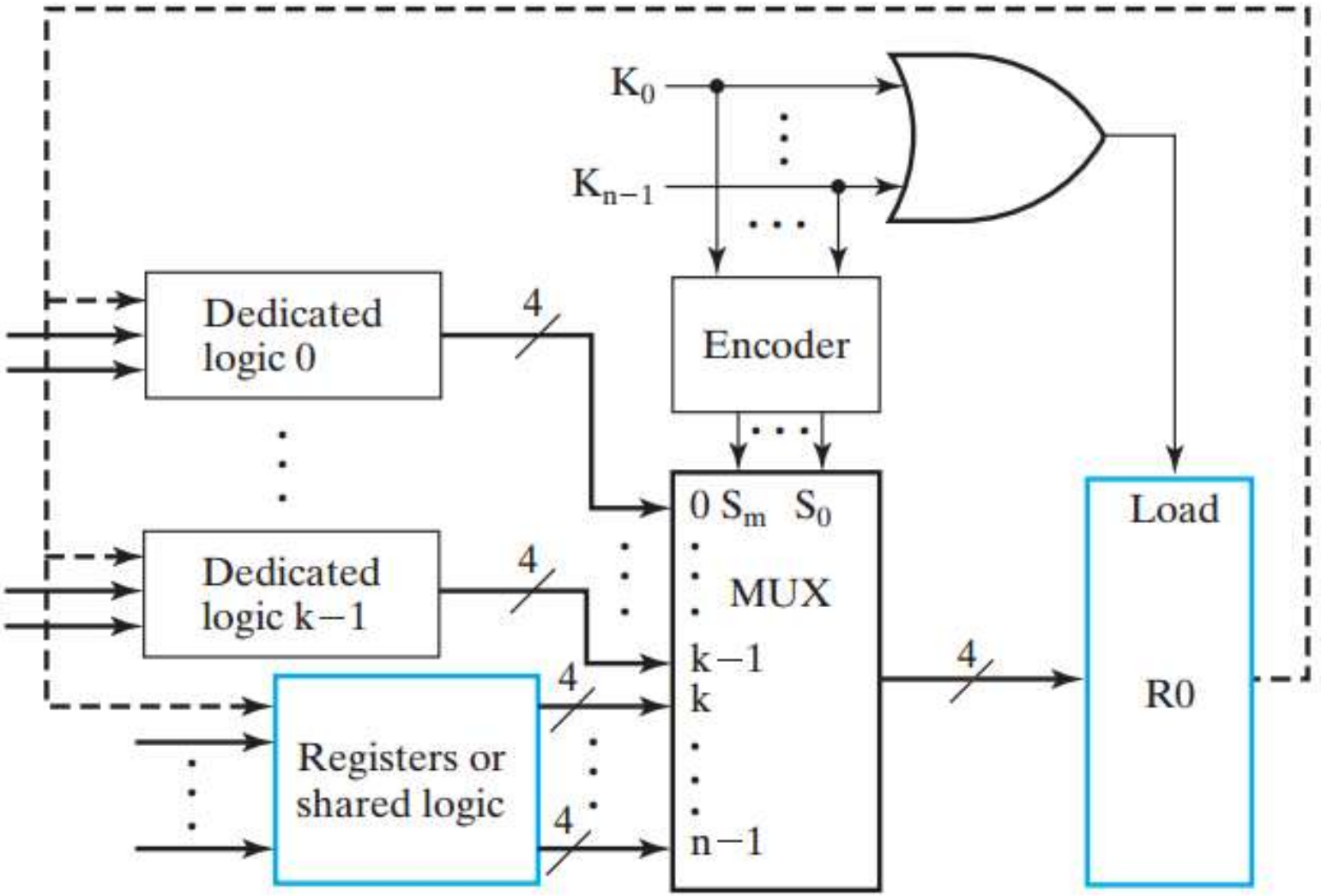
- Multiplexers connected to register inputs produce flexible transfer structures (Note: Clocks are omitted for clarity)
- The transfers are:

$$K1: R0 \leftarrow R1$$

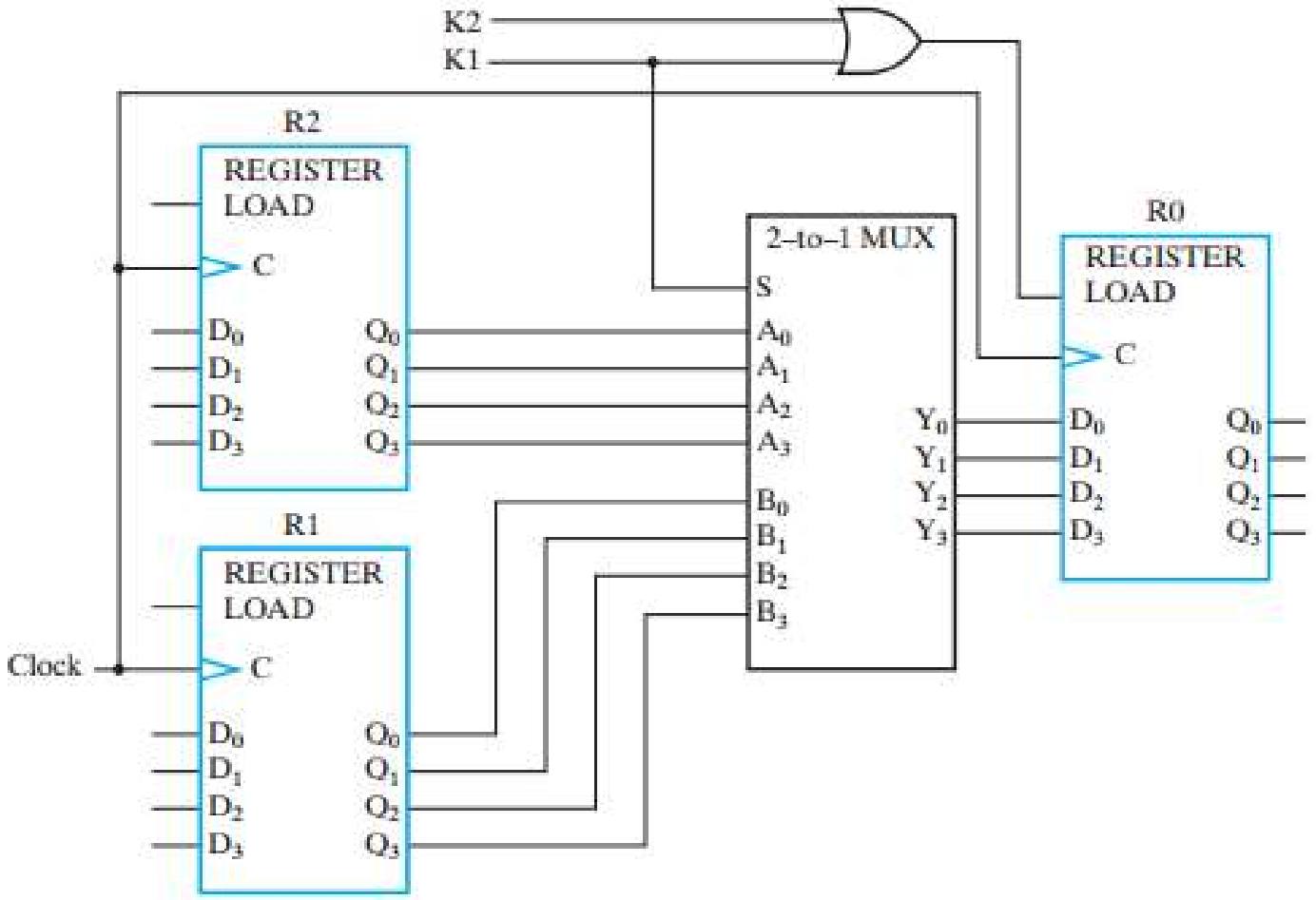
$$\overline{K1} \cdot K2: R0 \leftarrow R2$$



Multiplexer-Based Transfers

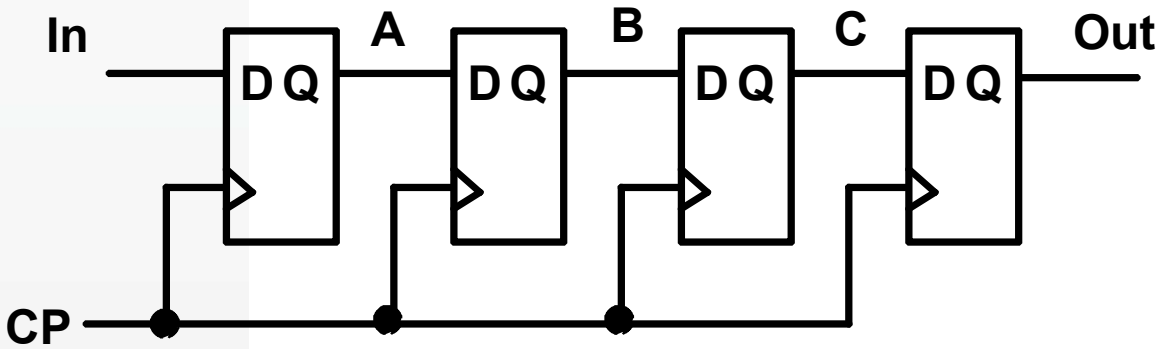


Multiplexer-Based Transfers



Shift Registers

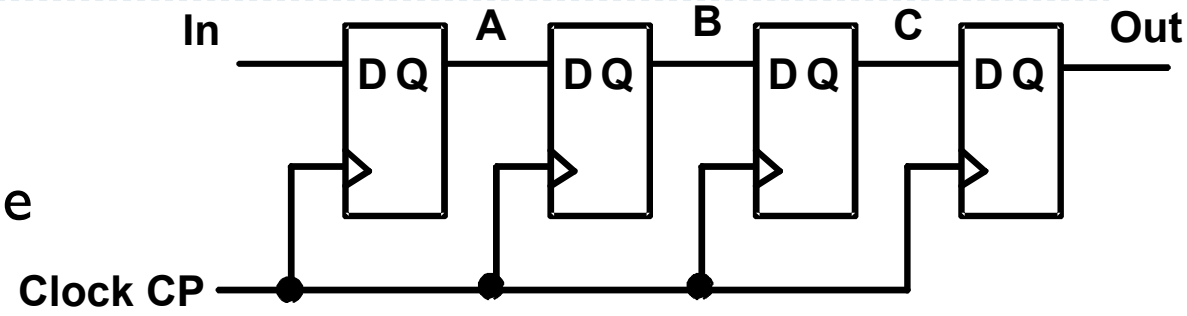
- Shift Registers move data laterally within the register toward its MSB or LSB position
- In the simplest case, the shift register is simply a set of D flip-flops connected in a row like this:



- Data input, In, is called a *serial input* or the *shift right input*.
- Data output, Out, is often called the *serial output*.
- The vector (A, B, C, Out) is called the *parallel output*.

Shift Registers

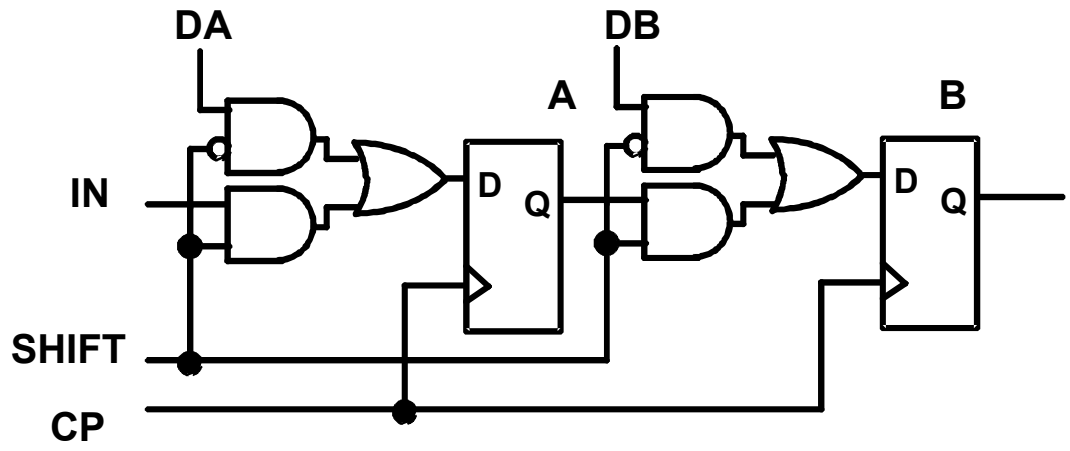
- The behavior of the serial shift register is given in the listing on the lower right
- T0 is the register state just before the first clock pulse occurs
- T1 is after the first pulse and before the second.
- Initially unknown states are denoted by "?"
- Complete the last three rows of the table



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				

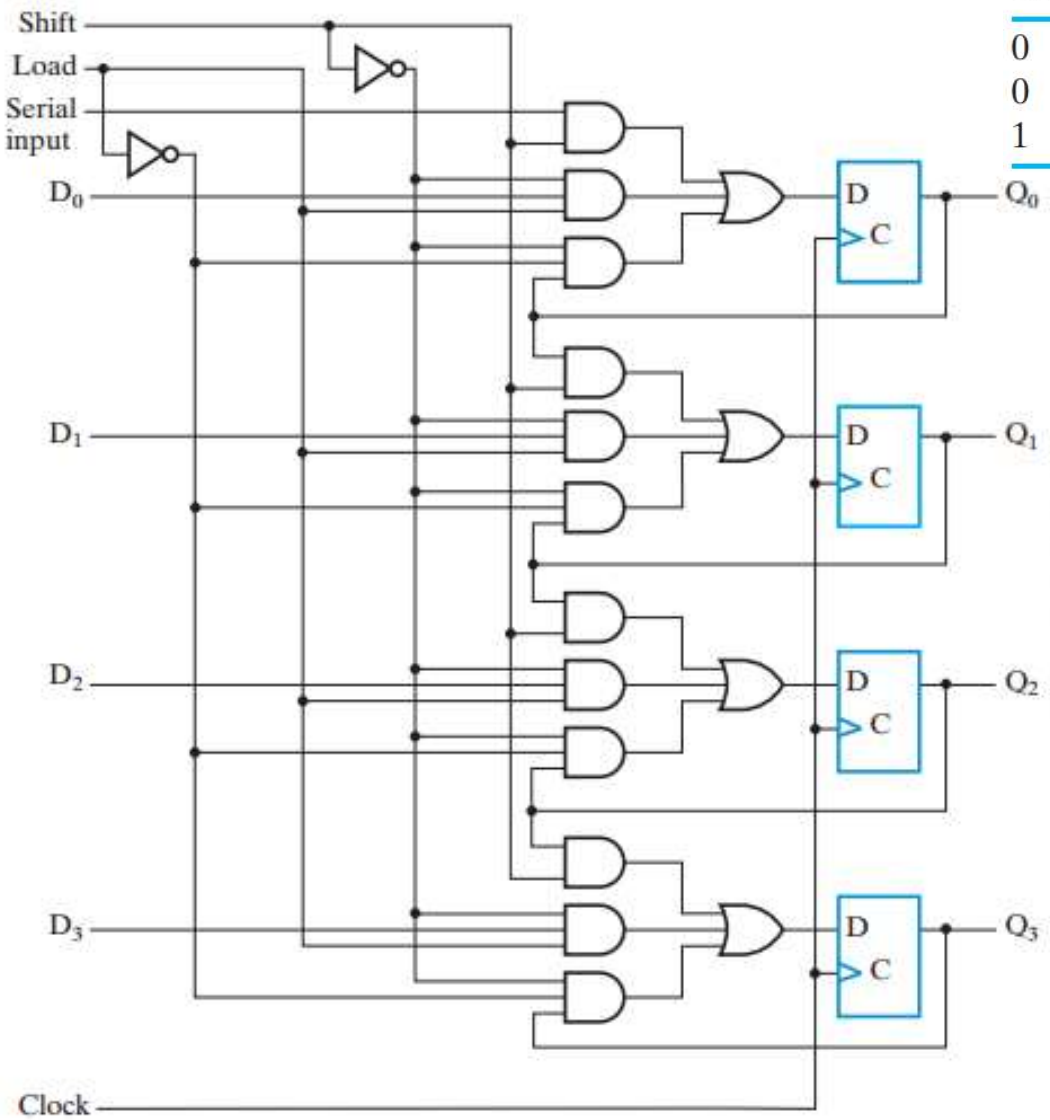
Parallel Load Shift Registers

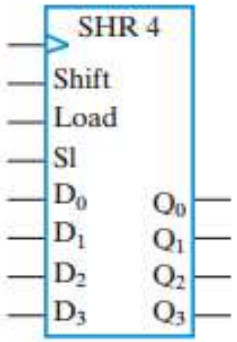
- By adding a mux between each shift register stage, data can be shifted or loaded
- If SHIFT is low, A and B are replaced by the data on D_A and D_B lines, else data shifts right on each clock.
- By adding more bits, we can make *n*-bit parallel load shift registers.



Parallel Load Shift Registers

Shift	Load	Operation
0	0	No change
0	1	Load parallel data
1	X	Shift down from Q_0 to Q_3





(b) Symbol

Shift Registers with Additional Functions

- By placing a 4-input multiplexer in front of each D flip-flop in a shift register, we can implement a circuit with shifts right, shifts left, parallel load, hold.
- Shift registers can also be designed to shift more than a single bit position right or left
- Shift registers can be designed to shift a variable number of bit positions specified by a variable called a *shift amount*.

Registers Summary

- Two basic ways in which information can be entered/outputted
  - Parallel: All 0/1 symbols handled simultaneously. Require as many lines as symbols being transferred.
  - Serial: Involves the symbol-by-symbol availability of information in a time sequence.
- Four possible ways registers can transfer information:
  - Serial-in/serial-out
  - Serial-in/parallel-out
  - Parallel-in/parallel-out
  - Parallel-in/serial-out

Serial-in, Serial-out, Unidirectional Shift Register

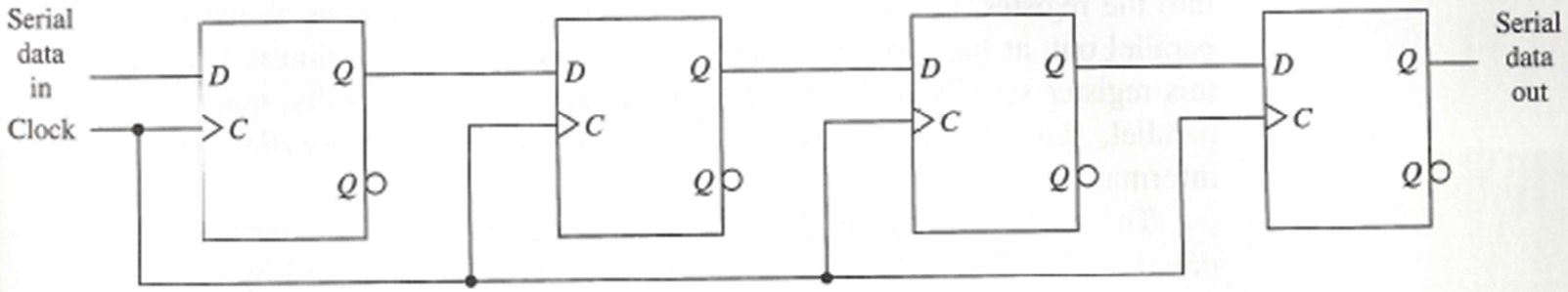


Figure 6.26 Serial-in, serial-out unidirectional shift register.

Serial-in, Parallel-out Unidirectional Shift Register

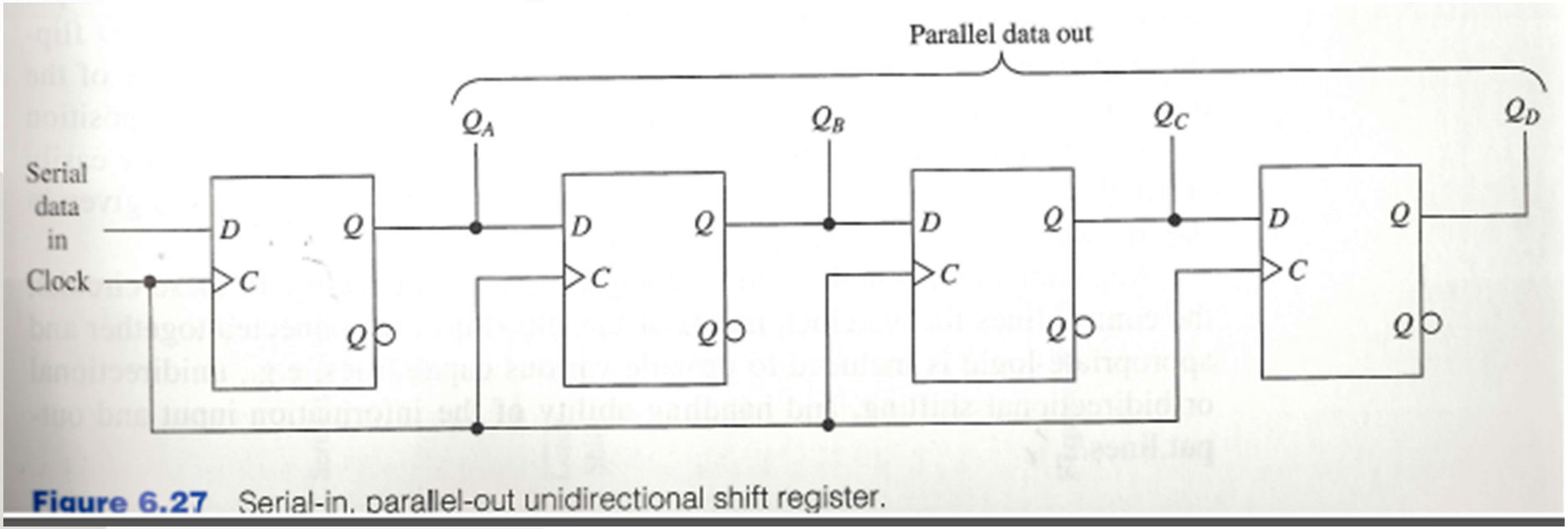


Figure 6.27 Serial-in, parallel-out unidirectional shift register.

Parallel-in, Parallel-out Unidirectional Shift Register

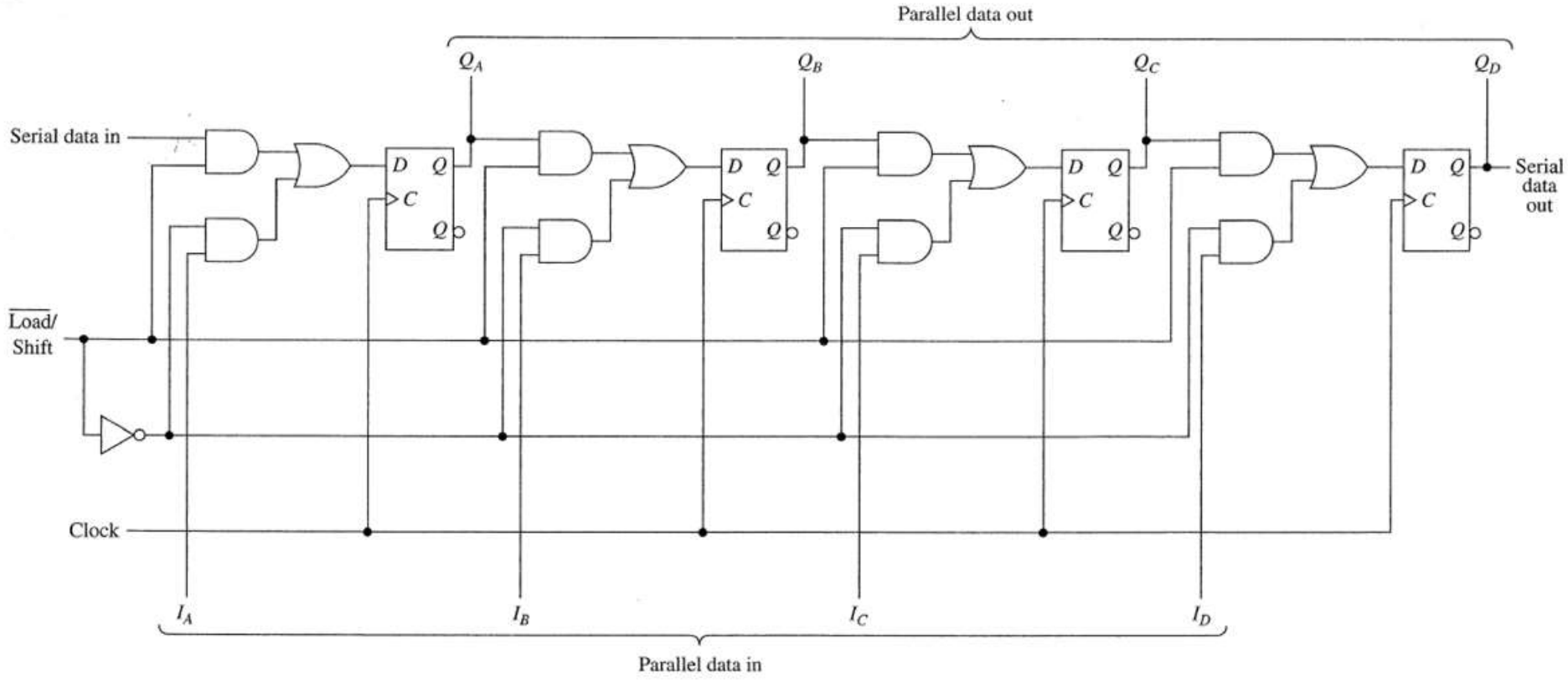
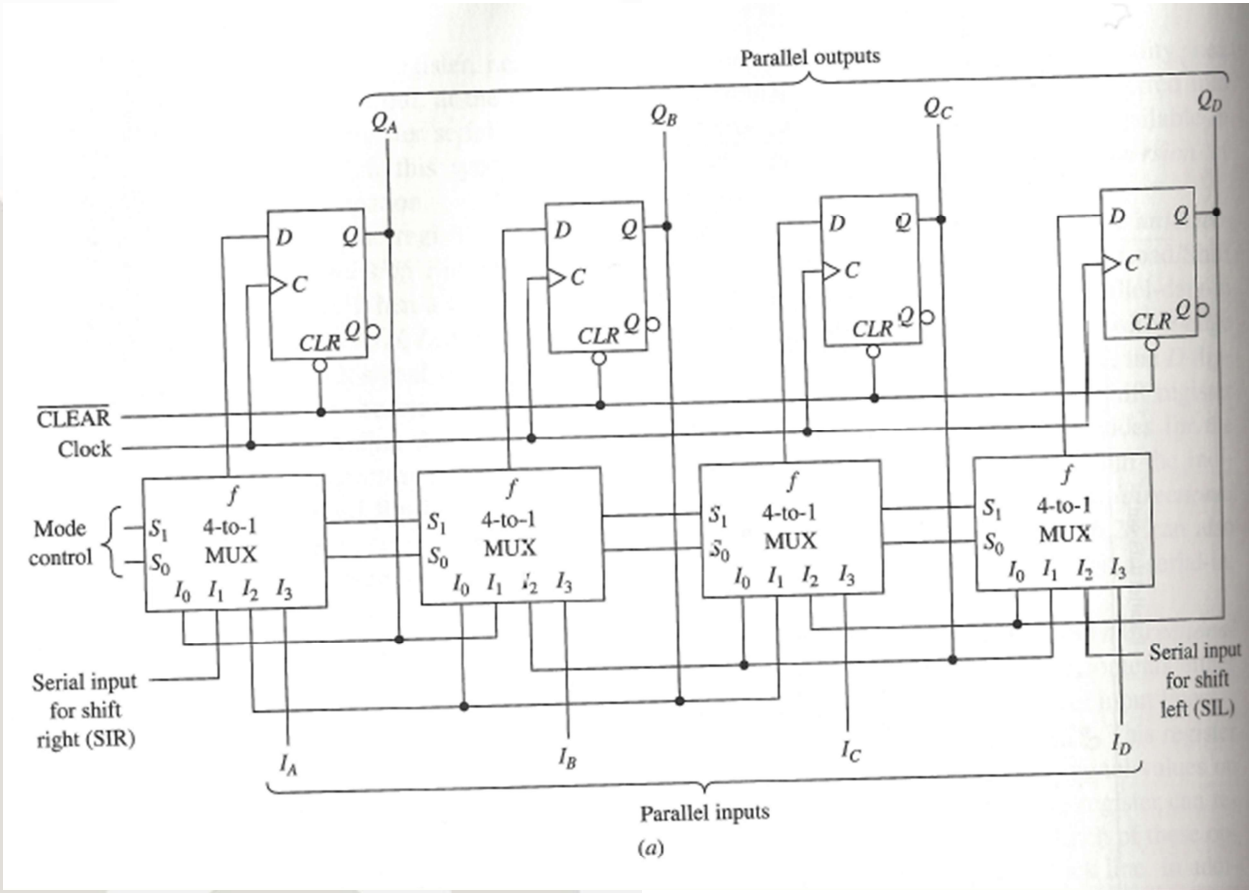
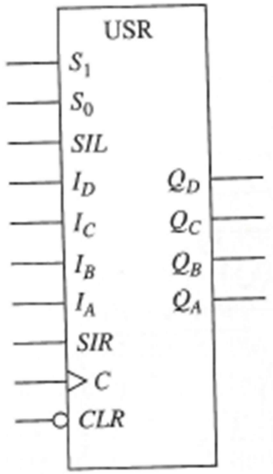


Figure 6.28 Parallel-in unidirectional shift register.

Universal Shift Register





Select lines S_1	S_0	Register operation
0	0	Hold
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Any Questions?