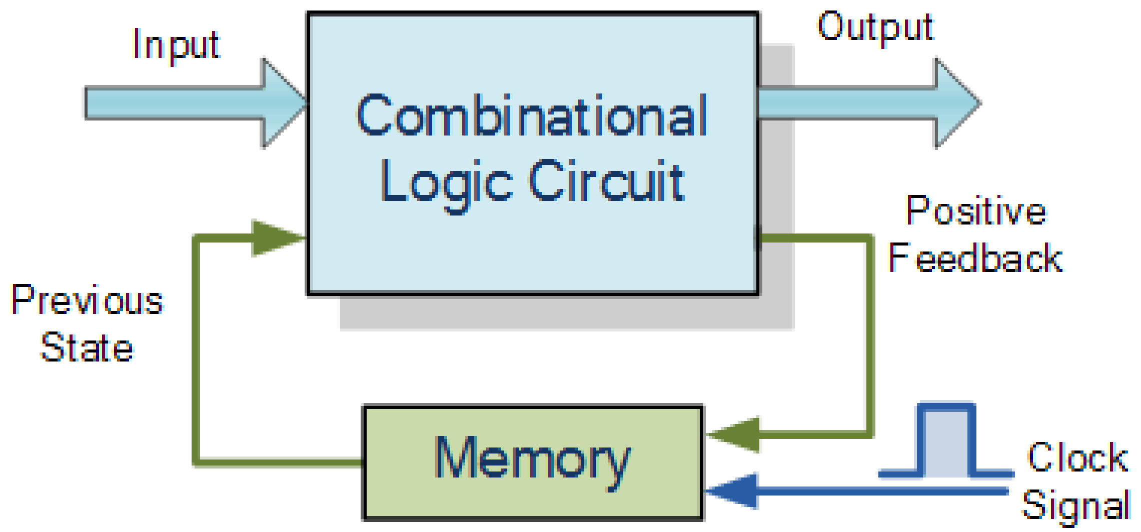
College of Computer and Information Sciences
Department of Computer Science

CSC 220: Computer Organization

Unit 7
Sequential Circuits
(Flip Flop, Registers)

Sequential circuits



- In contrast, the outputs of a sequential circuit depend on not only the inputs, but also the state, or the current contents of some memory.
- This makes things more difficult to understand since the same inputs can yield *different* outputs, depending on what's stored in memory.
- The memory contents can also change as the circuit runs, so the *order* in which things occur makes a difference.

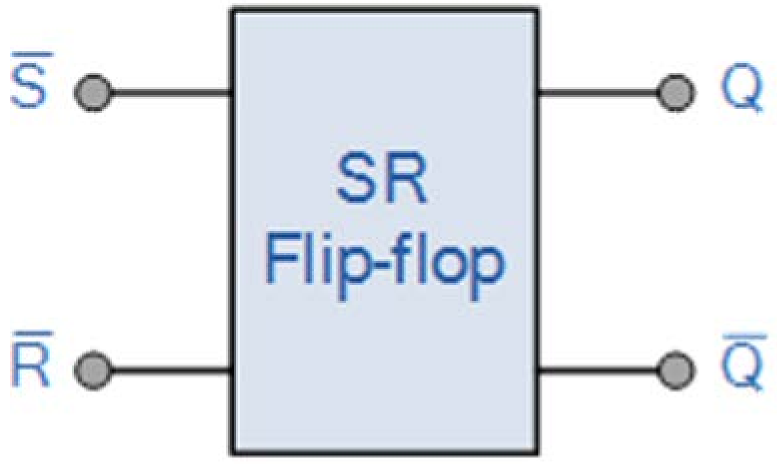
SR Flip-Flop

The **SR flip-flop**, also known as a *SR Latch*, can be considered as one of the most basic sequential logic circuit possible.

This simple flip-flop is basically a one-bit memory bistable device

It has two inputs,

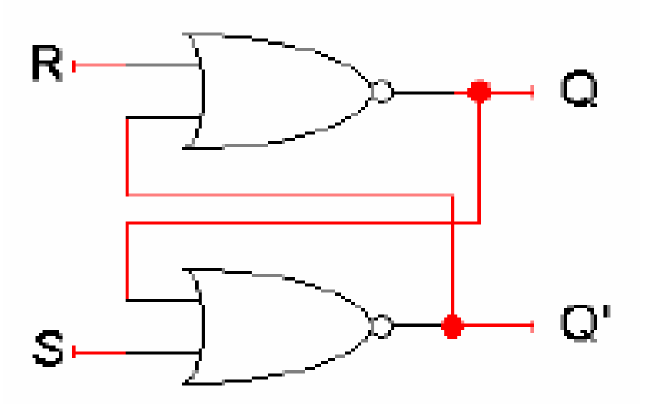
one which will “SET” the device (meaning the output = “1”), and is labelled S and another which will “RESET” the device (meaning the output = “0”), labelled R.



Symbol

A really confusing circuit

- Let's use NOR gates instead of inverters. The SR latch here has two inputs S and R, which will let us control the outputs Q and Q'.



- Q and Q' feed back into the circuit, so they're not only outputs, they're also inputs!
- To figure out how Q and Q' change, we must look at not only the inputs S and R, but also the *current* values of Q and Q'.

$$\boxed{\begin{aligned} Q_{next} &= (R + Q'_{current})' \\ Q'_{next} &= (S + Q_{current})' \end{aligned}}$$

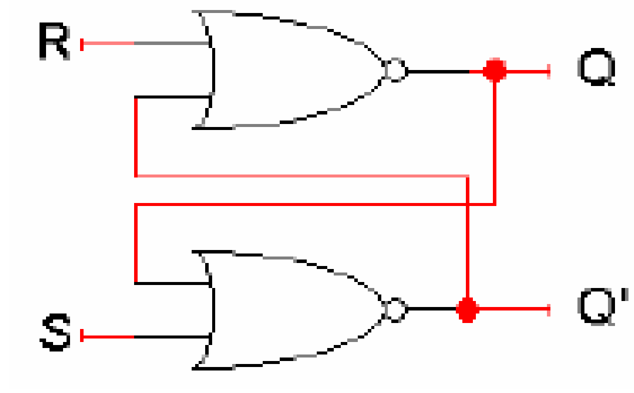
- Let's see how different input values for S and R affect this thing.

Storing a value: SR = 00

- What if S = 0 and R = 0?
- The equations on the right reduce to:

$$Q_{next} = (0 + Q'_{current})' = Q_{current}$$
$$Q'_{next} = (0 + Q_{current})' = Q'_{current}$$

- So when SR = 00, then $Q_{next} = Q_{current}$.
- This is exactly what we need to store values in the latch.



$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

Setting the latch: SR = 10

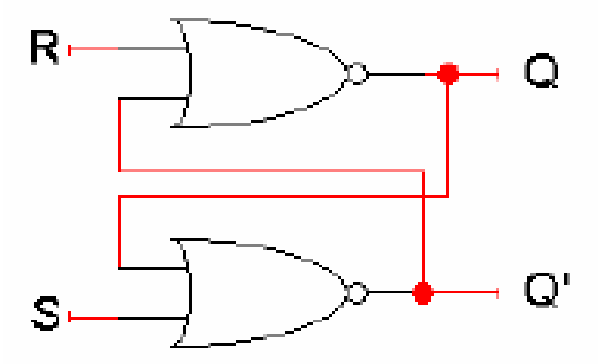
- What if S = 1 and R = 0?
- Since S = 1, Q'_{next} is 0, *regardless* of $Q_{current}$.

$$Q'_{next} = (1 + Q_{current})' = 0$$

- Then this new value of Q' goes into the top NOR gate, along with R = 0.

$$Q_{next} = (0 + 0)' = 1$$

- So when SR = 10, then $Q'_{next} = 0$ and $Q_{next} = 1$. This is how you set the latch to 1; the S input stands for "set."
- Notice it can take up to two steps (two gate delays) from the time S becomes 1 to the time Q_{next} becomes 1.
- But once Q_{next} becomes 1, the outputs will stop changing. This is a stable state.



$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

SR latches are memories!

- This characteristic table shows that our latch provides everything we need in a memory: we can set it, reset it, or keep the current value.

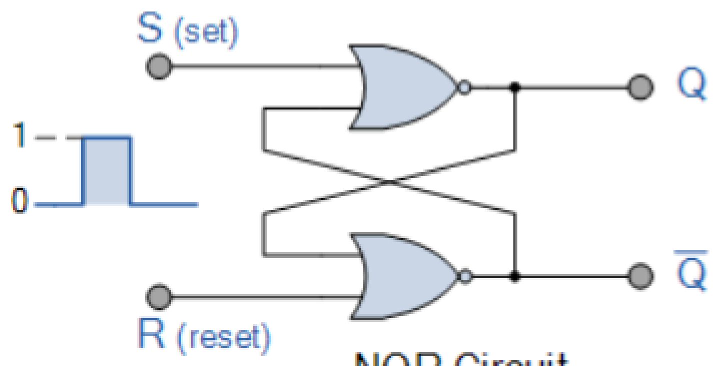
S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

- The output Q represents the data stored in the latch. It is also called the state of the latch.

- We can expand the table above into a state table, which explicitly shows that the *next* values of Q and Q' depend on their *current* values, as well as on the inputs S and R.

Inputs		Current		Next	
S	R	Q	Q'	Q	Q'
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

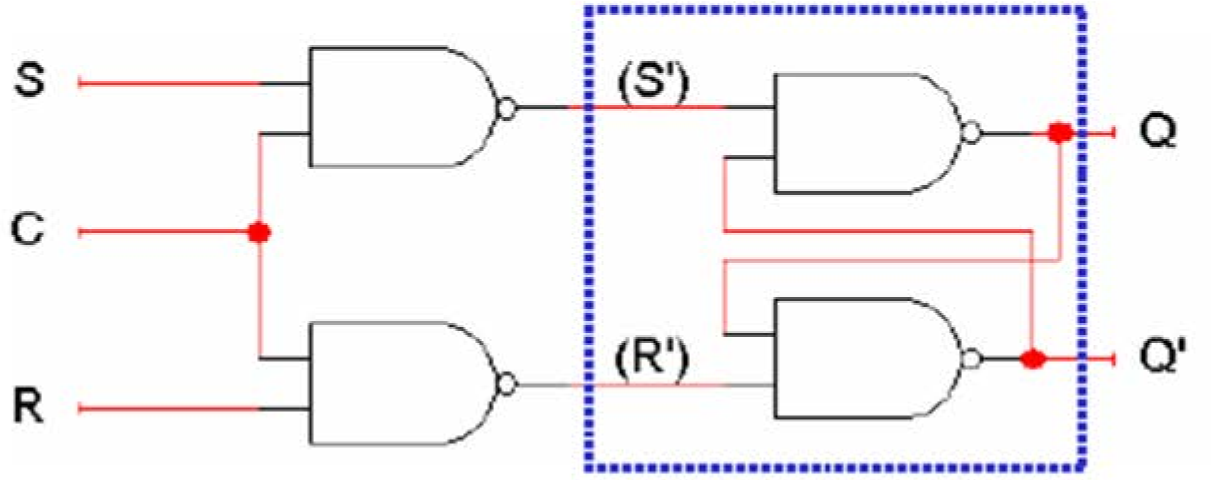
The NOR Gate SR Flip-flop



NOR Circuit

An SR latch with a control input

- Here is an SR latch with a control input C, which acts like an enable.

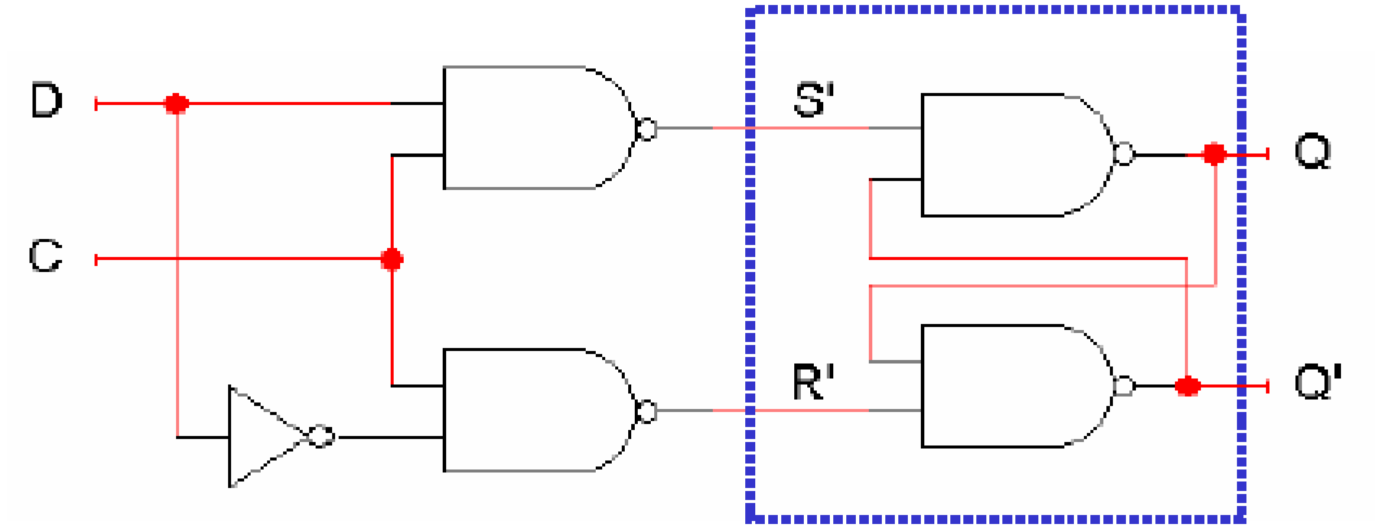


C	S	R	S'	R'	Q
0	x	x	1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	!

- Notice the hierarchical design!
  - The dotted blue box contains the S'R' latch from the previous slide.
  - The additional NAND gates are simply used to generate appropriate inputs for the S'R' latch.
- We'll see more of the control input later today.

D latch

- A D latch is also based on an S'R' latch. The additional gates generate the S' and R' signals, based on inputs D ("data") and C ("control").
  - When C = 0, S' and R' are both 1, so Q does not change.
  - When C = 1, the latch output Q will equal the input D.

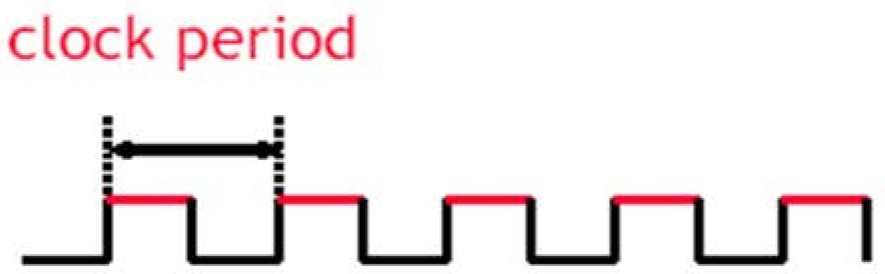


C	D	Q
0	x	No change
1	0	0
1	1	1

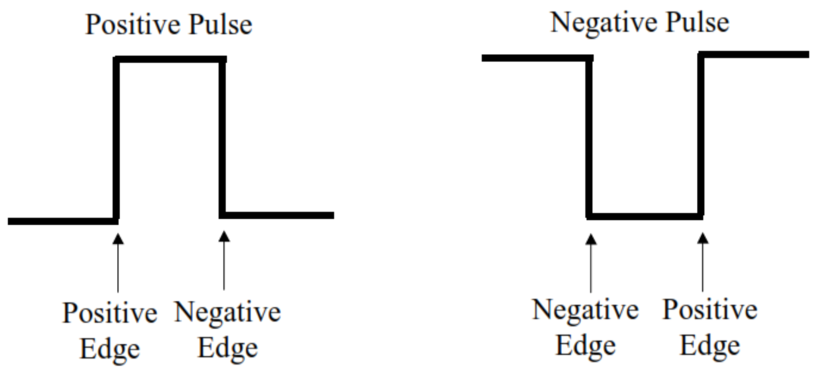
- There are two main advantages of a D latch.
  - No more messing with one input for set and another input for reset!
  - This latch has no "bad" input combinations to avoid. Any of the four possible assignments to C and D are valid.

Clocks and synchronization

- A clock is a special device that continuously outputs 0s and 1s.
  - The time it takes the clock to change from 1 to 0 and back to 1 is called the clock period, or clock cycle time.
  - The clock frequency is the inverse of the clock period. The unit of measurement for frequency is the hertz.

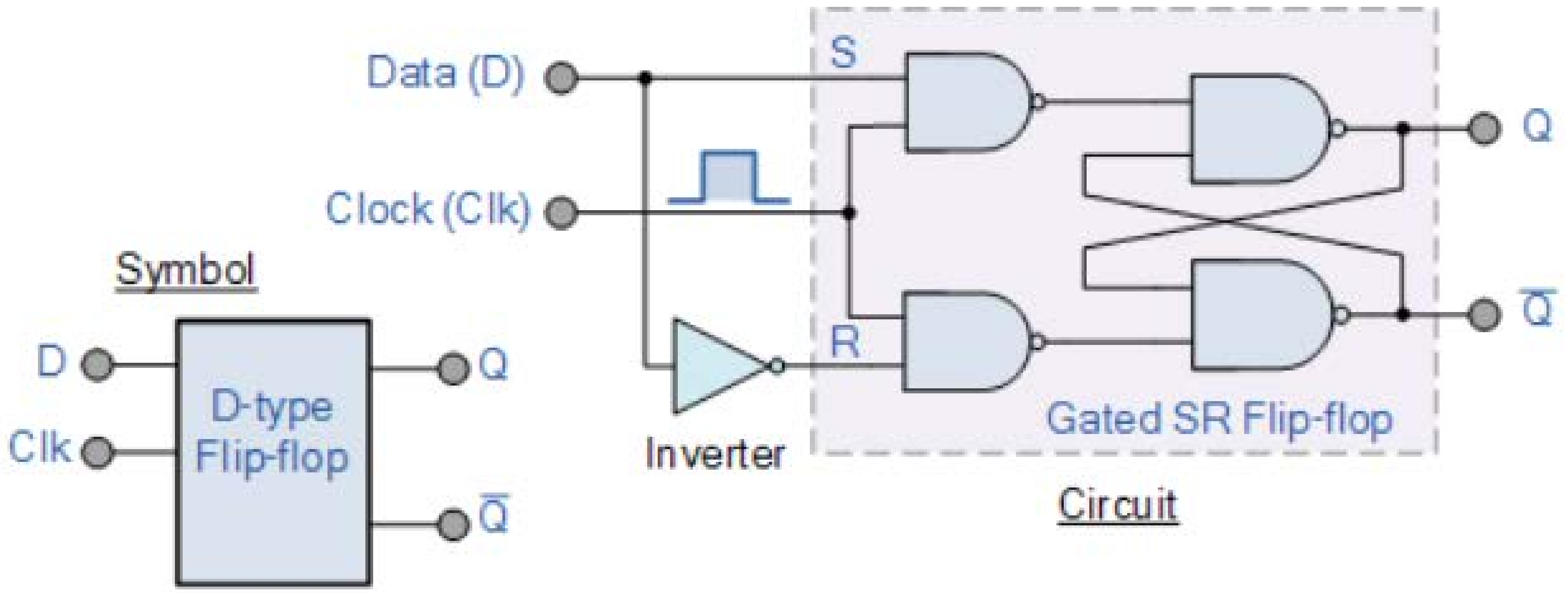


Clock Pulse Definition



- Clocks are often used to synchronize circuits.
  - They generate a repeating, predictable pattern of 0s and 1s that can trigger certain events in a circuit, such as writing to a latch.
  - If several circuits share a common clock signal, they can coordinate their actions with respect to one another.
- This is similar to how humans use real clocks for synchronization.

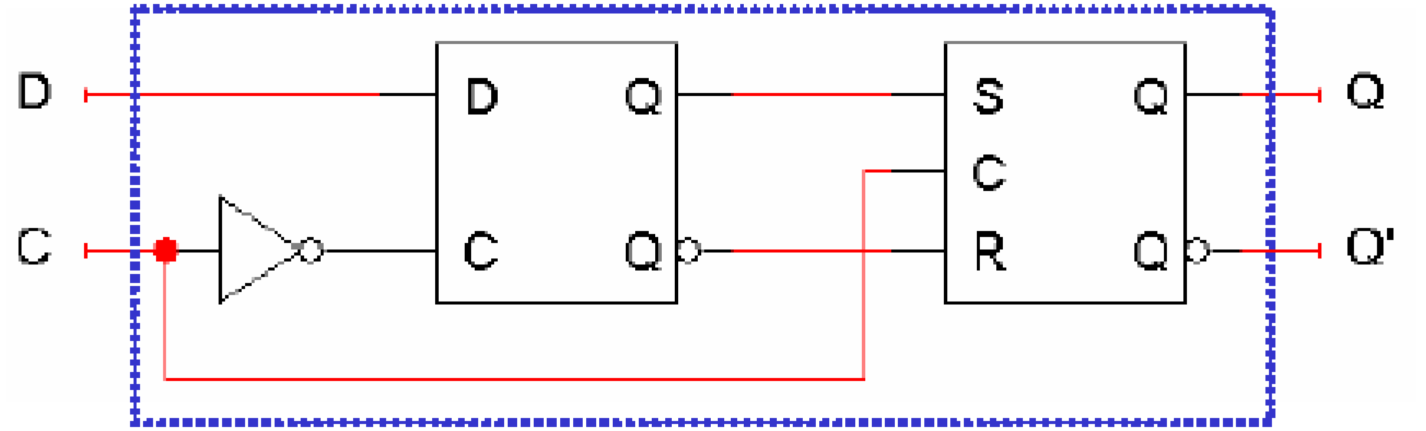
D-type Flip-Flop Circuit

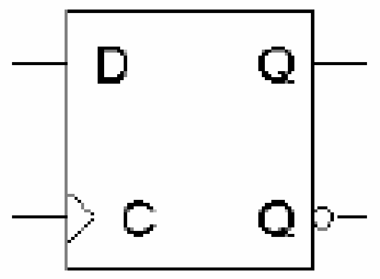


The output of the flip flop would always change on every pulse applied to this data input.

To avoid this an additional input called the “CLOCK” or “ENABLE” input is used to isolate the data input from the flip flop’s latching circuitry after the desired data has been stored. The effect is that D input condition is only copied to the output Q when the clock input is active. This then forms the basis of another sequential device called a **D Flip Flop**.

A positive edge-triggered D flip-flop



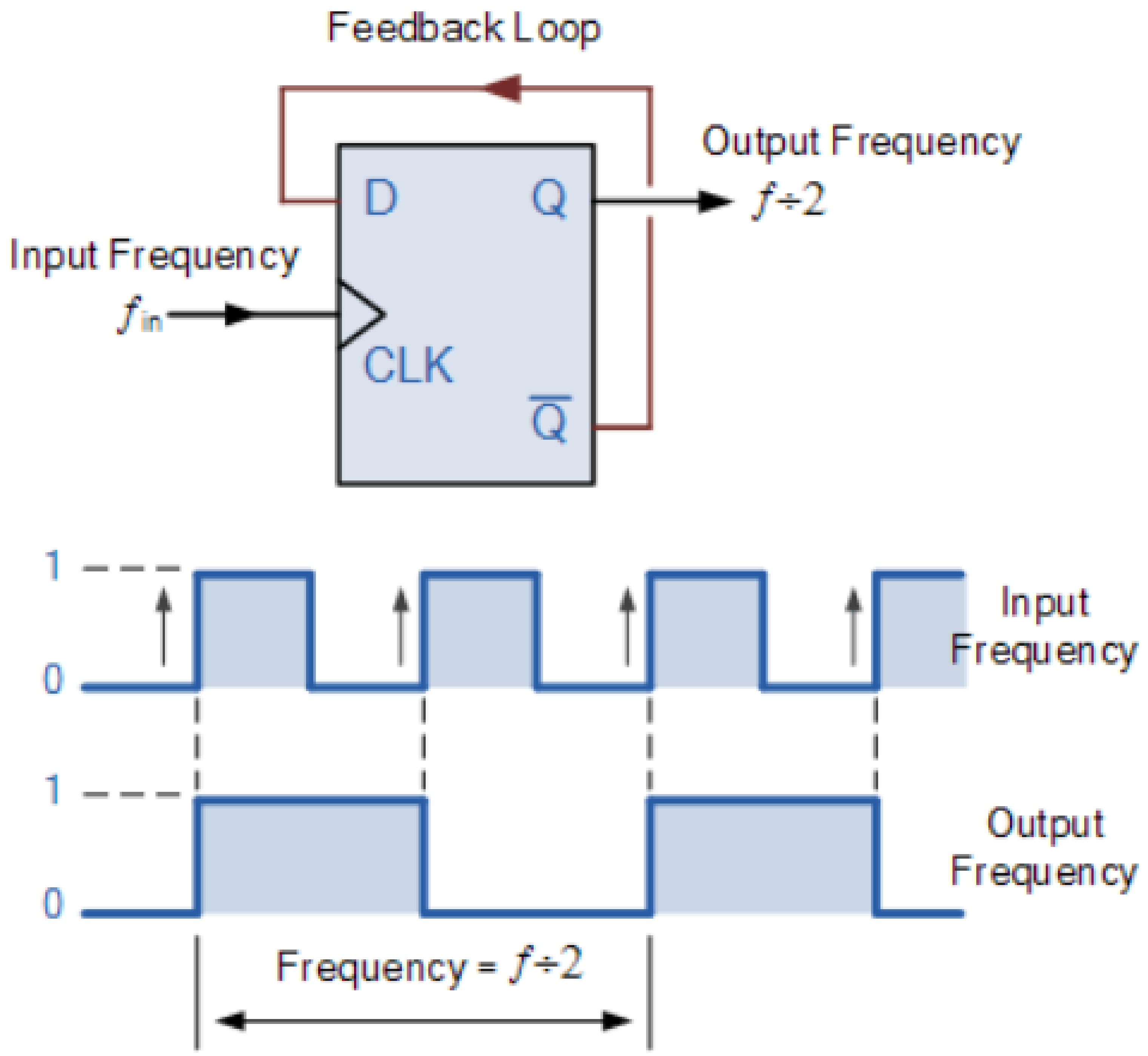


- This positive edge-triggered D flip-flop includes two latches.
  - The flip-flop output Q changes only *after* the positive edge of C.
  - The change is based on the flip-flop input value D that was present at the positive edge of the clock signal.
- A D flip-flop behaves like a D latch except for its positive edge-triggered nature, which is not explicit in the table below.

C	D	Q
0	x	No change
1	0	0 (reset)
1	1	1 (set)

Divide-by-2 Counter

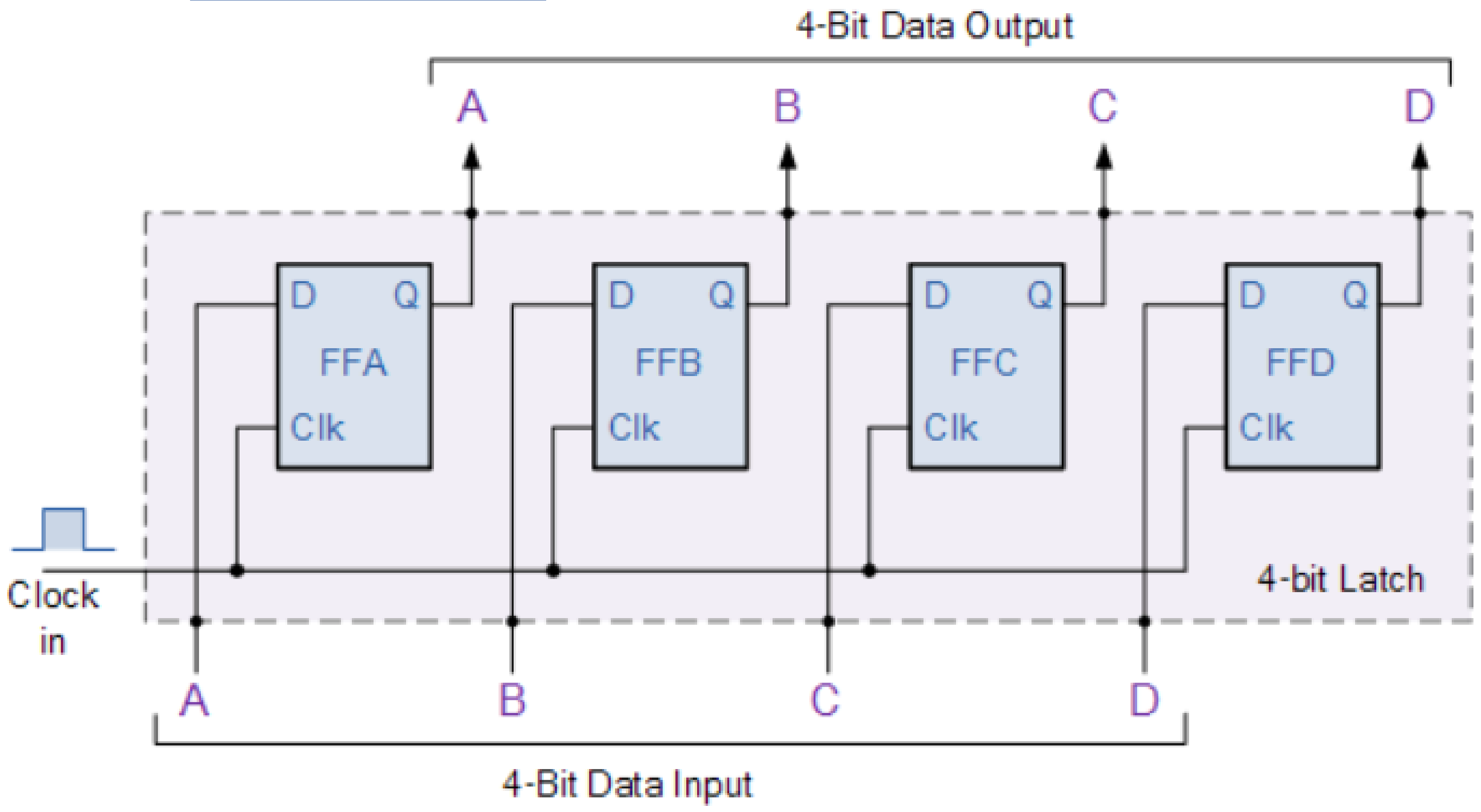
It can be seen from the frequency waveforms above, that by “feeding back” the output from Q to the input terminal D, the output pulses at Q have a frequency that are exactly one half (ƒ/2) that of the input clock frequency, ($ƒ_{IN}$). In other words the circuit produces **frequency division** as it now divides the input frequency by a factor of two (an octave) as Q = 1 once every two clock cycles.



D Flip Flops as Data Latches

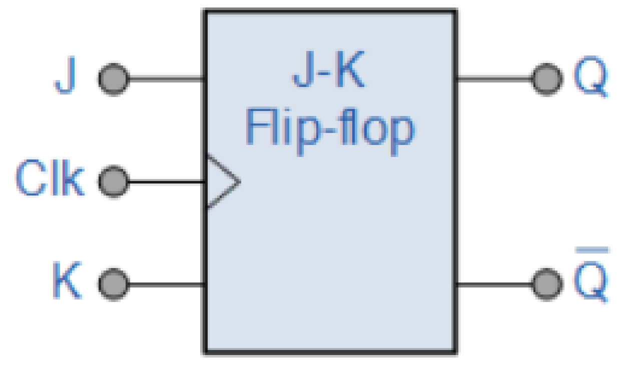
A data latch can be used as a device to hold or remember the data present on its data input, thereby acting a bit like a single bit memory device and IC's such as the TTL 74LS74 or the CMOS 4042 are available in Quad format exactly for this purpose. By connecting together four, *1-bit* data latches so that all their clock inputs are connected together and are "clocked" at the same time, a simple "4-bit" Data latch can be made as shown below.

4-bit Data Latch

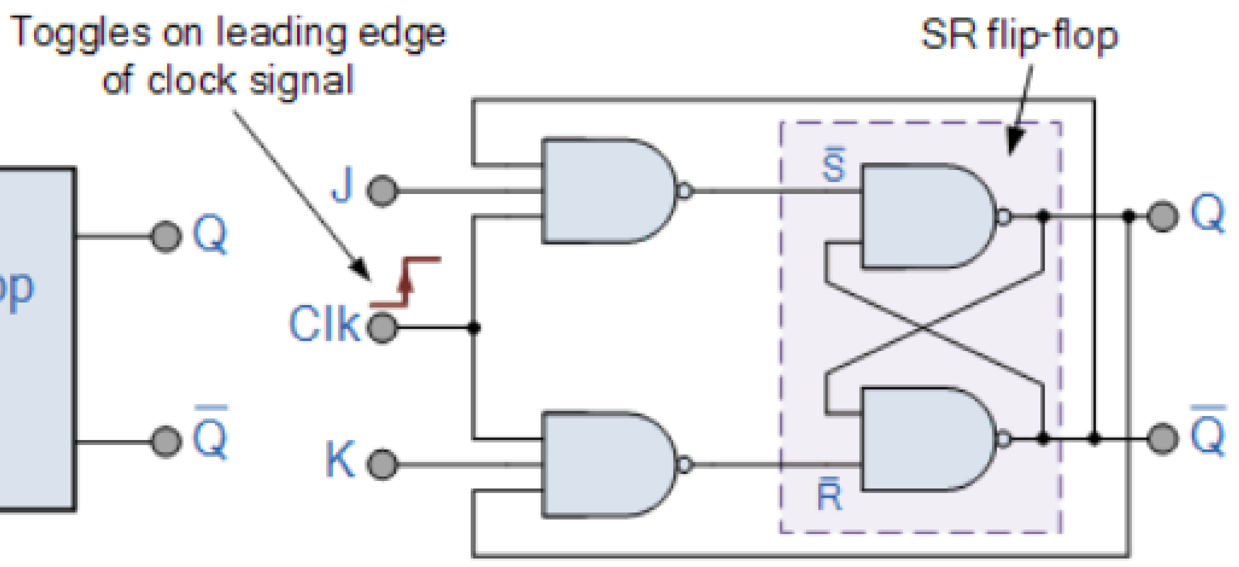


The JK Flip Flop

The Basic JK Flip-flop



Symbol



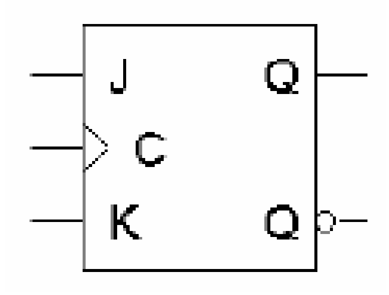
Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack Kilby. Then this equates to: J = S and K = R.

The Truth Table for the JK Function

	Input		Output		Description
	J	K	Q	Q_{next}	
same as for the SR Latch	0	0	0	0	Memory no change
	0	0	1	1	
	0	1	1	0	Reset Q » 0
	0	1	0	0	
	1	0	0	1	Set Q » 1
	1	0	1	1	
toggle action	1	1	0	1	Toggle
	1	1	1	0	

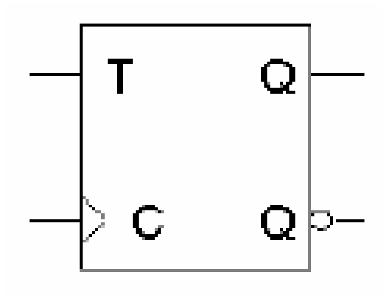
Flip-flop variations

- We can make different versions of flip-flops based on the D flip-flop, just like we made different latches based on the S'R' latch.
- The JK flip-flop has inputs that act like S and R, but JK = 11 *complements* the flip-flop's current state.



C	J	K	Q_{next}
0	x	x	No change
1	0	0	No change
1	0	1	0 (reset)
1	1	0	1 (set)
1	1	1	$Q'_{current}$

- A T flip-flop can only maintain or complement its current state.



C	T	Q_{next}
0	x	No change
1	0	No change
1	1	$Q'_{current}$

Characteristic tables

- The tables that we've made so far are called characteristic tables.
  - They show the next state Q(t+1) in terms of the current state Q(t) and the inputs.
  - For simplicity, the control input C is usually not listed.
  - Again, these tables don't indicate the positive edge-triggered nature of the flip-flops.

D	Q(t+1)	Operation
0	0	Reset
1	1	Set

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

T	Q(t+1)	Operation
0	Q(t)	No change
1	Q'(t)	Complement

Characteristic equations

- We can also write characteristic equations, where the next state Q(t+1) is defined in terms of the current state Q(t) and the flip-flop inputs.

D	Q(t+1)	Operation
0	0	Reset
1	1	Set

$Q(t+1) = D$

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

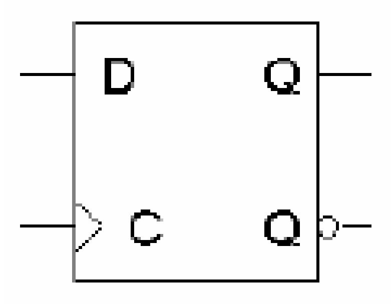
$Q(t+1) = K'Q(t) + JQ'(t)$

T	Q(t+1)	Operation
0	Q(t)	No change
1	Q'(t)	Complement

$$\begin{aligned} Q(t+1) &= T'Q(t) + TQ'(t) \\ &= T \oplus Q(t) \end{aligned}$$

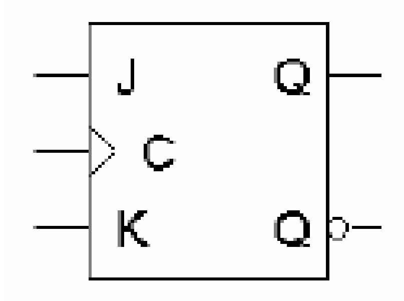
Flip-flop review

Flip-flops | Characteristic tables | Characteristic equations



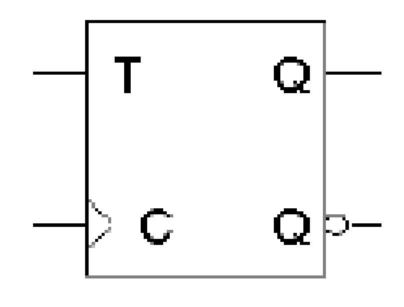
D	Q(t+1)	Operation
0	0	Reset
1	1	Set

$Q(t+1) = D$



J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

$Q(t+1) = K'Q(t) + JQ'(t)$

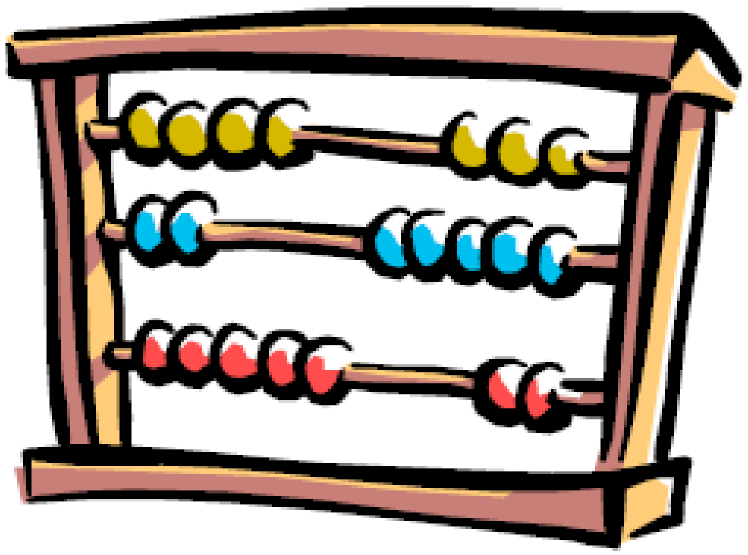


T	Q(t+1)	Operation
0	Q(t)	No change
1	Q'(t)	Complement

$Q(t+1) = T \oplus Q(t)$

Regisrers

Registers and counters

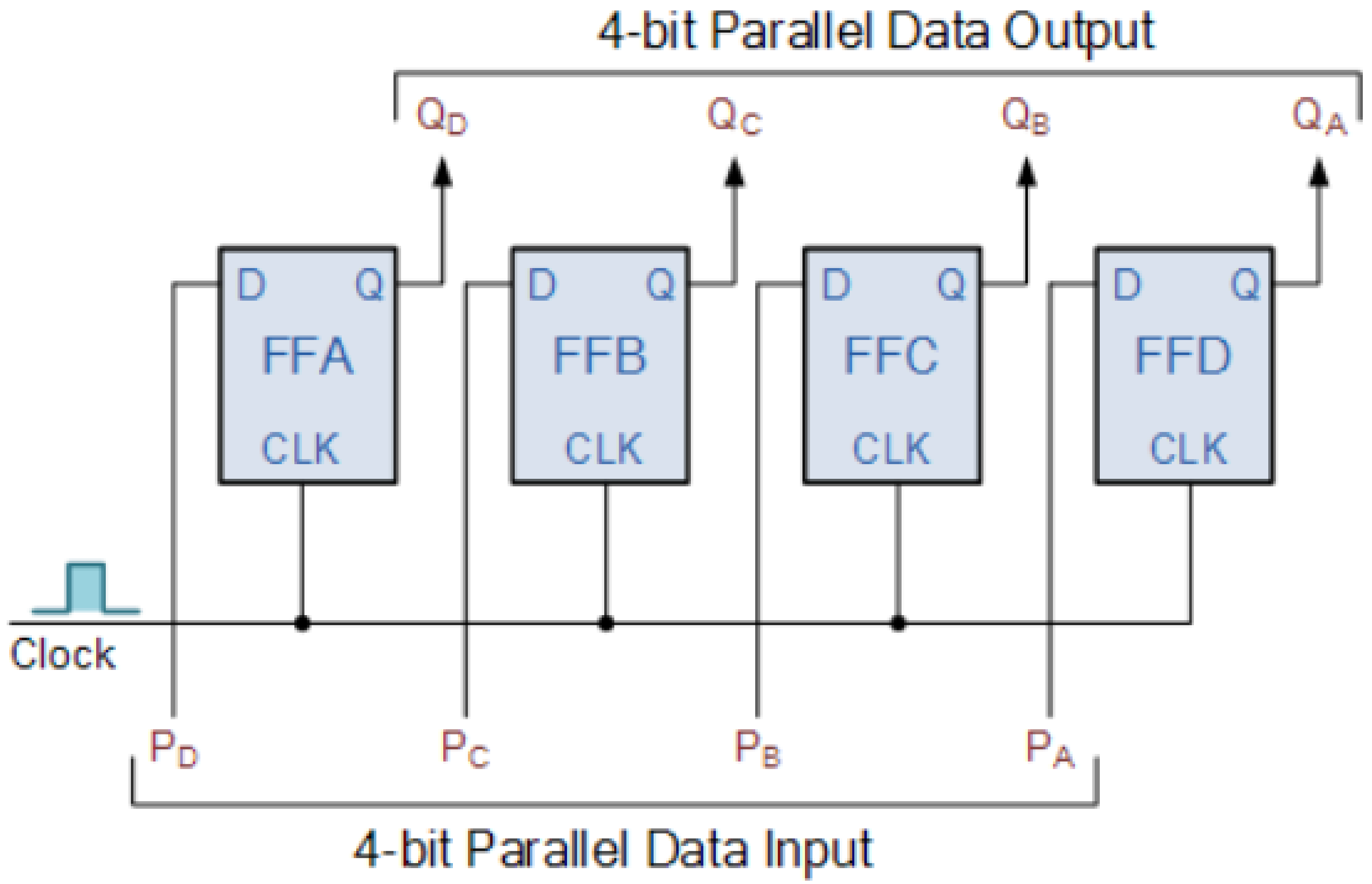


- Today we'll see two common sequential devices, registers and counters.
  - First we'll study some different kinds of registers and discuss how to build them. Several example circuits are also shown.
  - Then we'll talk about counters in more detail, looking at both some implementations and applications.
- These are not only examples of sequential analysis and design, but also real devices used in larger circuits, as we'll see in the coming weeks.

Registers

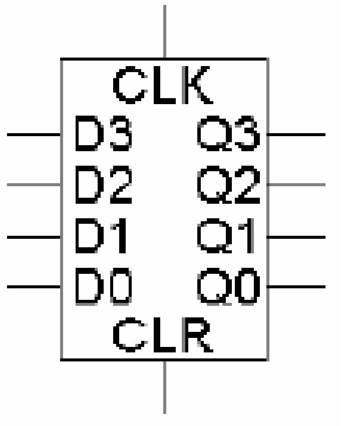
- Flip-flops are limited because they can store only one bit.
  - We had to use two flip-flops for most of our examples so far.
  - Most computers work with integers and single-precision floating-point numbers that are 32-bits long.
- A register is an extension of a flip-flop that can store multiple bits.
- Registers are commonly used as temporary storage in a processor.
  - They are faster and more convenient than main memory.
  - More registers can help speed up complex calculations.
- Later we'll learn more about how registers are used in processors, and some of the differences between registers and random-access memories or RAM.

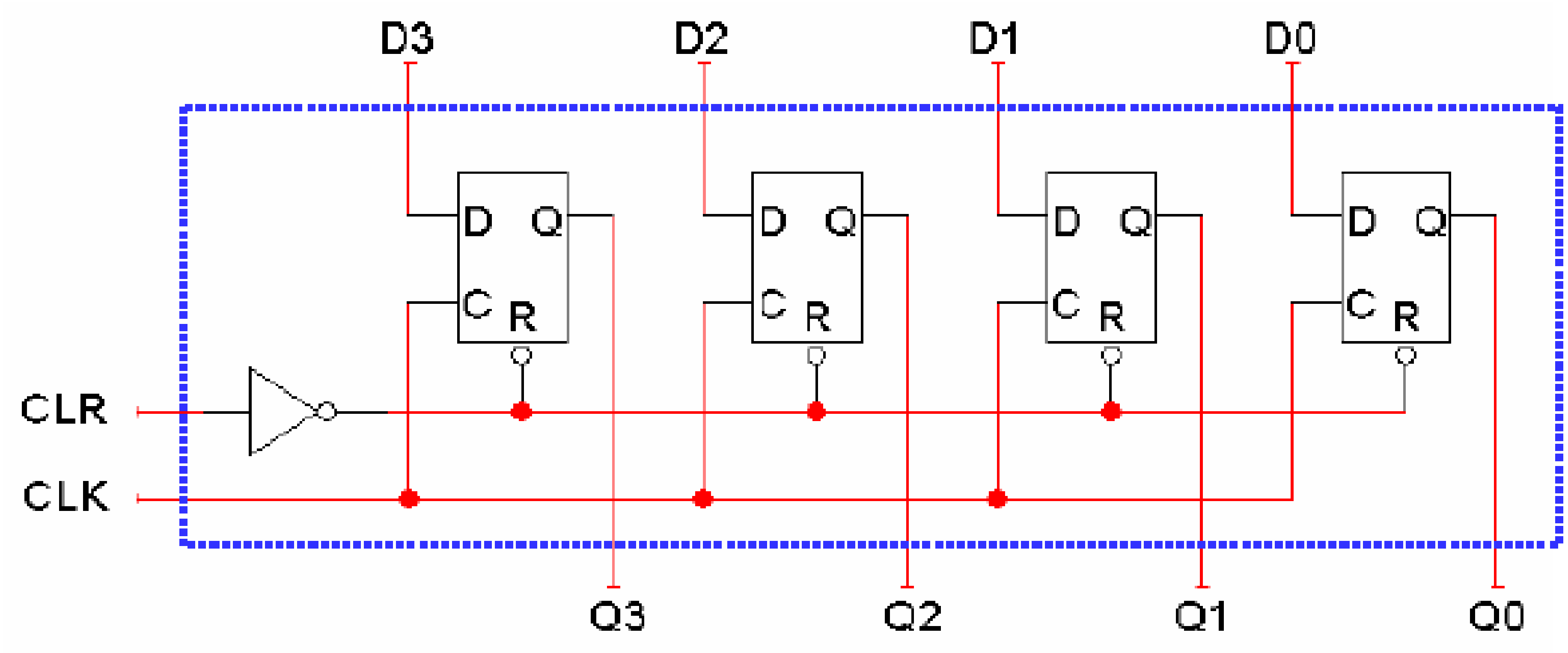
4-bit Parallel-in to Parallel-out Shift Register



A basic register

- Basic registers are easy to build. We can store multiple bits just by putting a bunch of flip-flops together!
- A 4-bit register from LogicWorks, Reg-4, is on the right, and its internal implementation is below.
  - This register uses D flip-flops, so it's easy to store data without worrying about flip-flop input equations.
  - All the flip-flops share a common CLK and CLR signal.

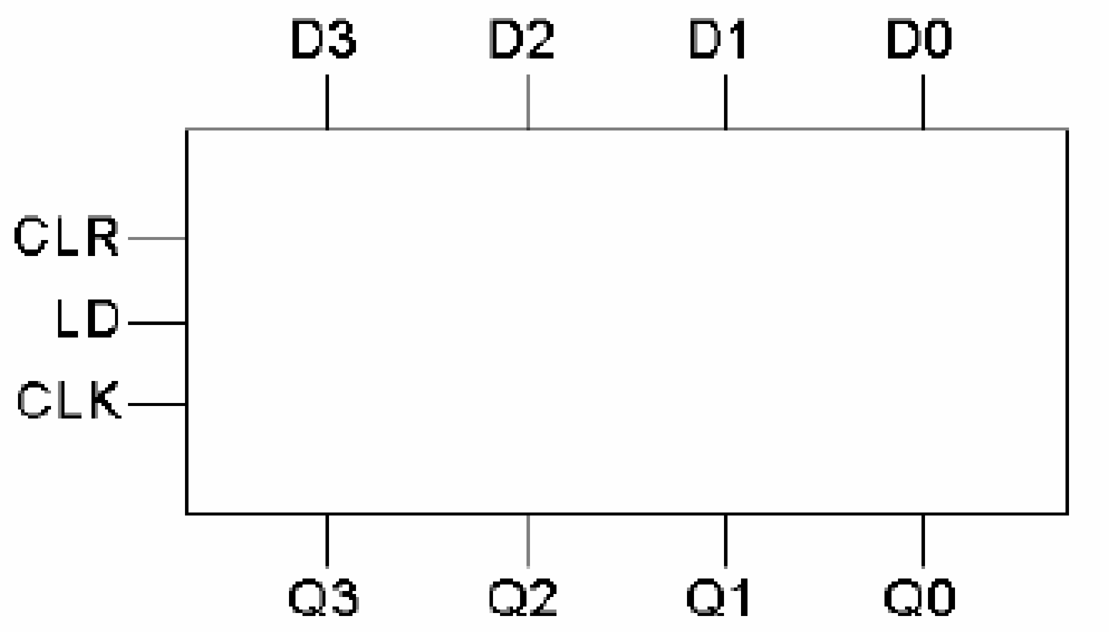


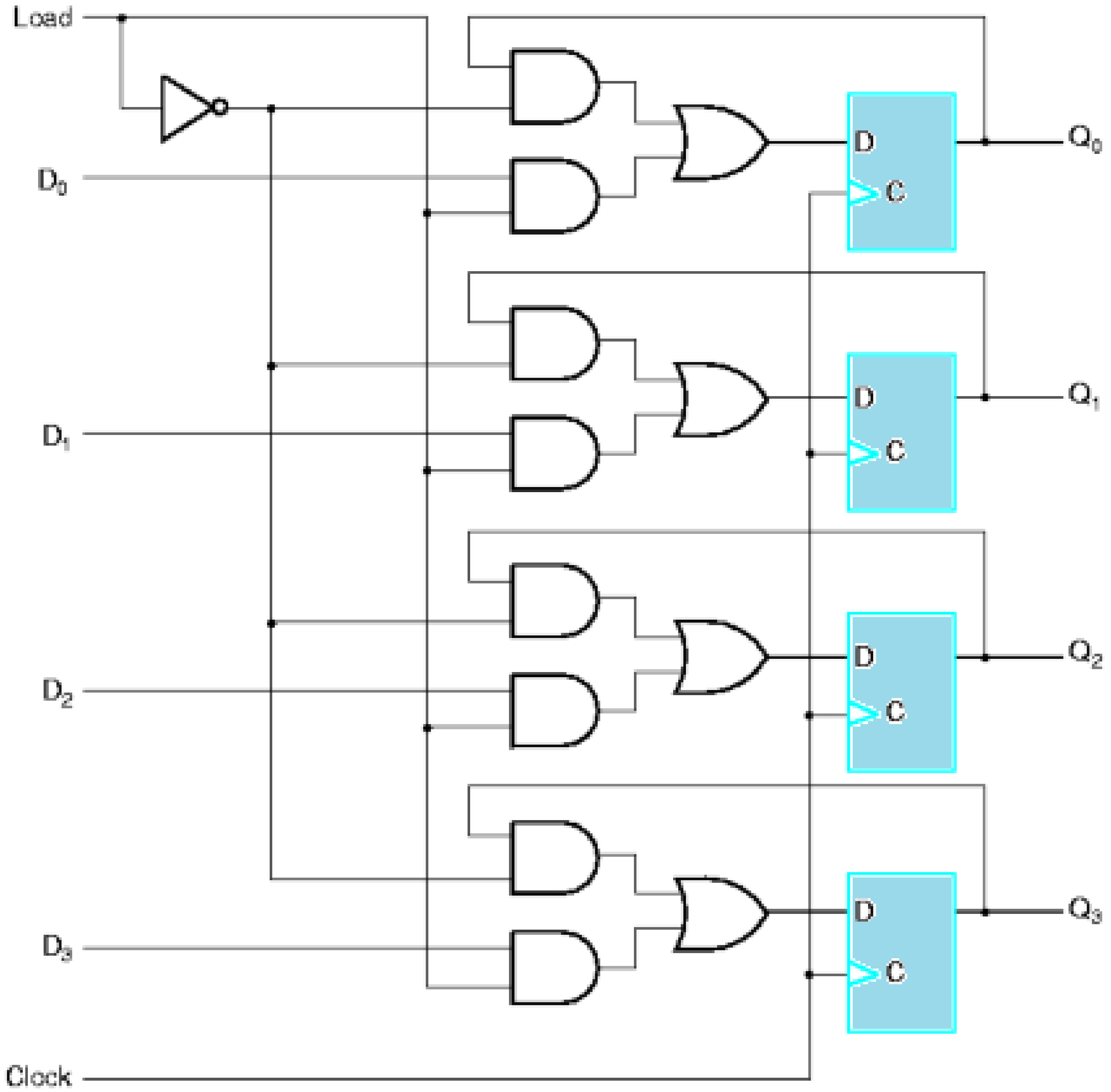


Adding another operation

- The input D3-D0 is copied to the output Q3-Q0 on *every* clock cycle.
- How can we store the current value for more than one cycle?
- Let's try to add a load input signal LD to the register.
  - If LD = 0, the register keeps its current contents.
  - If LD = 1, the register stores a new value, taken from inputs D3-D0.

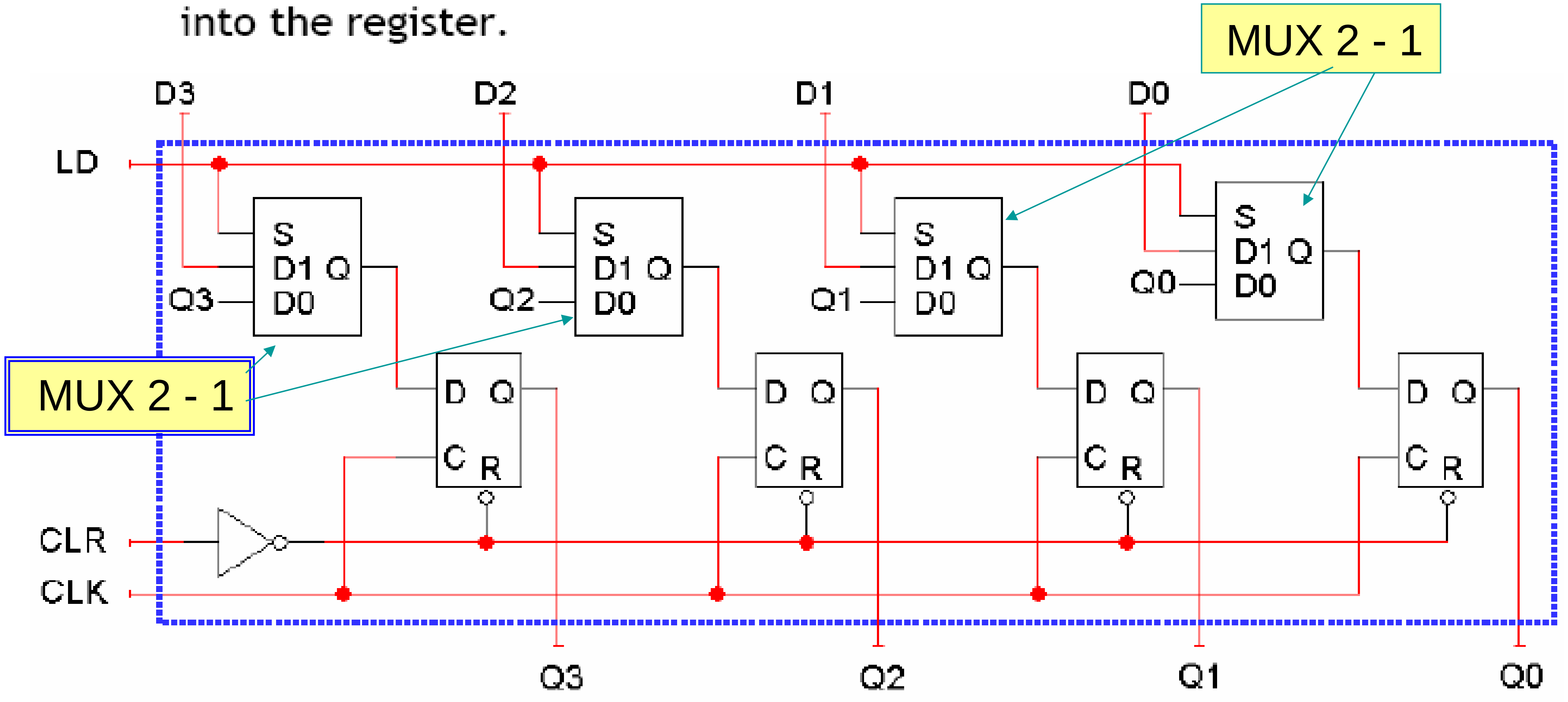
LD	Q(t+1)
0	Q(t)
1	D_3-D_0




Load
D_0
D_1
D_2
D_3
Clock
D
C
Q_0
Q_1
Q_2
Q_3

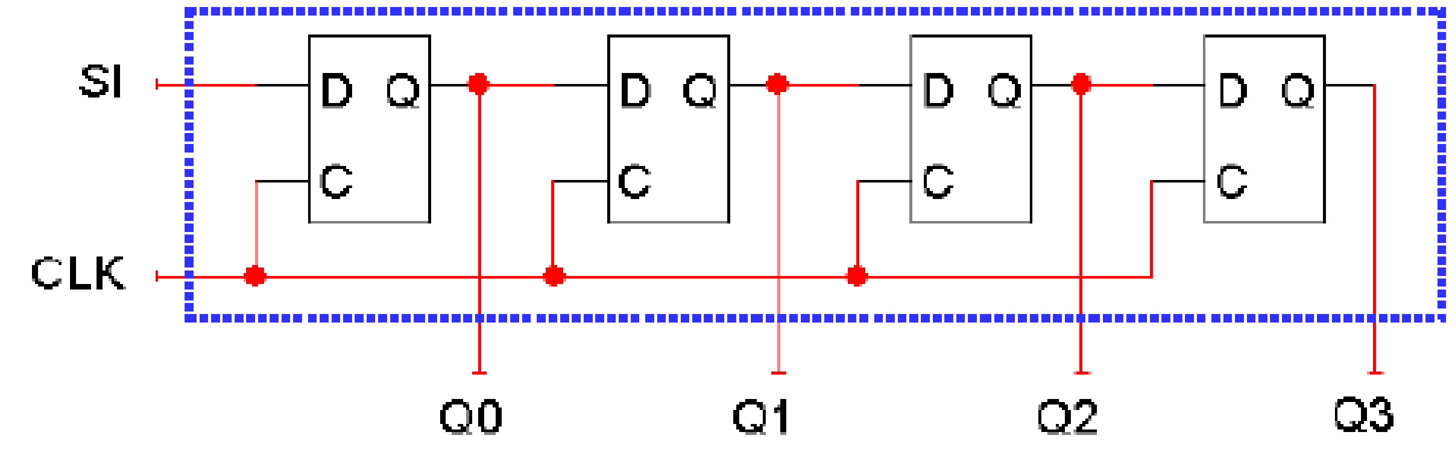
A better parallel load

- Another idea is to modify the flip-flop D inputs and not the clock signal.
  - When LD = 0 the flip-flop inputs are Q3-Q0, so each flip-flop keeps its current value.
  - When LD = 1 the flip-flop inputs are D3-D0, so this new value is loaded into the register.

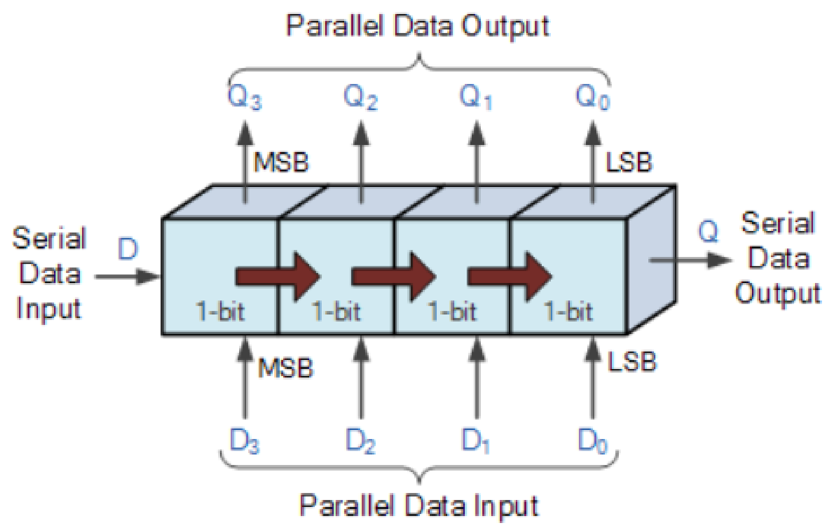


Shift registers

- A shift register "shifts" its output once every clock cycle. SI is an input that supplies a new bit to shift "into" the register.



$$Q0(t+1) = SI$$
$$Q1(t+1) = Q0(t)$$
$$Q2(t+1) = Q1(t)$$
$$Q3(t+1) = Q2(t)$$



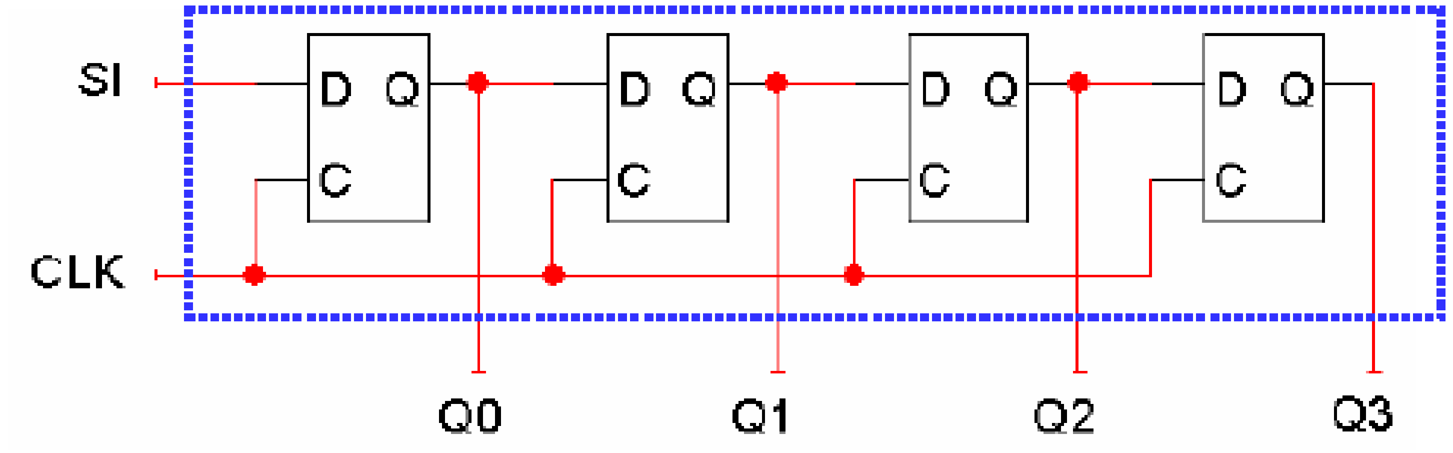
- Here is one example transition.

Present State Q0-Q3	Input SI	Next State Q0-Q3
0110	1	1011

- The current Q3 (0 in this example) will be lost on the next cycle.

Serial-in to Serial-out (SISO) Shift Register

Shift direction



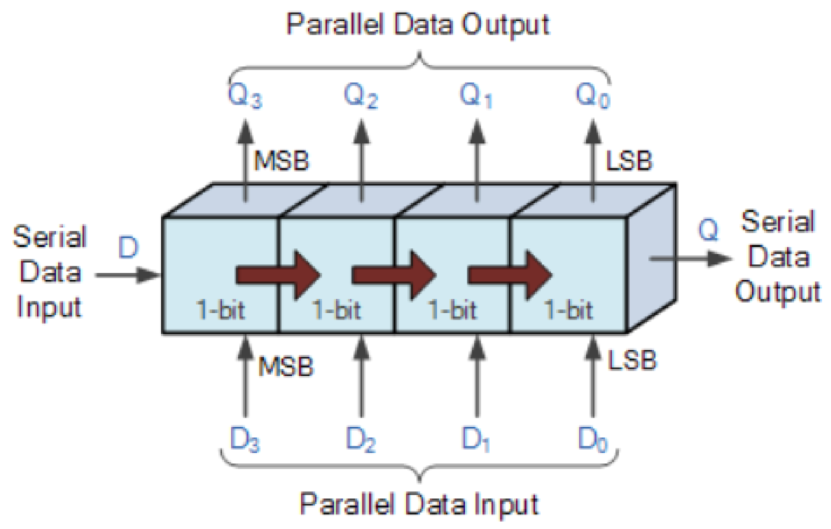
Q0(t+1) = SI
Q1(t+1) = Q0(t)
Q2(t+1) = Q1(t)
Q3(t+1) = Q2(t)

- The circuit and example make it look like the register shifts "right."

Present Q0-Q3	SI	Next Q0-Q3
ABCD	X	XABC

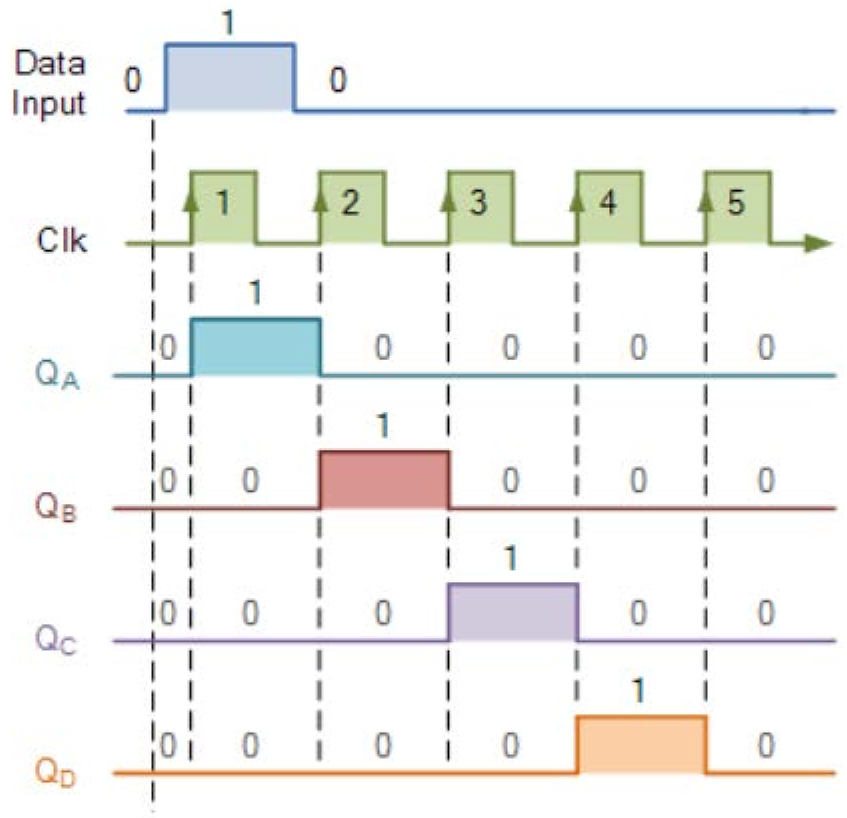
- But it all depends on your interpretation of the bits. If you regard Q3 as the most significant bit, then the register appears to shift in the *opposite* direction!

Present Q3-Q0	SI	Next Q3-Q0
DCBA	X	CBAX



Basic Data Movement Through A Shift Register

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

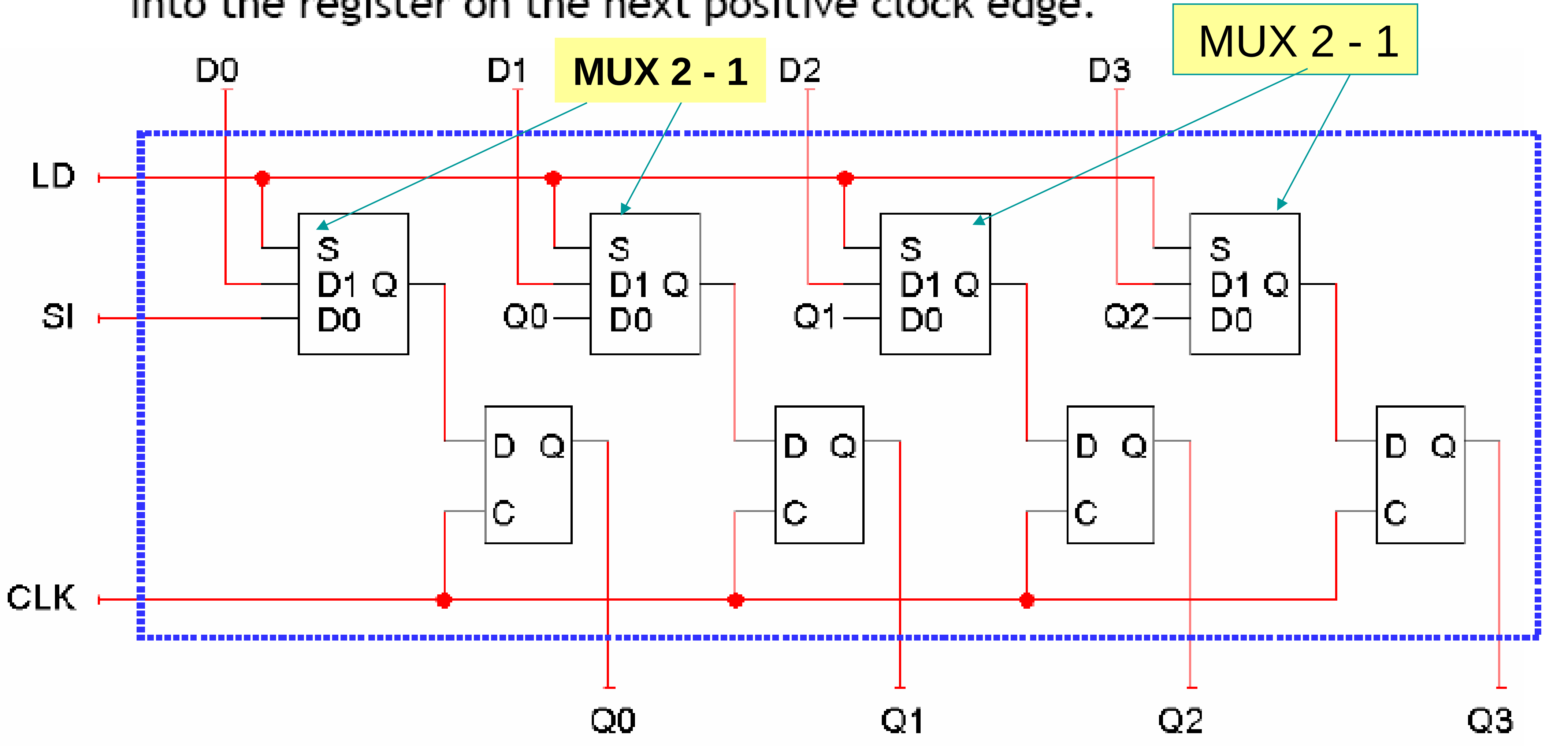


Shift registers with parallel load

- We can add a parallel load operation, just as we did for regular registers.

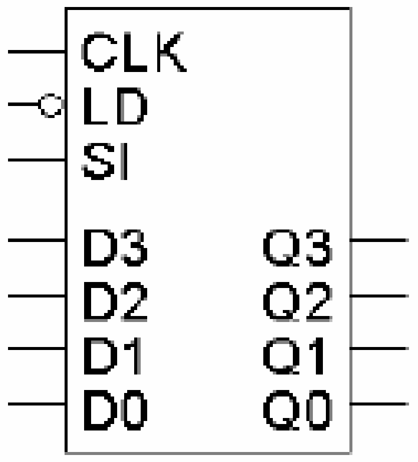
Serial – When LD = 0 the flip-flop inputs will be SIQ0Q1Q2, so the register will shift on the next positive clock edge.

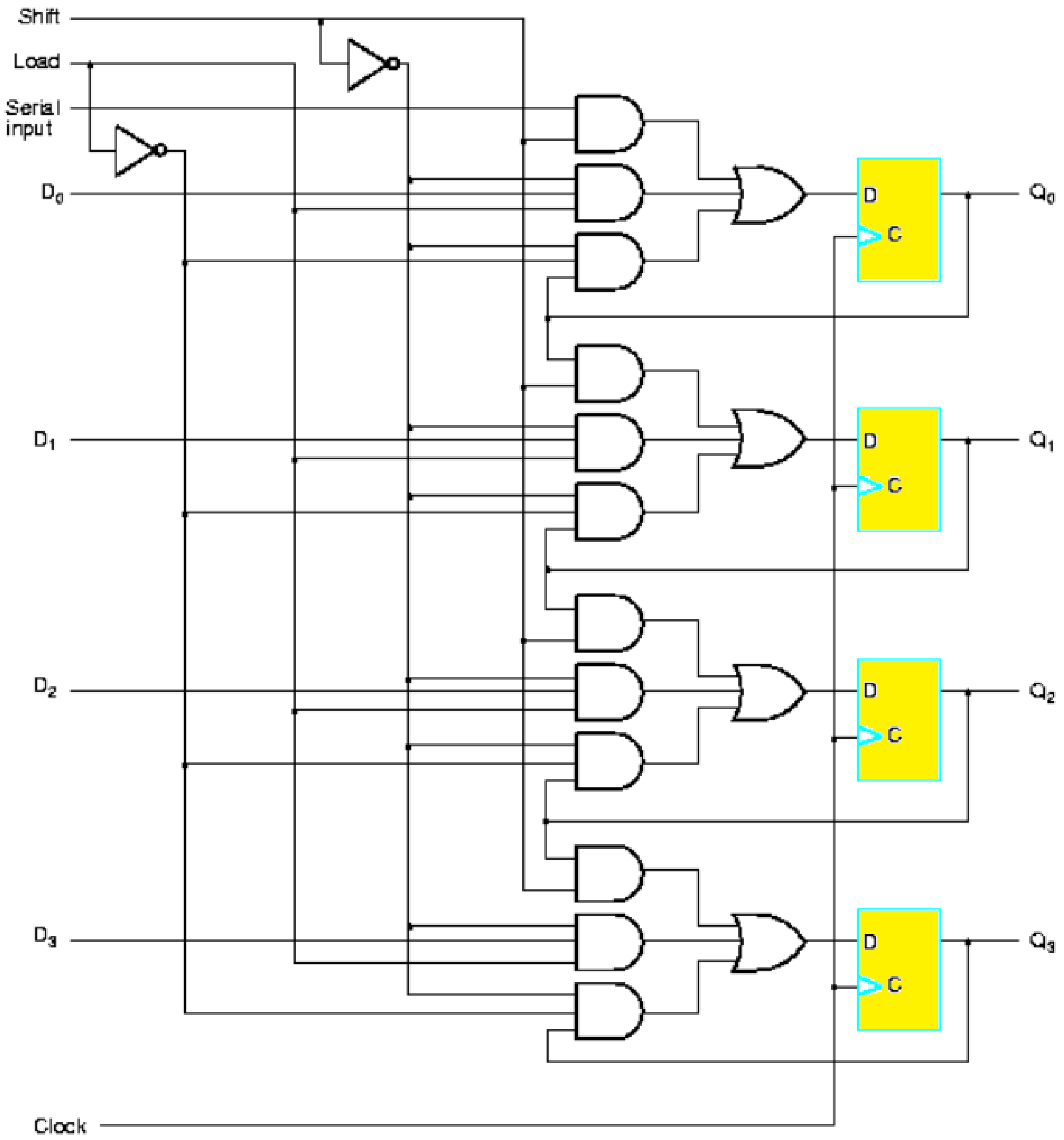
Parallel – When LD = 1, the flip-flop inputs are D0-D3, and a new value is loaded into the register on the next positive clock edge.

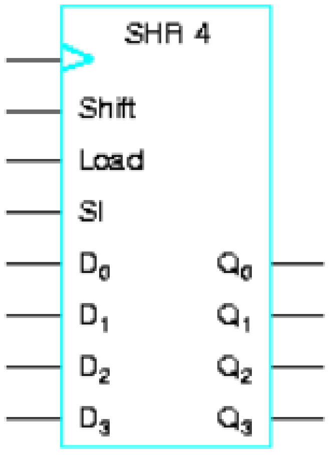


Shift registers in LogicWorks

- Here is a block symbol for the Shift Reg-4 from LogicWorks.
- Its internal implementation is shown on the previous page, except the LD input here is active-low instead.




Shift
Load
Serial input
D_0
D_1
D_2
D_3
Clock
D
C
Q_0
Q_1
Q_2
Q_3


SHR 4
Shift
Load
SI
D_0
D_1
D_2
D_3
Q_0
Q_1
Q_2
Q_3

(b) Symbol

S1	S0	
0	0	No change
0	1	Shift left (down
1	0	(Shift right (up
1	1	Parallel load

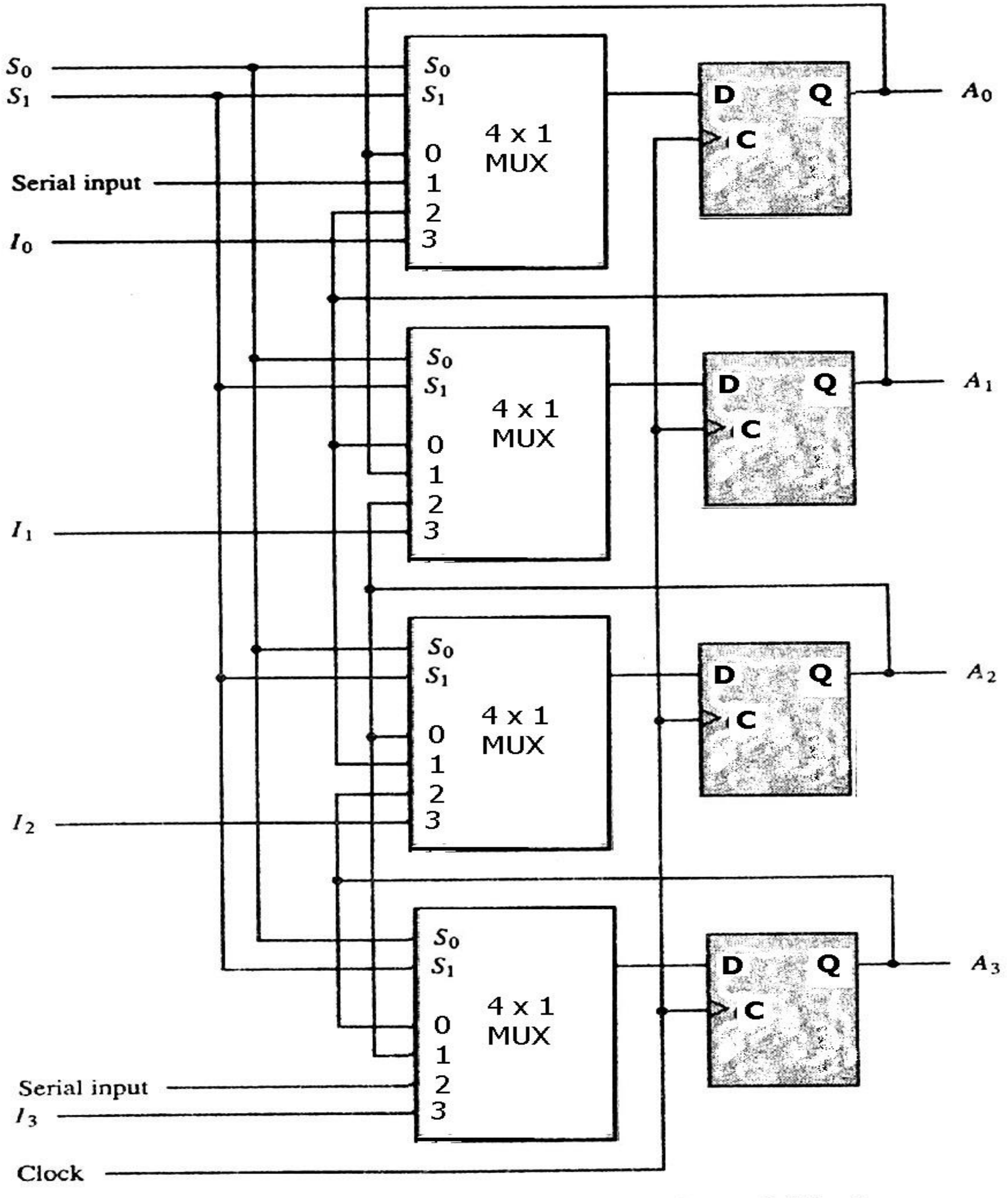


Figure 2-9 Bidirectional shift register with parallel load.

Other types of shift registers

- Logical shifts – Standard shifts like we just saw. In the absence of a SI input, 0 occupies the vacant position.
  - Left: 0110 -> 1100
  - Right: 0110 -> 0011
- Circular shifts (also called ring counters or rotates) – The shifted out bit wraps around to the vacant position.
  - Left: 1001 -> 0011
  - Right: 1001 -> 1100
- Switch-tail ring counter (aka Johnson counter) – Similar to the ring counter, but the serial input is the complement of the serial output.
  - Left: 1001 -> 0010
  - Right: 1001 -> 0100
- Arithmetical shifts – Left shifting is the same as a logical shift. Right shifting however maintains the MSB.
  - Left: 0110 -> 1100
  - Right: 0110 -> 0011; 1011 -> 1101

Example

- Consider a 4-bit register with the following inputs
  - parallel data inputs ABCD = 0011
  - left serial input LSI = 1 (This is the serial input for a left shift.)
  - right serial input RSI = 0 (This is the serial input for a right shift.)
  - The three control inputs S_2, S_1, S_0 operate as shown in the table to the right.
  - Let the initial register content be $Q_A Q_B Q_C Q_D$ = 0101

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0				
0	0	1				
0	1	0				
1	0	0				
0	1	1				
1	0	1				
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LS1	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0				
0	0	1				
0	1	0				
1	0	0				
0	1	1				
1	0	1				
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LSI	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1				
0	1	0				
1	0	0				
0	1	1				
1	0	1				
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111
Other Information			
Parallel Load			ABCD = 0011
LS1			1
RSI			0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0				
1	0	0				
0	1	1				
1	0	1				
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LS1	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	1
1	0	0				
0	1	1				
1	0	1				
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111
Other Information			
Parallel Load			ABCD = 0011
LSI			1
RSI			0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	1
1	0	0	0	0	1	1
0	1	1				
1	0	1				
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LS1	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	1
1	0	0	0	0	1	1
0	1	1	1	0	0	1
1	0	1				
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LS1	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	1
1	0	0	0	0	1	1
0	1	1	1	0	0	1
1	0	1	0	0	1	1
1	1	1				
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LS1	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	1
1	0	0	0	0	1	1
0	1	1	1	0	0	1
1	0	1	0	0	1	1
1	1	1	1	1	1	1
0	0	0				
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LS1	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	1
1	0	0	0	0	1	1
0	1	1	1	0	0	1
1	0	1	0	0	1	1
1	1	1	1	1	1	1
0	0	0	1	1	1	1
1	1	0				

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111

Other Information	
Parallel Load	ABCD = 0011
LSI	1
RSI	0

Example

S2	S1	S0	Q_A	Q_B	Q_C	Q_D
			0	1	0	1
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	1
1	0	0	0	0	1	1
0	1	1	1	0	0	1
1	0	1	0	0	1	1
1	1	1	1	1	1	1
0	0	0	1	1	1	1
1	1	0	0	0	0	0

S2	S1	S0	Operation
0	0	0	left shift
0	0	1	circular left shift
0	1	0	right shift
0	1	1	circular right shift
1	0	0	no change
1	0	1	parallel load
1	1	0	complement each bit
1	1	1	set to 1111
Other Information			
Parallel Load			ABCD = 0011
LS1			1
RSI			0