

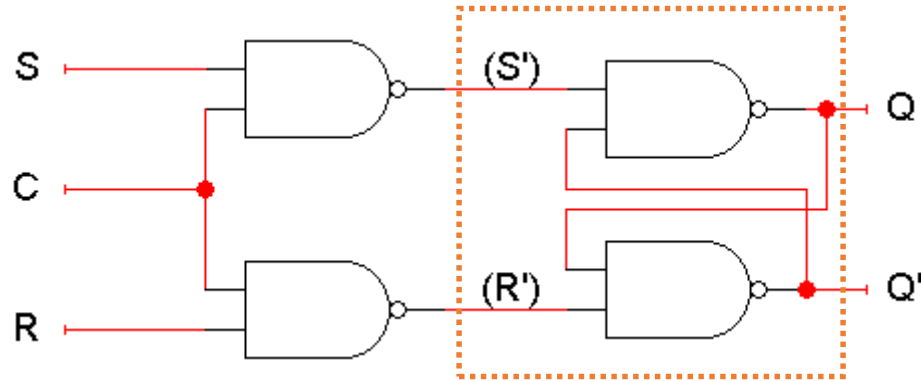
Sequential Design

Contents

- Latches
- Flip-flops
- Registers
- Counters

An SR latch with a control input

- SR latch with a control input C

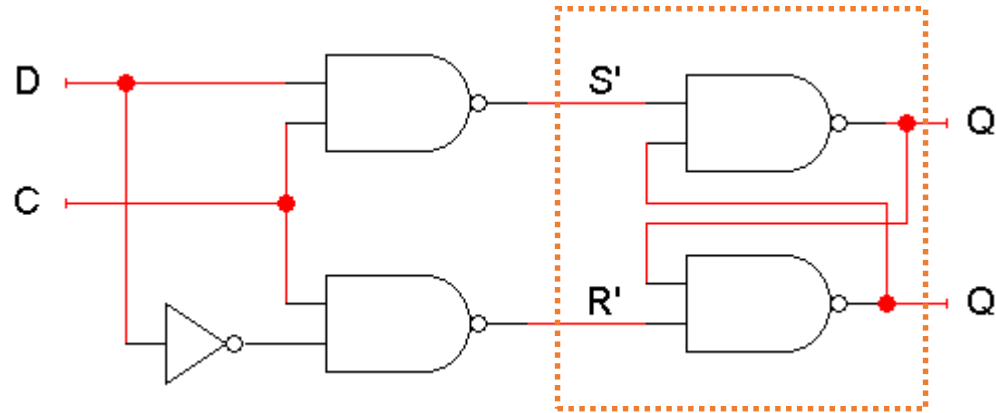


C	S	R	S'	R'	Q
0	x	x	1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	Avoid!

- The dotted box is the S'R' latch.
- The additional NAND gates are simply used to generate the correct inputs for the S'R' latch.
- The control input acts just like an enable.

D latch

- D latch is 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 the state Q does not change.
 - When $C = 1$, the latch output Q will equal the input D .
- No more messing with one input for set and another input for reset!



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

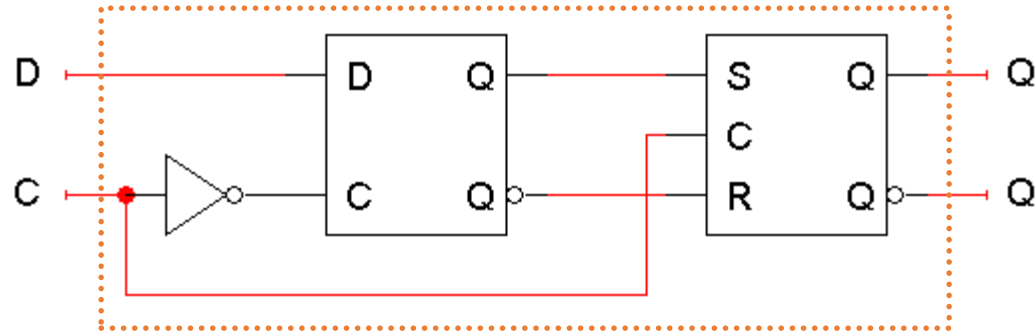
- Also, this latch has no “bad” input combinations to avoid. Any of the four possible assignments to C and D are valid.

About clocks

- Clocks are used extensively in computer architecture.
- All processors run with an internal clock.
 - Modern chips run at frequencies up to 5 GHz.
 - This works out to a cycle time as little as 0.2 ps
- Memory modules are often rated by their clock speeds too
 - Ex: PC133, DDR4 memory

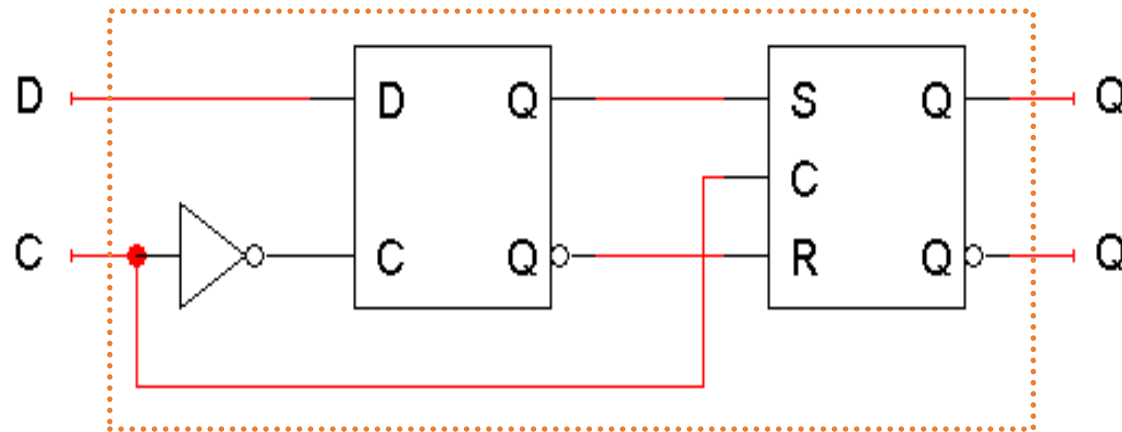
Flip-flops

- The issue was how to enable a latch for just an instant.
- Here is the internal structure of a **D flip-flop**.
 - The flip-flop inputs are C and D, and the outputs are Q and Q'.
 - The D latch on the left is the **master**, while the SR latch on the right is called the **slave**.



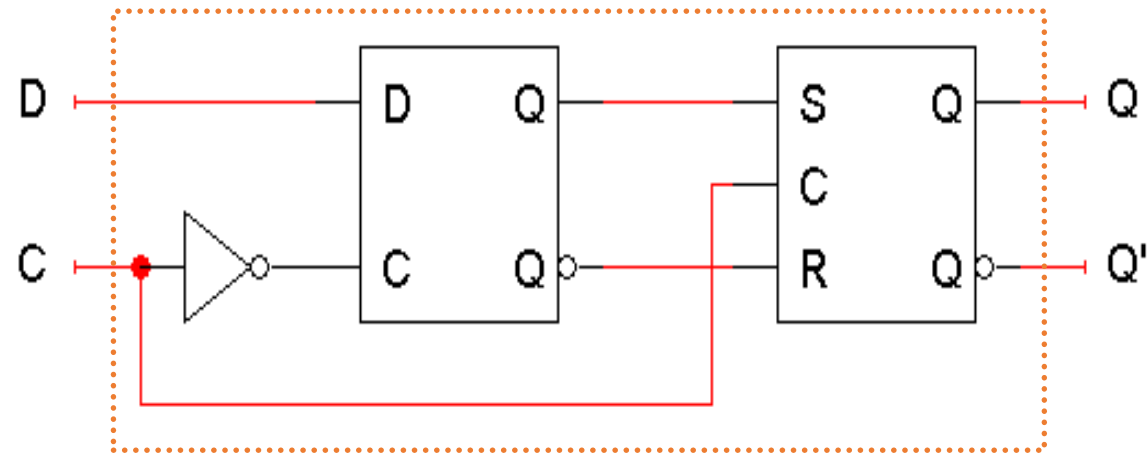
- Note the layout here.
 - The flip-flop input D is connected directly to the master latch.
 - The master latch output goes to the slave.
 - The flip-flop outputs come directly from the slave latch.

D flip-flops when $C=0$



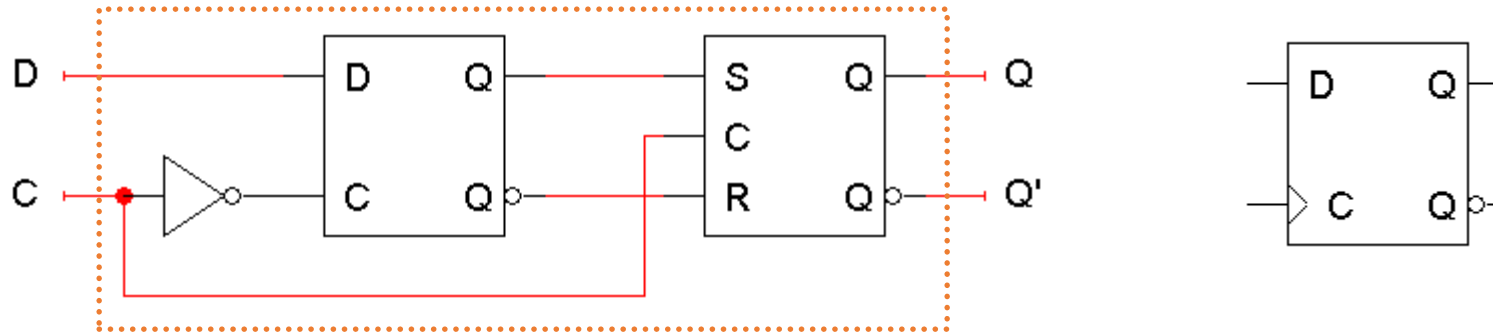
- The D flip-flop's control input C enables *either* the D latch or the SR latch, but not both.
- When $C = 0$:
 - The master latch is enabled, and it monitors the flip-flop input D. Whenever D changes, the master's output changes too.
 - The slave is disabled, so the D latch output has no effect on it. Thus, the slave just maintains the flip-flop's current state.

D flip-flops when C=1



- *As soon as C becomes 1,*
 - The master is disabled. Its output will be the *last* D input value seen just before C became 1.
 - Any subsequent changes to the D input while C = 1 have no effect on the master latch, which is now disabled.
 - The slave latch is enabled. Its state changes to reflect the master's output, which again is the D input value from right when C became 1.

Positive edge triggering



- This is called a **positive edge-triggered** flip-flop.
 - The flip-flop output Q changes *only* after the positive edge of C.
 - The change is based on the flip-flop input values that were present right at the positive edge of the clock signal.
- The D flip-flop's behavior is similar to that of a D latch except for the positive edge-triggered nature, which is not explicit in this table.

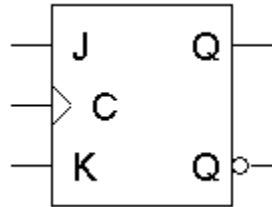
C	D	Q
0	x	No change
1	0	0 (reset)
1	1	1 (set)

Direct inputs

- What is the starting value of Q?
- We could set the initial value synchronously, at the next positive clock edge, but this actually makes circuit design more difficult.
- Instead, most flip-flops provide **direct**, or asynchronous, inputs that let you immediately set or clear the state.
 - You would “reset” the circuit once, to initialize the flip-flops.
 - The circuit would then begin its regular, synchronous operation.

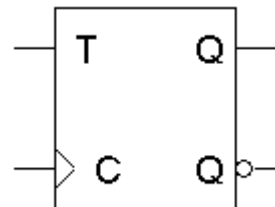
Flip-flop variations

- 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.
- A **JK flip-flop** has inputs that act like S and R, but the inputs JK=11 are used to *complement* 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	$Q'_{current}$

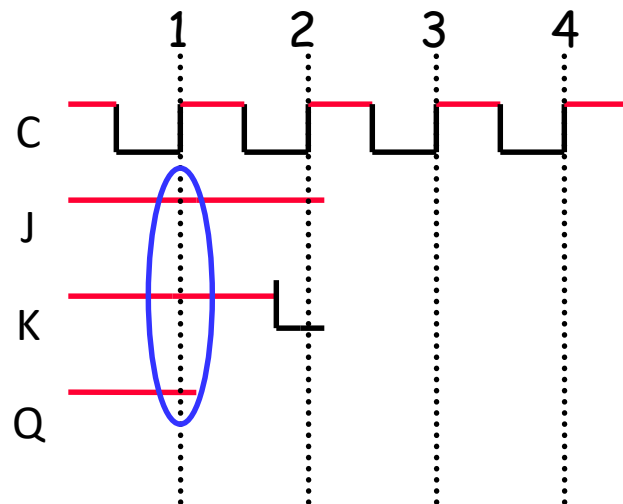
- A **T flip-flop** can only maintain or complement its current state.



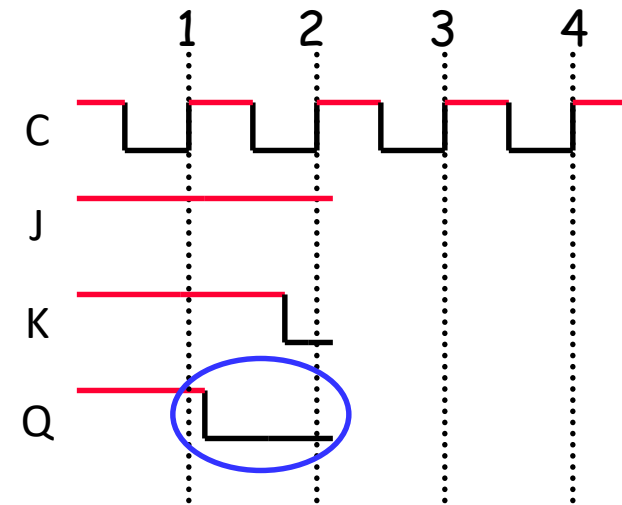
C	T	Q_{next}
0	x	No change
1	0	No change
1	1	$Q'_{current}$

Flip flop timing diagrams

- “Present state” and “next state” are relative terms.
- In the example JK flip-flop timing diagram on the left, you can see that at the first positive clock edge, $J=1$, $K=1$ and $Q(1) = 1$.
- We can use this information to find the “next” state, $Q(2) = Q(1)'$.
- $Q(2)$ appears right after the first positive clock edge, as shown on the right. It will not change again until after the second clock edge.



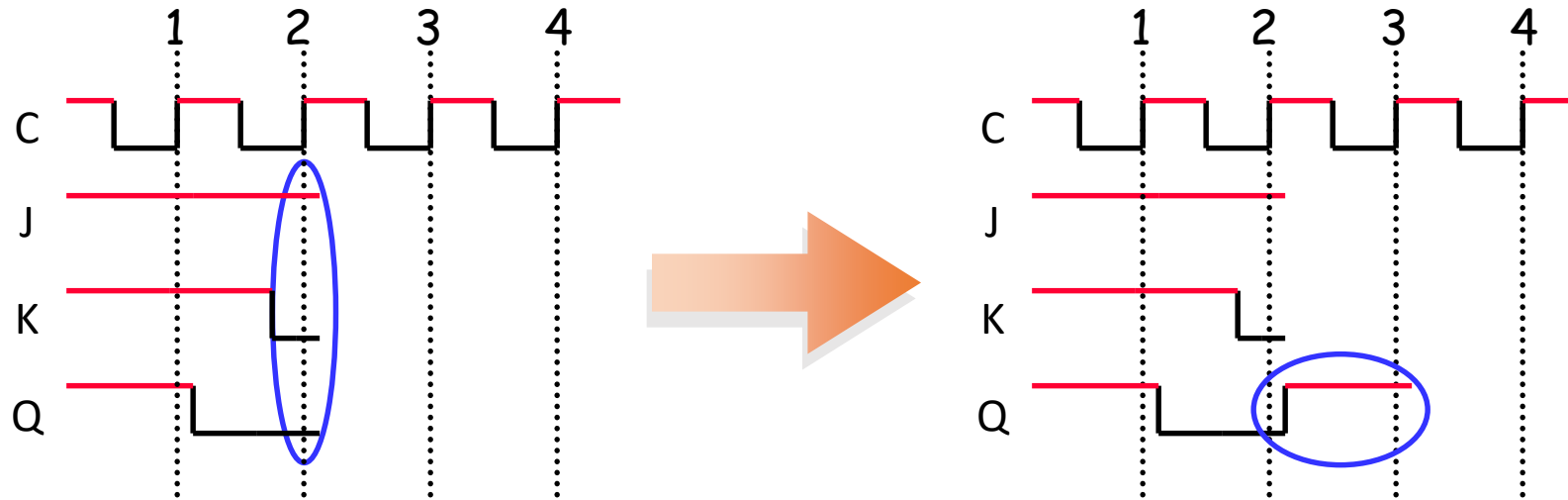
These values at clock cycle 1...



... determine the “next” Q

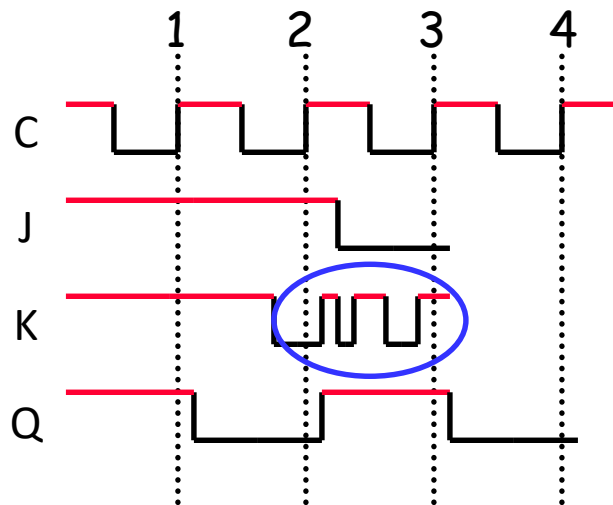
“Present” and “next” are relative

- Similarly, the values of J, K and Q at the second positive clock edge can be used to find the value of Q during the third clock cycle.
- When we do this, Q(2) is now referred to as the “present” state, and Q(3) is now the “next” state.



Positive edge triggered

- One final point to repeat: the flip-flop outputs are affected only by the input values *at the positive edge*.
 - In the diagram below, K changes rapidly between the second and third positive edges.
 - But it's only the input values at the third clock edge ($K=1$, and $J=0$ and $Q=1$) that affect the next state, so here Q changes to 0.
- This is a fairly simple timing model. In real life there are “setup times” and “hold times” to worry about as well, to account for internal and external delays.



Registers

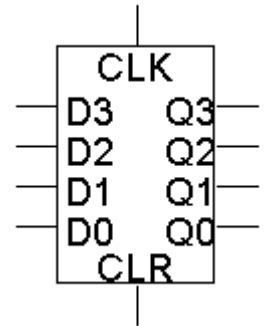
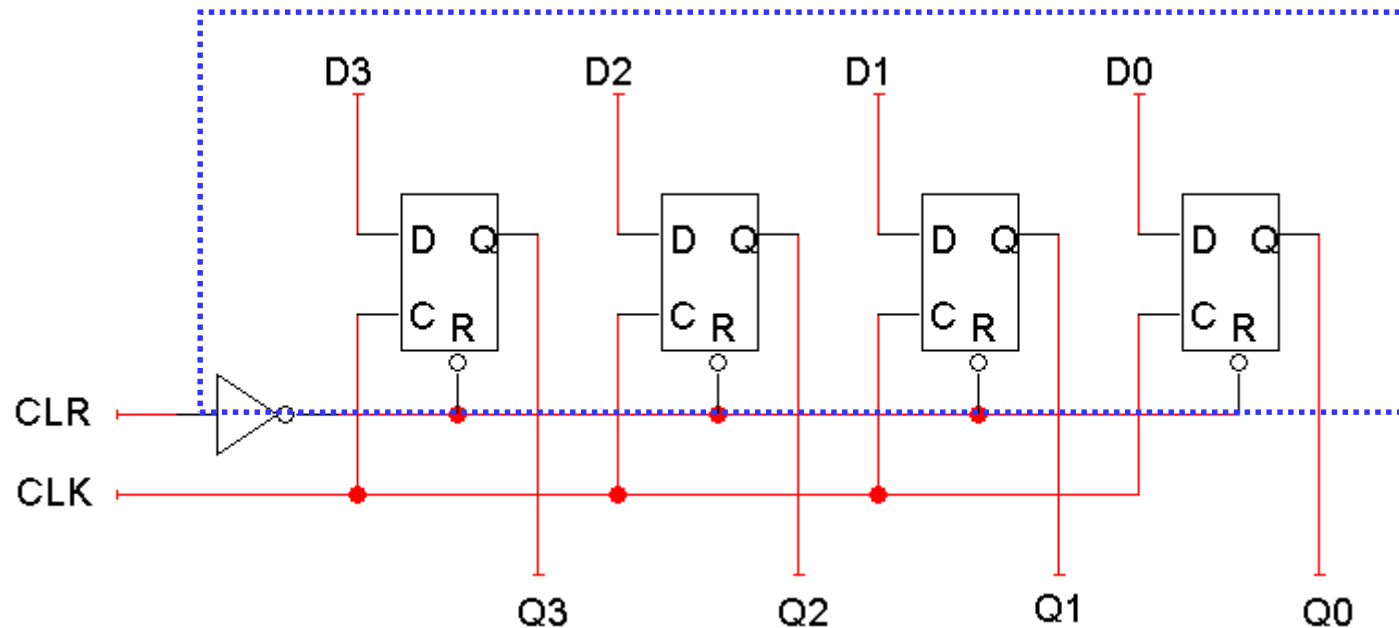
- Registers good example of sequential analysis and design
- They are also frequently used in building larger sequential circuits
- Registers hold larger quantities of data than individual flip-flops
 - Registers are central to the design of modern processors
 - There are many different kinds of registers

What good are registers?

- Flip-flops are limited because they can store only one bit.
 - We have to use two flip-flops for two-bit counter
 - 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

A basic register

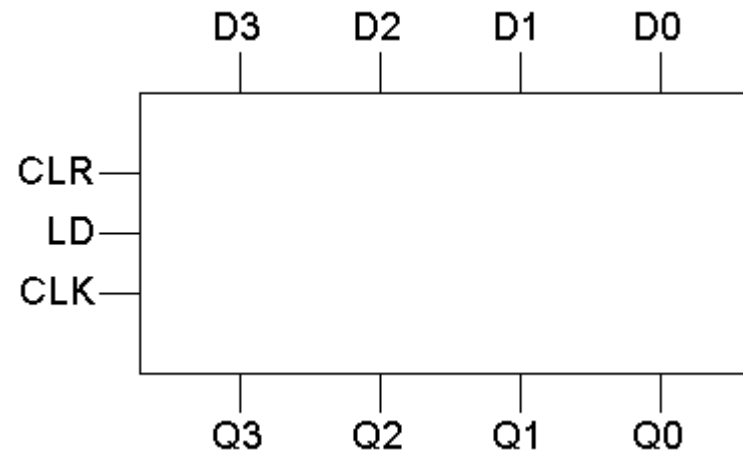
- Basic registers are easy to build.
- We can store multiple bits just by putting a bunch of flip-flops together
- A 4-bit register is on the right, and its internal implementation is below.
 - This register uses D flip-flops, so it's easy to store data without worrying about flip-flop input equations.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



Adding a parallel load operation

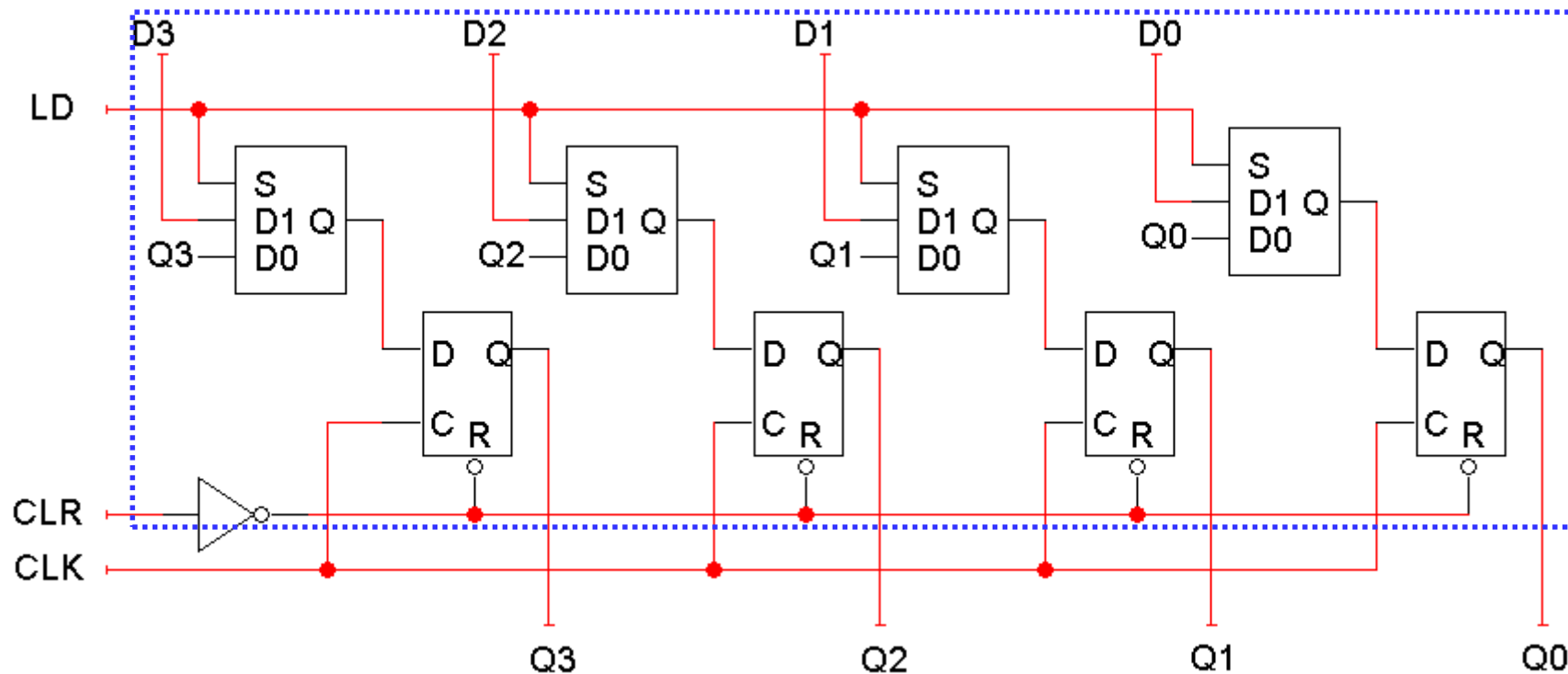
- The input D_3-D_0 is copied to the output Q_3-Q_0 on *every* clock cycle.
- How can we store the current value for more than one cycle?
- Let's add a load input signal LD to the register.
 - If $LD = 0$, the register keeps its current contents.
 - If $LD = 1$, the register stores a new value, taken from inputs D_3-D_0 .

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

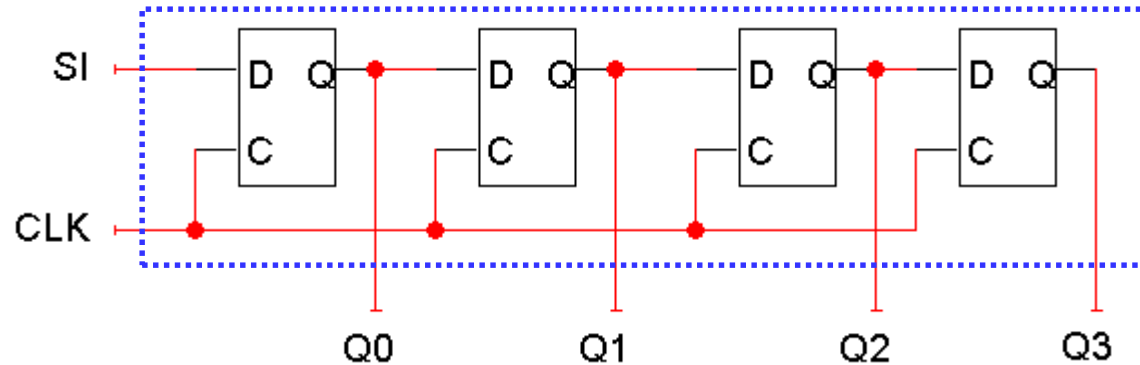


A better parallel load

- Another idea is to modify the flip-flop D inputs
 - When $LD = 0$, the flip-flop inputs are Q_3-Q_0 , so each flip-flop just keeps its current value.
 - When $LD = 1$, the flip-flop inputs are D_3-D_0 , and this new value is “loaded” into the register.



Shift registers



$$\begin{aligned} Q_0(t+1) &= SI \\ Q_1(t+1) &= Q_0(t) \\ Q_2(t+1) &= Q_1(t) \\ Q_3(t+1) &= Q_2(t) \end{aligned}$$

- A **shift register** “shifts” its output once every clock cycle.
- **SI** is an input that supplies a new bit to shift “into” the register.
- For example, if on some positive clock edge we have:

$$SI = 1$$

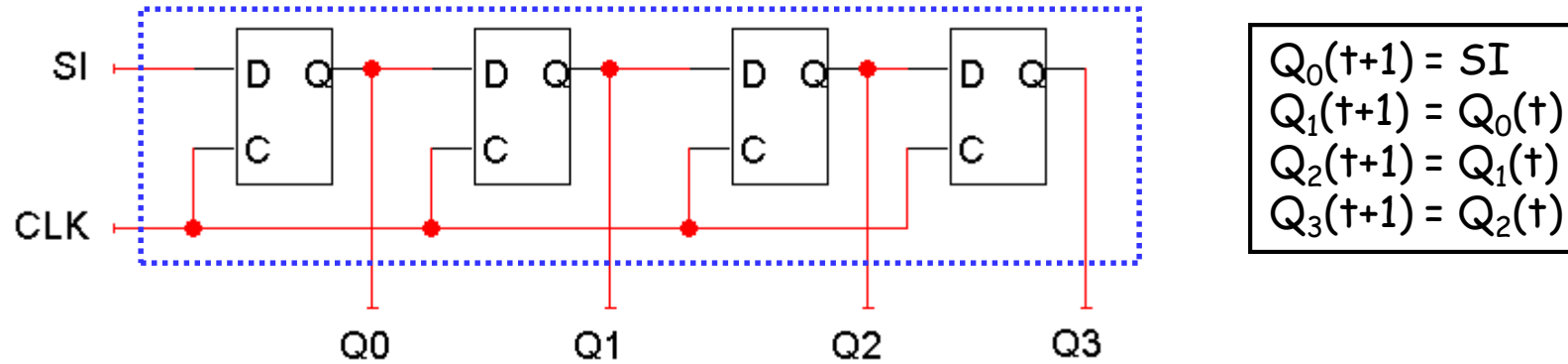
$$Q_0-Q_3 = 0110$$

then the next state will be:

$$Q_0-Q_3 = 1011$$

- The current Q_3 (0 in this example) will be lost on the next cycle.

Shift direction



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

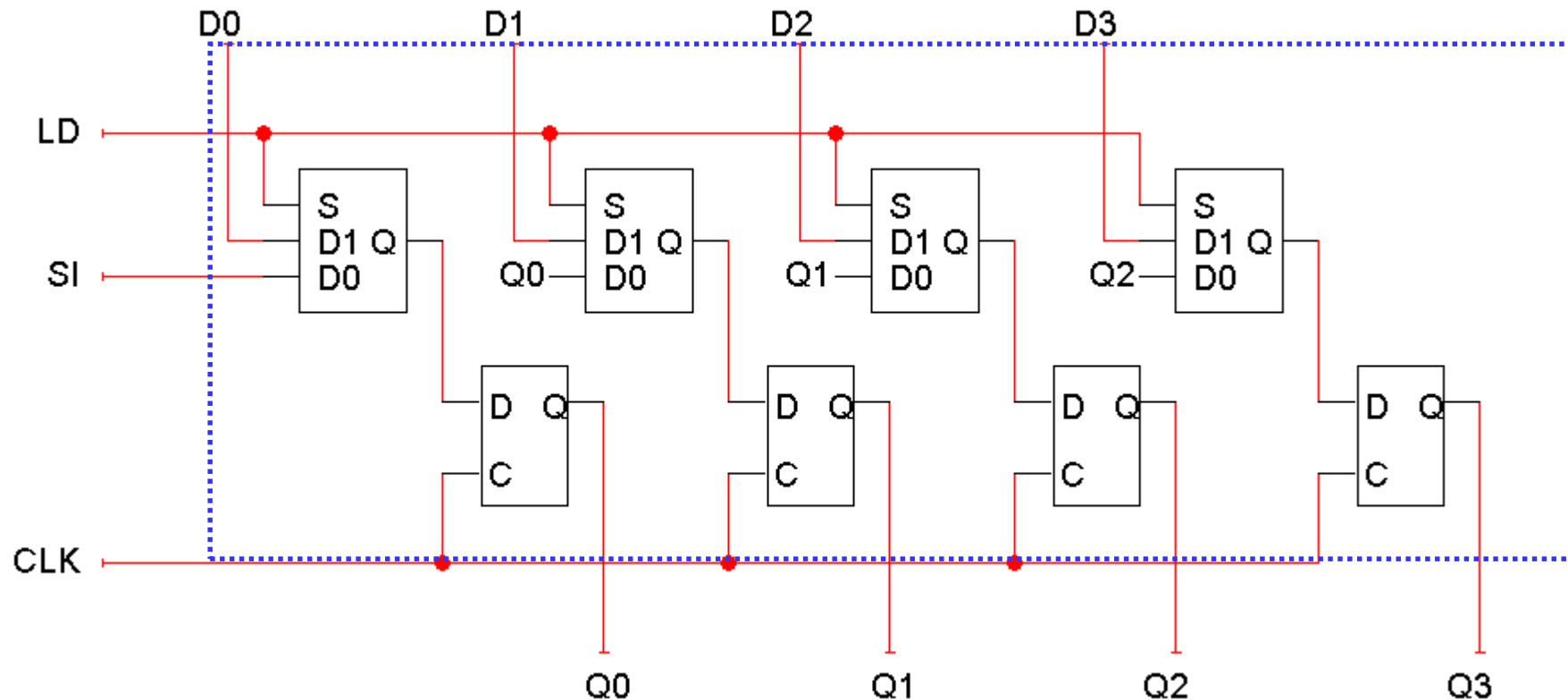
Present Q_0 - Q_3	SI	Next Q_0 - Q_3
ABCD	X	XABC

- But it really depends on your interpretation of the bits. If you consider Q3 to be the most significant bit instead, then the register is shifting in the *opposite* direction!

Present Q_3 - Q_0	SI	Next Q_3 - Q_0
DCBA	X	CBAX

Shift registers with parallel load

- We can add a parallel load, just like we did for regular registers.
 - When $LD = 0$, the flip-flop inputs will be $SIQ_0Q_1Q_2$, so the register shifts on the next positive clock edge.
 - When $LD = 1$, the flip-flop inputs are D_0-D_3 , and a new value is loaded into the shift register, on the next positive clock edge.



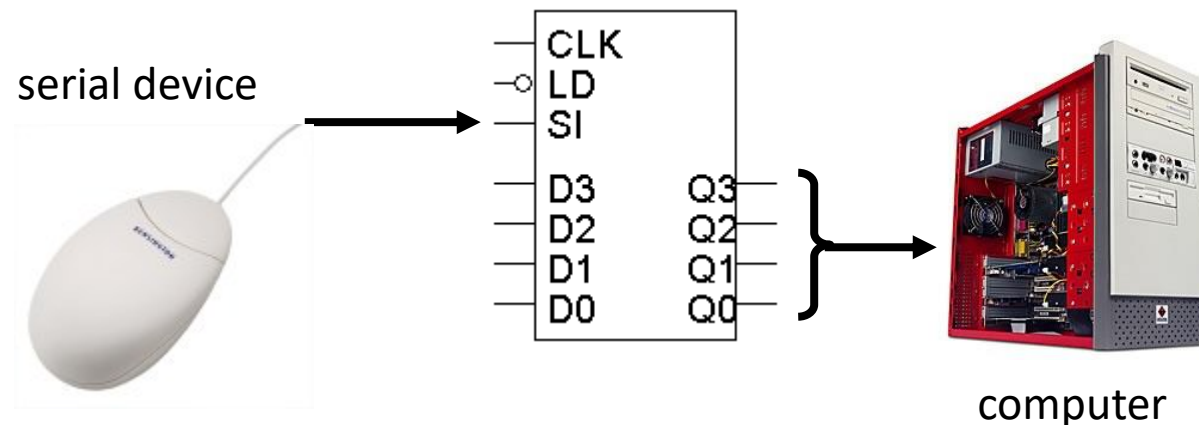
Serial data transfer

- One application of shift registers is converting between “serial data” and “parallel data.”
- Computers typically work with multiple-bit quantities.
 - ASCII text characters are 8 bits long.
 - Integers, single-precision floating-point numbers, and screen pixels are up to 32 bits long.
- But sometimes it’s necessary to send or receive data **serially**, or one bit at a time. Some examples include:
 - Input devices such as keyboard and mouse.
 - Output devices like printers.
 - Any serial port, USB or Firewire device transfers data serially.
 - Recent switch from Parallel ATA to Serial ATA in hard drives.



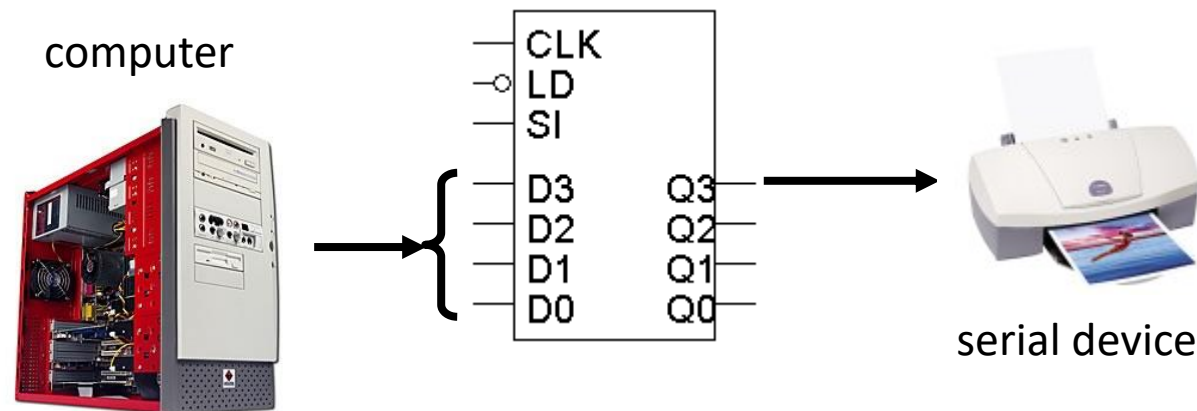
Receiving serial data

- To *receive* serial data using a shift register:
 - The serial device is connected to the register's SI input.
 - The shift register outputs Q3-Q0 are connected to the computer.
- The serial device transmits one bit of data per clock cycle.
 - These bits go into the SI input of the shift register.
 - After four clock cycles, the shift register will hold a four-bit word.
- The computer then reads all four bits at once from the Q3-Q0 outputs.



Sending data serially

- To *send* data serially with a shift register, you do the opposite:
 - The CPU is connected to the register's D inputs.
 - The shift output (Q3 in this case) is connected to the serial device.
- The computer first stores a four-bit word in the register, in one cycle.
- The serial device can then read the shift output.
 - One bit appears on Q3 on each clock cycle.
 - After four cycles, the entire four-bit word will have been sent.



Registers in Modern Hardware

- Registers store data in the CPU
 - Used to supply values to the ALU.
 - Used to store the results.
- If we can use registers, why bother with RAM?

CPU	GPR's	Size	L1 Cache	L2 Cache
Pentium 4	8	32 bits	8 KB	512 KB
Athlon XP	8	32 bits	64 KB	512 KB
Athlon 64	16	64 bits	64 KB	1024 KB
PowerPC 970 (G5)	32	64 bits	64 KB	512 KB
Itanium 2	128	64 bits	16 KB	256 KB
MIPS R14000	32	64 bits	32 KB	16 MB

Answer: Registers are expensive!

- Registers occupy the most expensive space on a chip – the core.
- L1 and L2 are very fast RAM – but not as fast as registers.

OVERVIEW OF COUNTERS

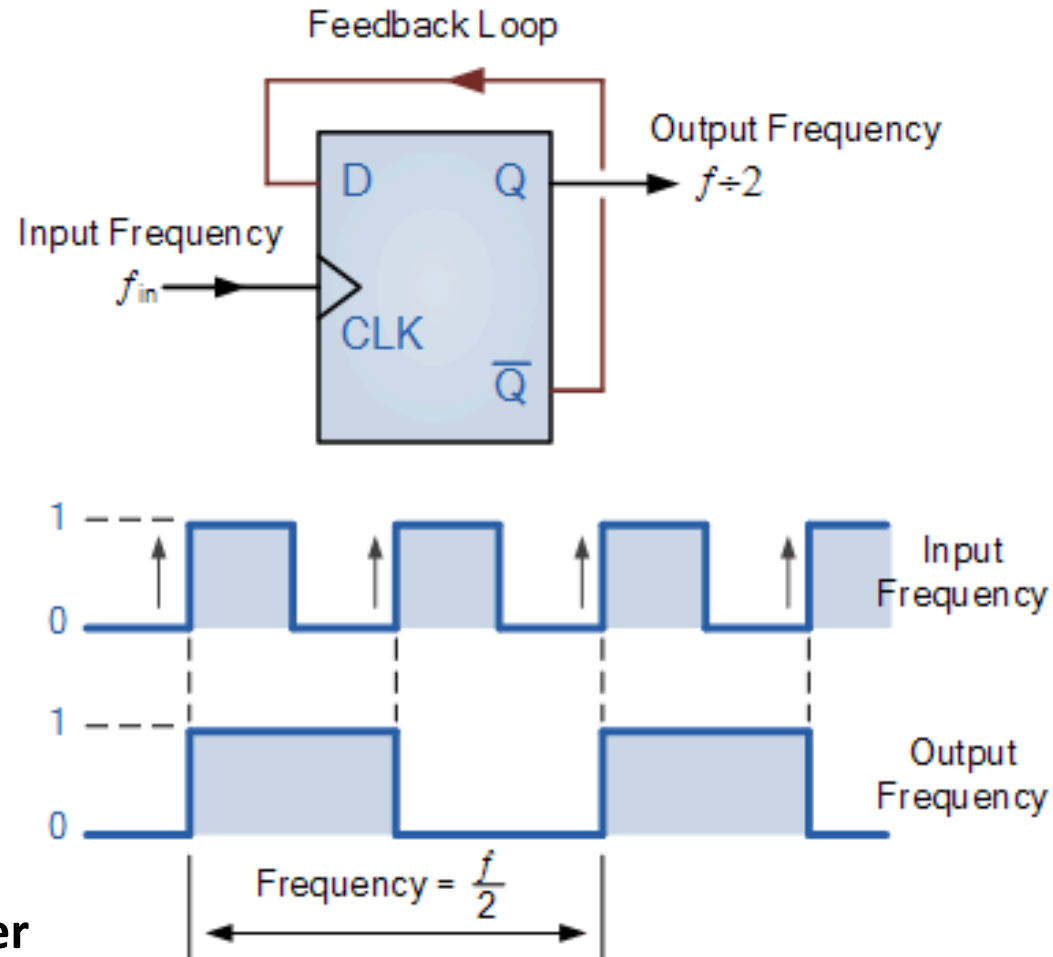
- Counter-by definition
 - One input (clock)
 - Outputs follow defined sequence
- Common tasks of counter
 - Count up or down
 - Increment or decrement count
 - Sequence events
 - Divide frequency
 - Address memory
 - As memory

CHARACTERISTICS OF COUNTERS

- Number of bits (4-bit, 8-bit, etc.)
- Maximum count
 - 4 bit = $2^4 = 0000$ to 1111 in binary
 - 8 bit = $2^8 = 0000\ 0000$ to $1111\ 1111$ in binary
- Modulus of counter-number of states
 - Decade counter
 - 4-bit
 - 8-bit
- Up or down counter
- Asynchronous or synchronous counter
- Presetable counter
- Self-stopping counter

COUNTER USED FOR FREQUENCY DIVISION

Divide by 2 Counter



Explain: Divide by 'N' Counter

Reference: ON Semiconductor – Divide by counters, Synchronous Logic

COUNTER USED FOR FREQUENCY DIVISION

- Divide by 3, modulo 3 counter
- Divide by 5, modulo 5 counter
- Divide by odd number, duty cycle not equal to 50%
- Divide by odd number, duty cycle equal to 50%
- Divide by, say 4.5