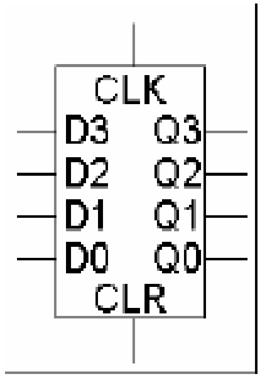
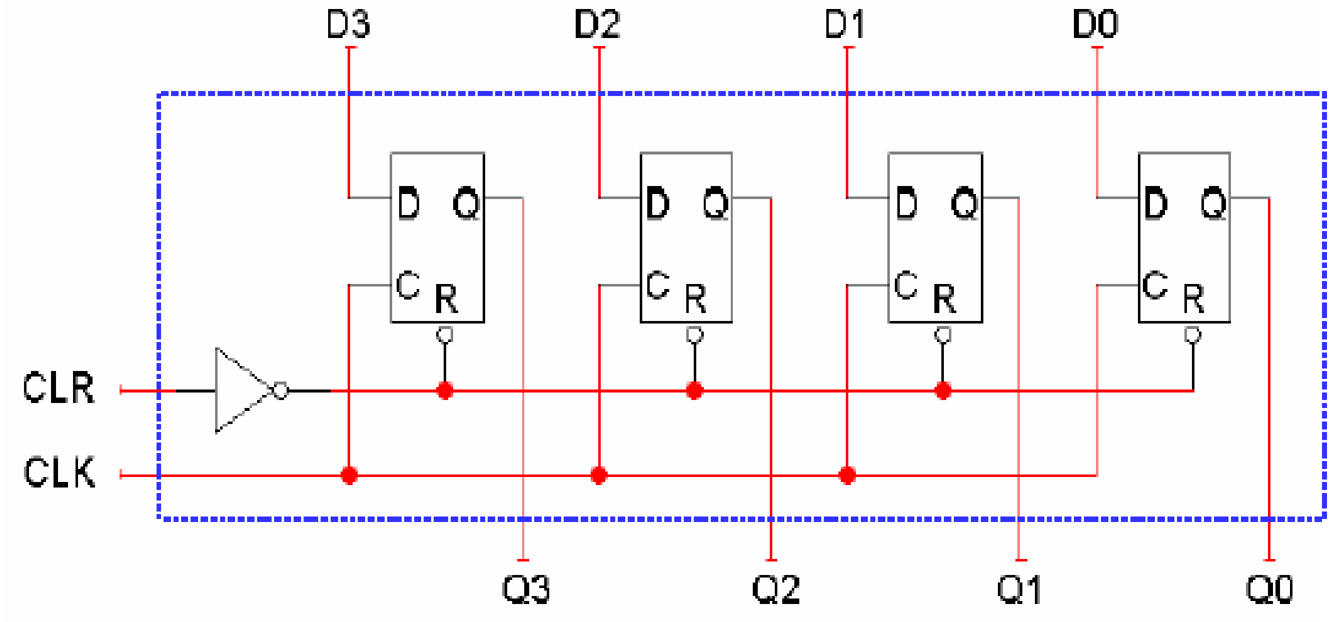
Registers

- 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.

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.

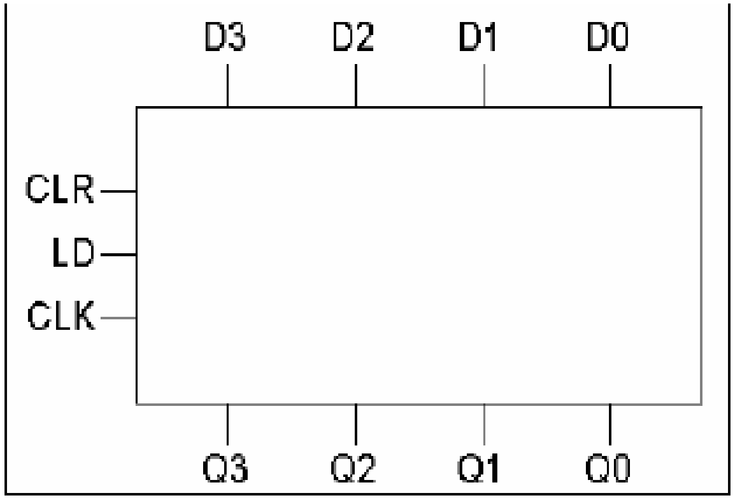




Adding another operation

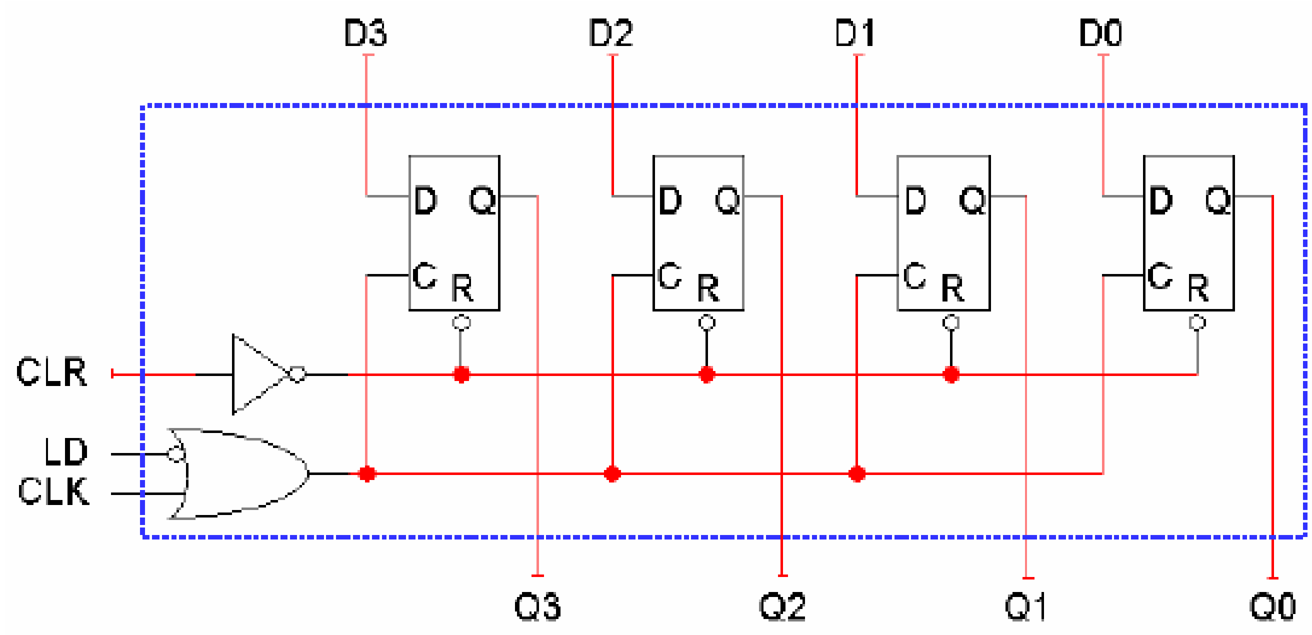
- The input D3-D0 is copied to the output Q3-Q0 on *every* clock cycle.
- How can we store the current value for more than one cycle?
- Let's try to add a load input signal LD to the register.
  - If LD = 0, the register keeps its current contents.
  - If LD = 1, the register stores a new value, taken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

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



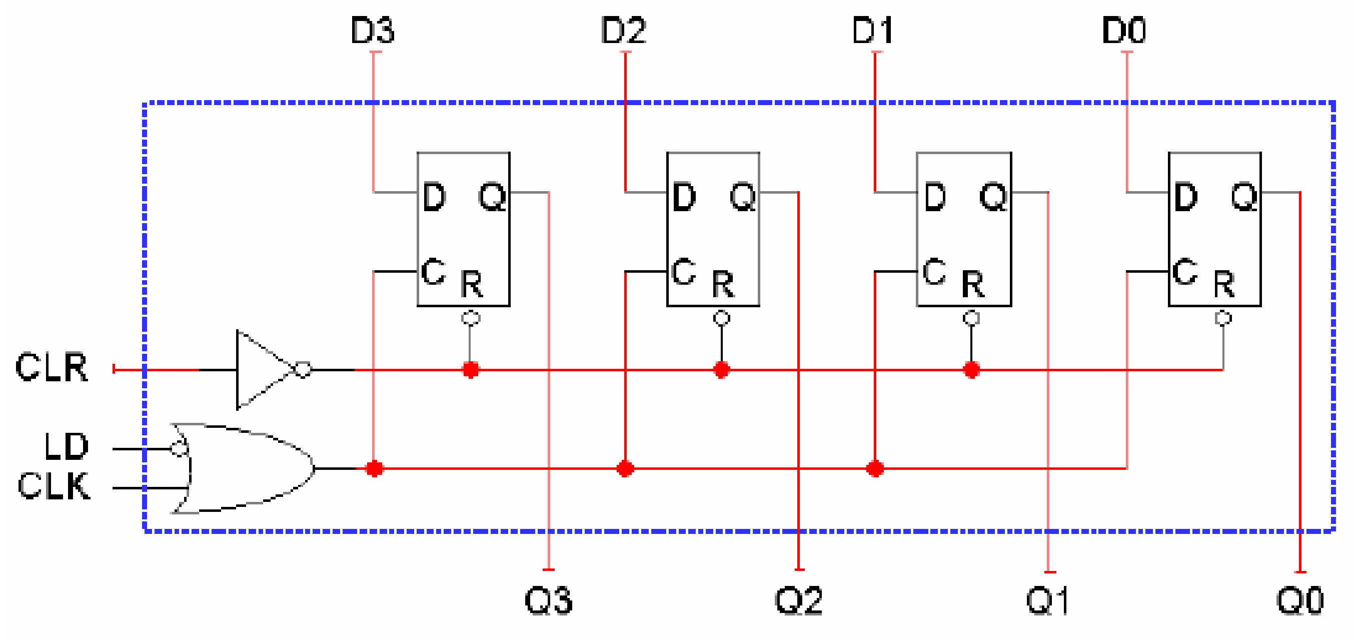
Clock gating

- We could implement the load ability by manipulating the CLK input, as shown below.
  - When LD = 0, the flip-flop C inputs are held at 1. There is no positive clock edge, so the flip-flops keep their current values.
  - When LD = 1, the CLK input passes through the OR gate, so all of the flip-flops will receive a positive clock edge and can load a new value from the D3-D0 inputs.



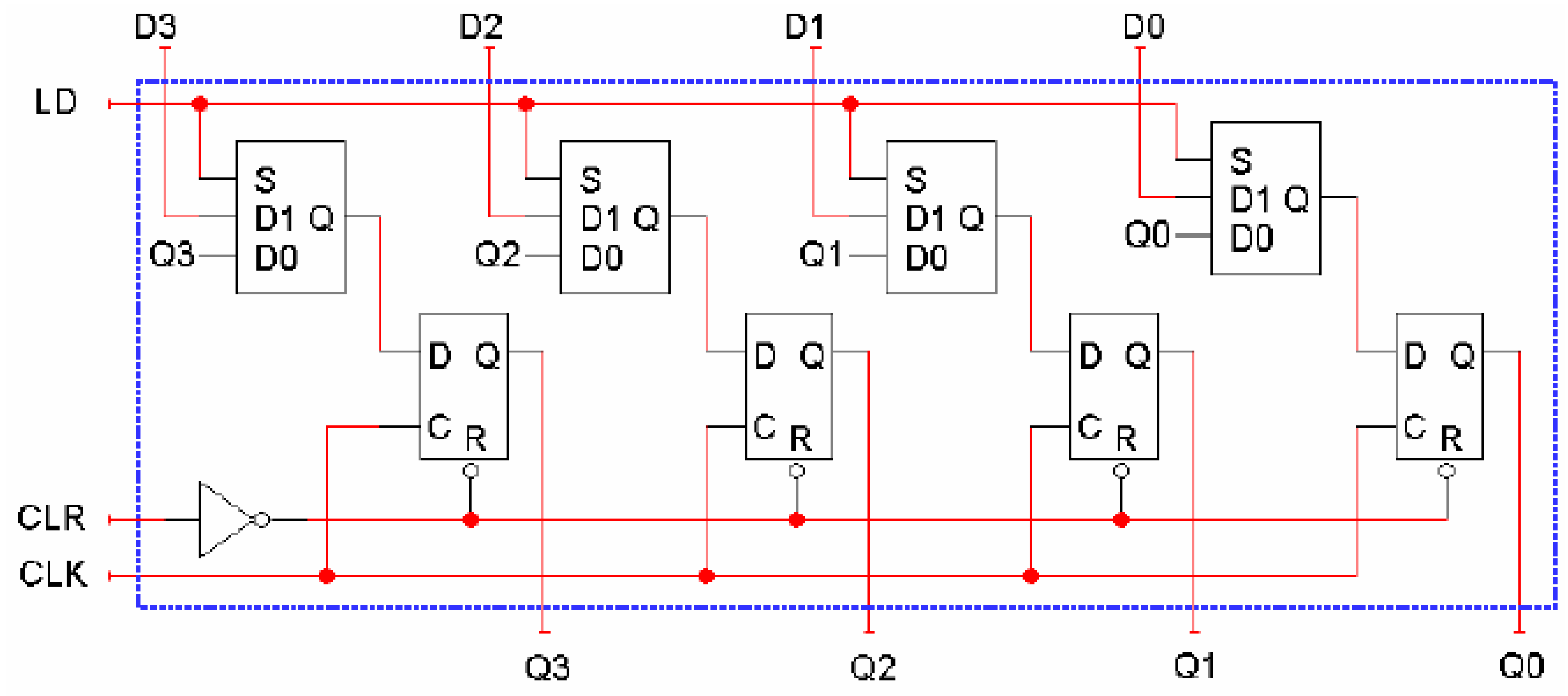
Clock gating is bad

- This is called clock gating, since gates are added to the clock signal.
- There can be timing problems similar to those of latches. Here, LD must be kept at 1 for the right length of time (one clock cycle) and no longer.
- The actual clock signal is delayed a little bit by the OR gate.
  - In more complex circuits, different flip-flops might receive the clock signal at slightly different times.
  - This clock skew can lead to synchronization problems.



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.



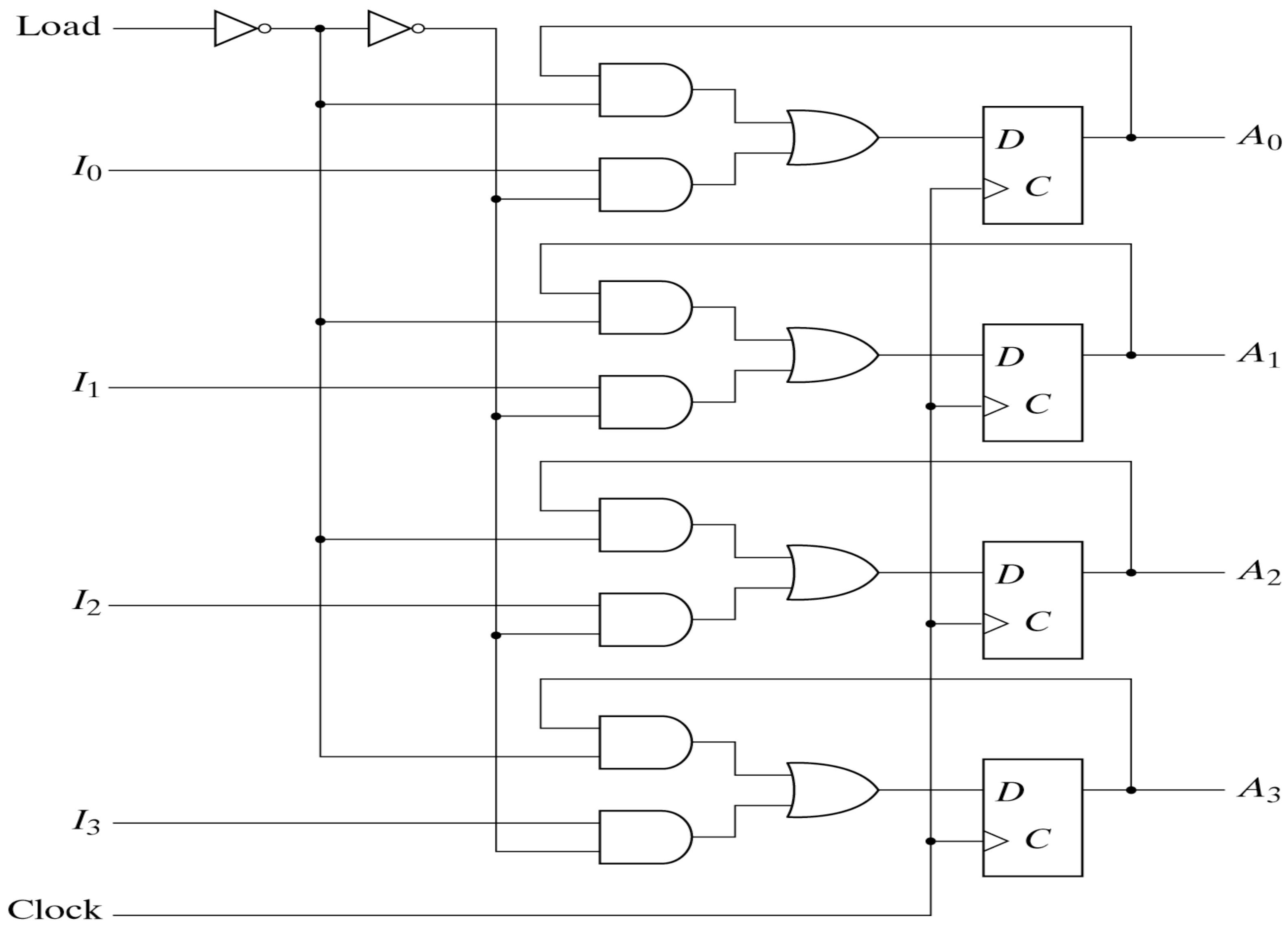
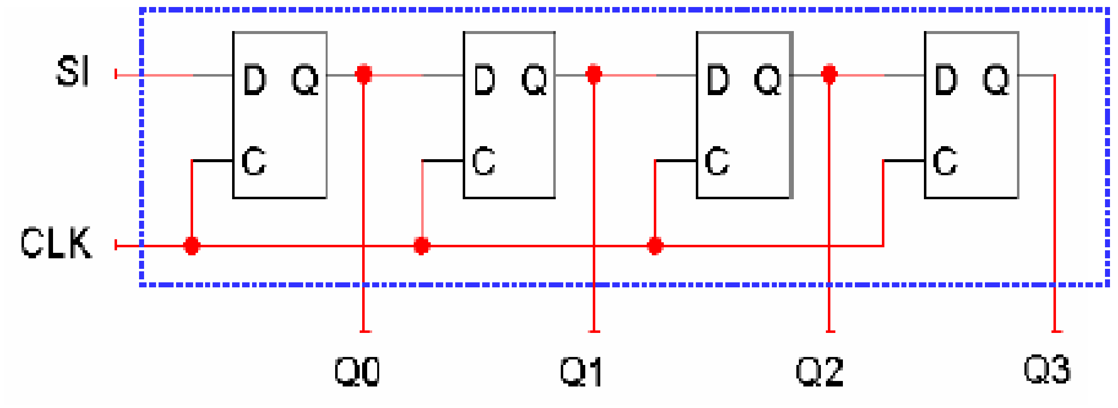


Fig. 6-2 4-Bit Register with Parallel Load

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.



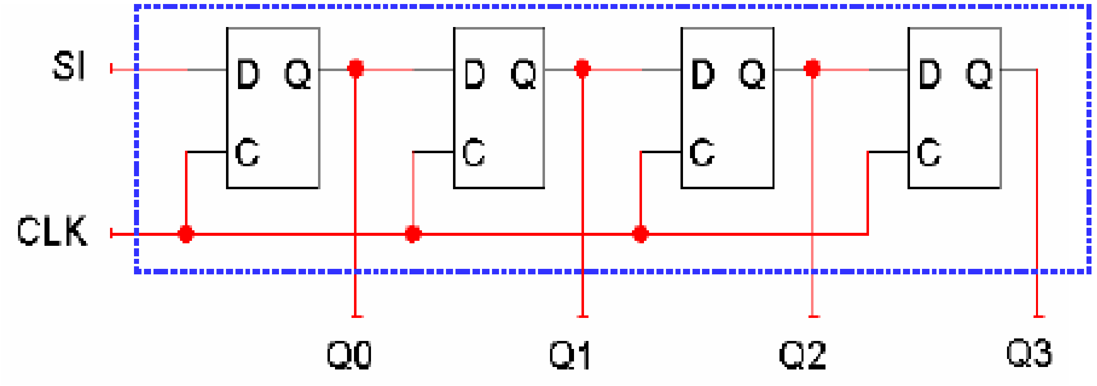
Q0(t+1) = SI
Q1(t+1) = Q0(t)
Q2(t+1) = Q1(t)
Q3(t+1) = Q2(t)

- 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.

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."

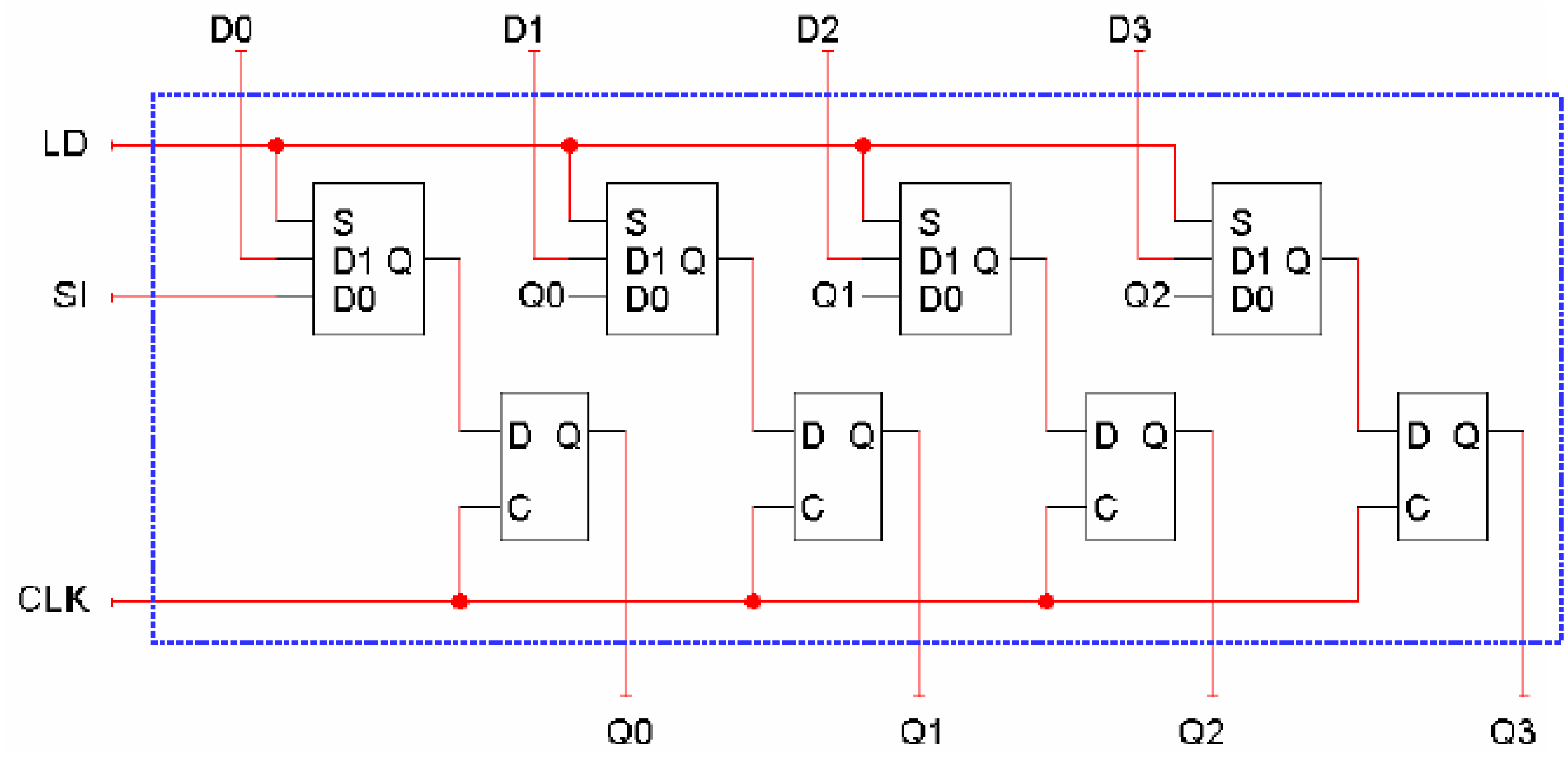
Present Q0-Q3	SI	Next Q0-Q3
ABCD	X	XABC

- 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

Shift registers with parallel load

- We can add a parallel load operation, just as we did for regular registers.
  - When LD = 0 the flip-flop inputs will be SIQ0Q1Q2, so the register will shift on the next positive clock edge.
  - 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.

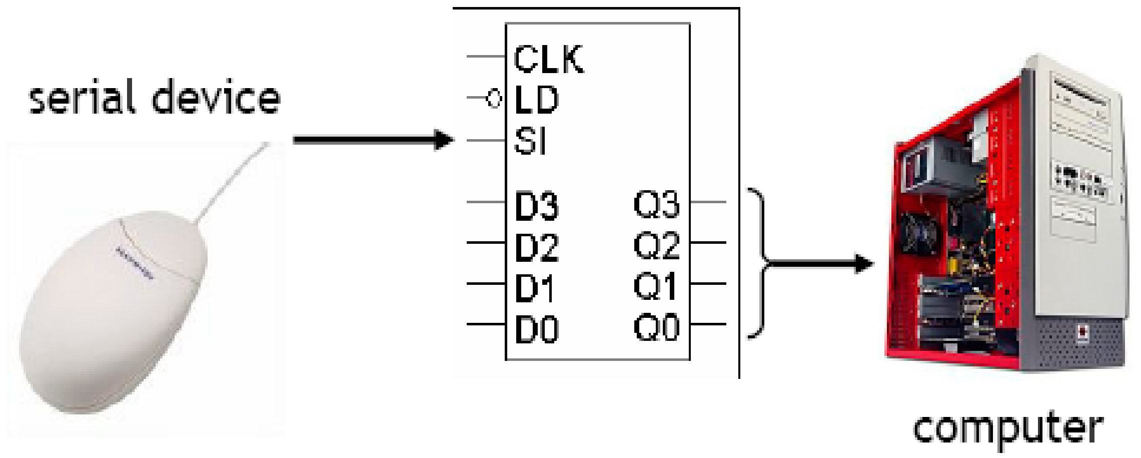


Serial data transfer

- One application of shift registers is converting between serial data and parallel data.
- Computers typically work with multi-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, one bit at a time. For example, USB and Firewire devices such as keyboards, mice and printers all transfer data serially.

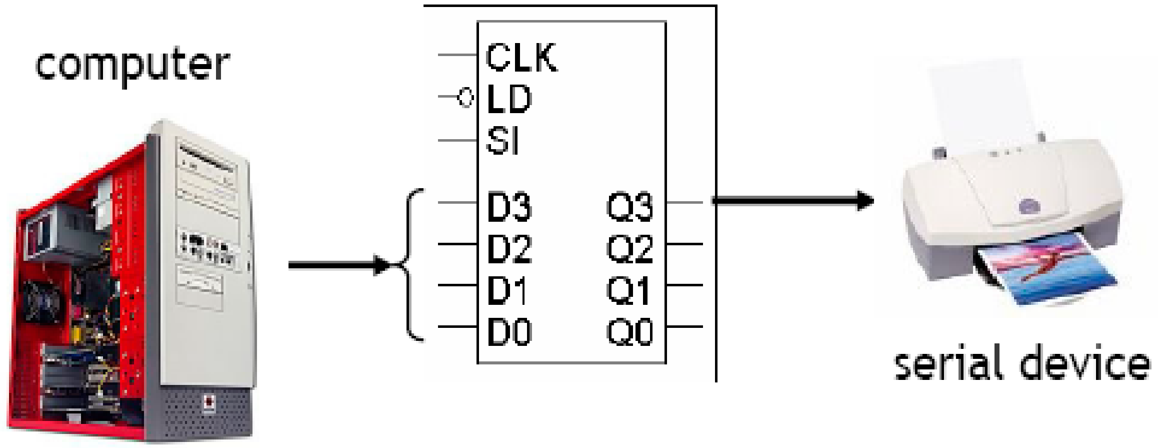
Receiving serial data

- You can convert serial data to parallel data using a shift register.
  - The serial device is connected to the register's SI input.
  - Shift register outputs Q3-Q0 are connected to the parallel device.
- 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

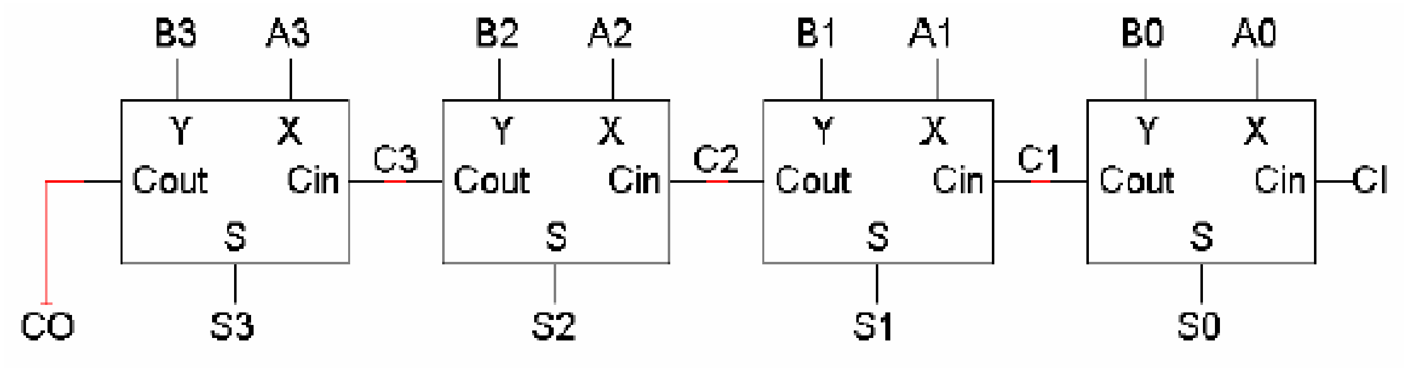
- To send data serially with a shift register, you can do the opposite.
  - The parallel device is connected to the register's D3-D0 inputs.
  - The shift register output Q3 is connected to the serial device.
- The computer first stores a four-bit word in the register, in one cycle.
- The serial device 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.



Serial addition

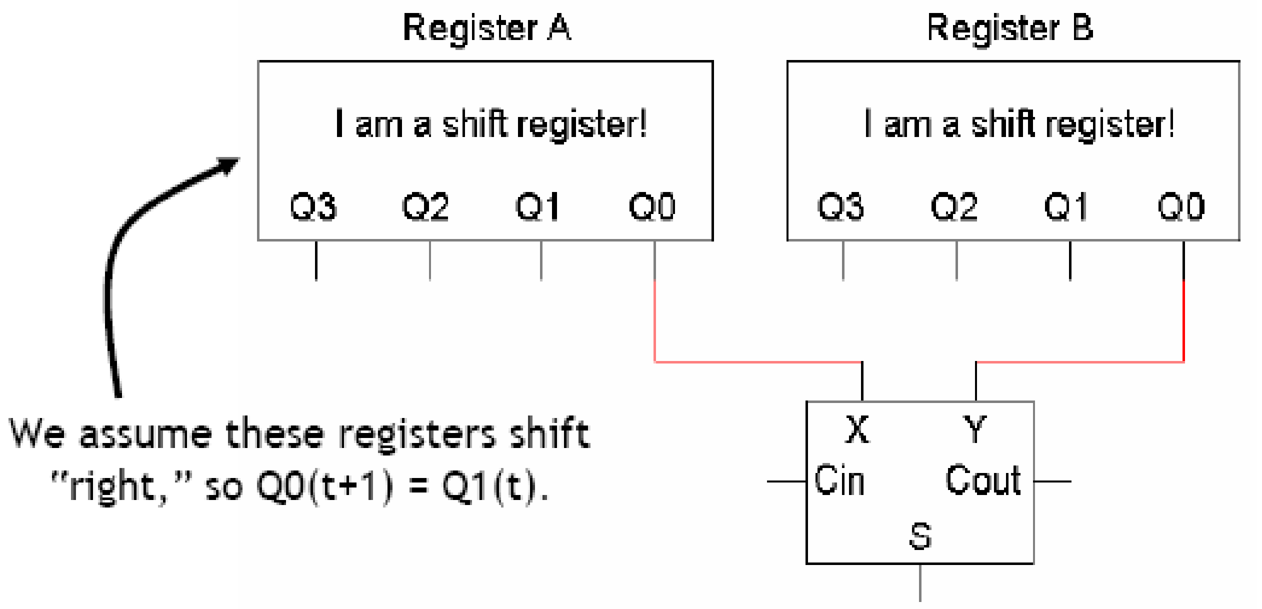
- A second example using shift registers is adding two *n*-bit numbers with significantly less hardware than a standard adder.
- A four-bit ripple-carry adder contains four full adders, but note that the addition really happens serially, one step at a time.

1. Add A0 + B0 + CI to get S0 and C1.
2. Add A1 + B1 + C1 to get S1 and C2.
3. Add A2 + B2 + C2 to get S2 and C3.
4. Add A3 + B3 + C3 to get S3 and CO.



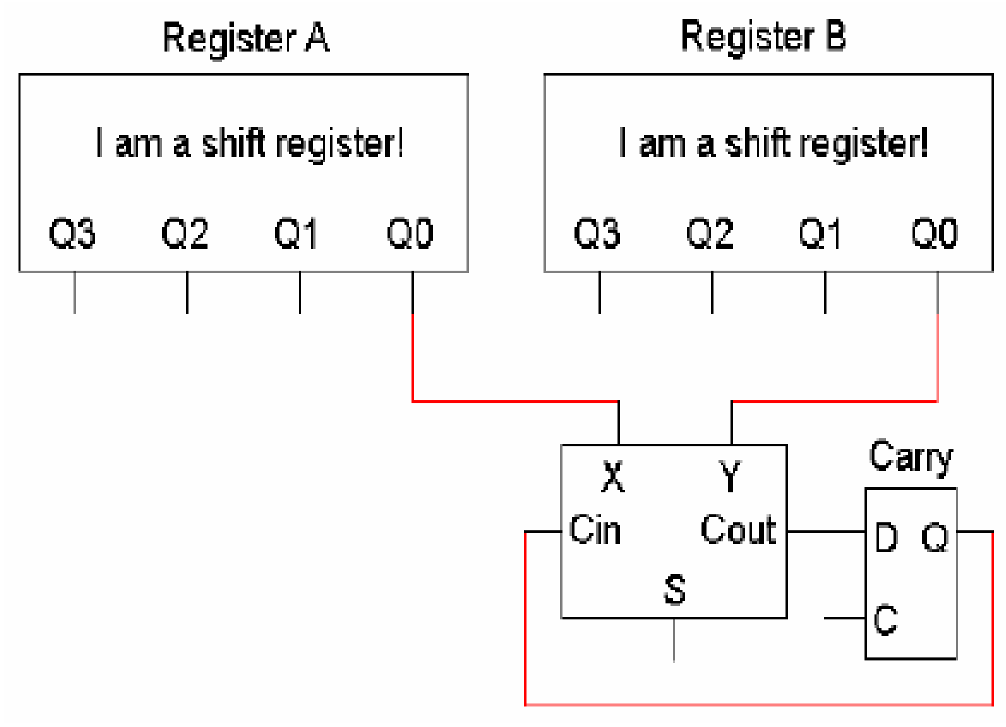
The basic setup for serial addition

- With shift registers, we can build an *n*-bit adder using only *one* full adder.
  - Inputs A and B are contained in shift registers.
  - Initially, the full adder computes A0 + B0.
  - On successive clock cycles, the values A and B are shifted to the right, so the adder computes A1 + B1, A2 + B2, etc.
  - The output S appears serially, one bit (S0, S1, S2, S3) per cycle.



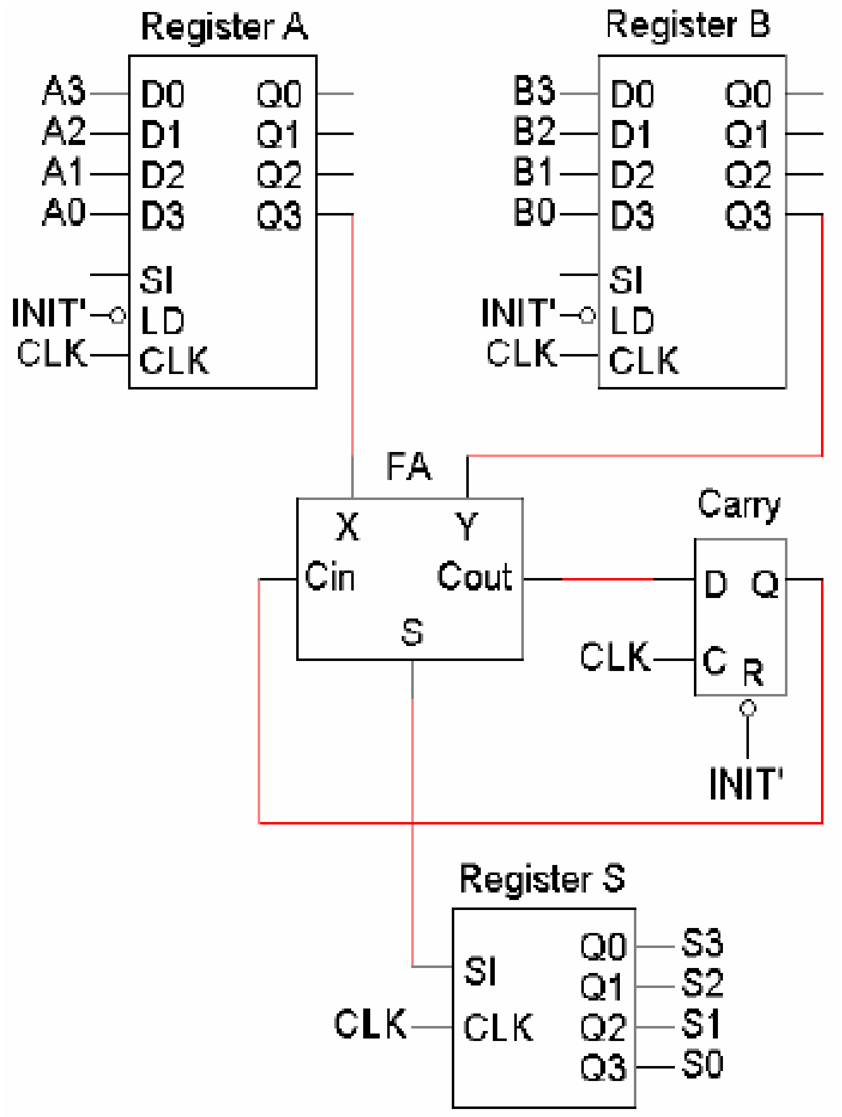
What about the carry?

- The carry out from one stage has to be added in the next stage.
- We need to add a D flip-flop as shown, so the carry out from one clock cycle is saved and used as the carry in for the next cycle.



The big unit

- First, set INIT = 1 for one clock cycle. This loads the initial values of A and B into the shift registers on top, and sets the D flip-flop to 0 (the initial carry in).
- When INIT = 0, the registers will begin shifting, and the full adder results will be written to register S one bit at a time.
- The addition is completed after four clock cycles. The sum is stored in S, and the carry out is in the D flip-flop.



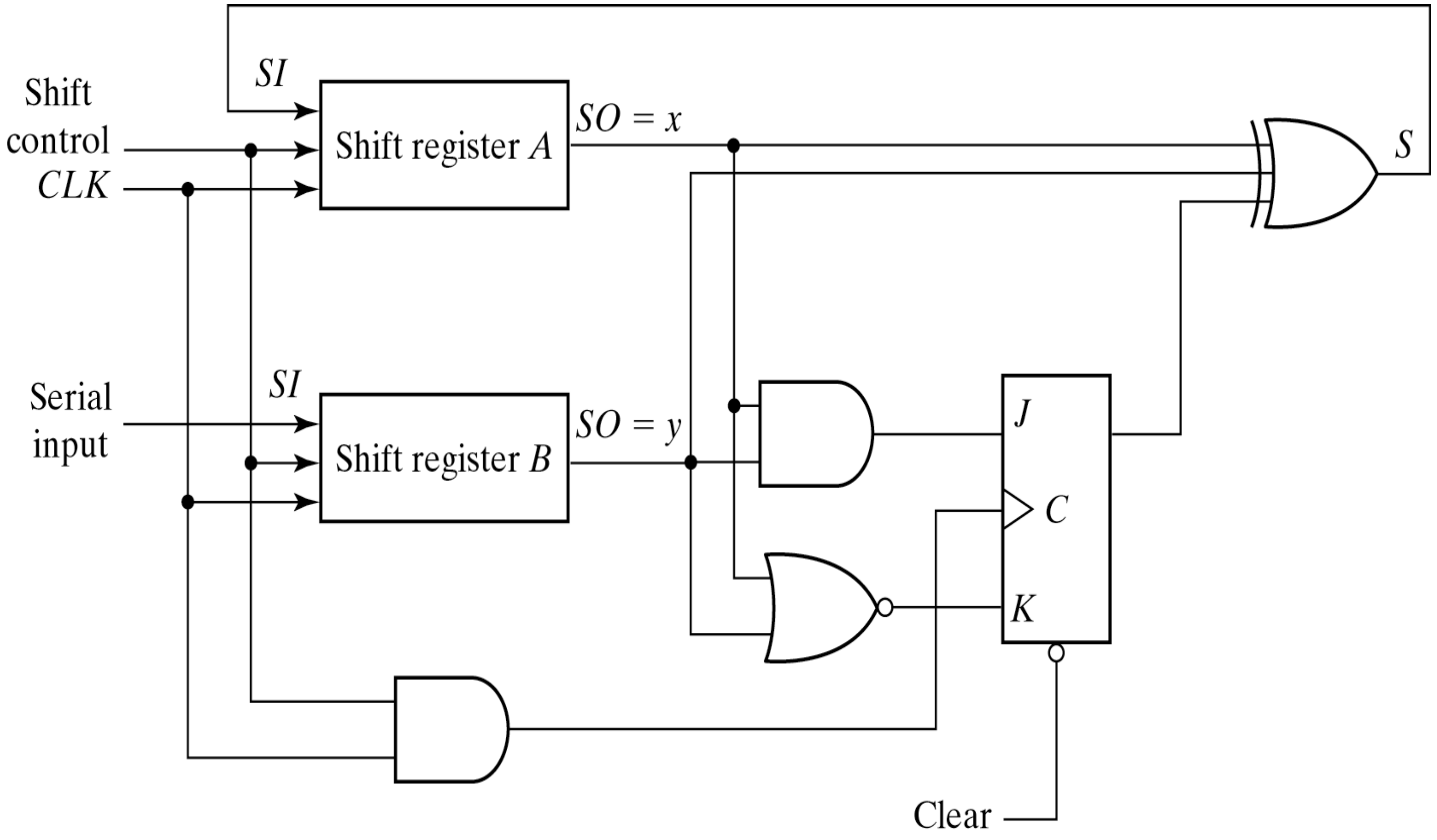


Fig. 6-6 Second form of Serial Adder

Serial addition: the good, the bad and the ugly

- There are several good things about these serial adders.
  - Only one full adder is needed, regardless of the length of the numbers being added. This can save a lot of circuitry.
  - Similar ideas can be applied to make serial multipliers, but with even more hardware savings.
- But there are some bad things too.
  - Adding two *n*-bit numbers takes *n* cycles, but in real processors we'd normally want the addition to be done in just one cycle.
  - Combinational circuits can use more efficient carry-lookahead adders instead of doing things purely sequentially.

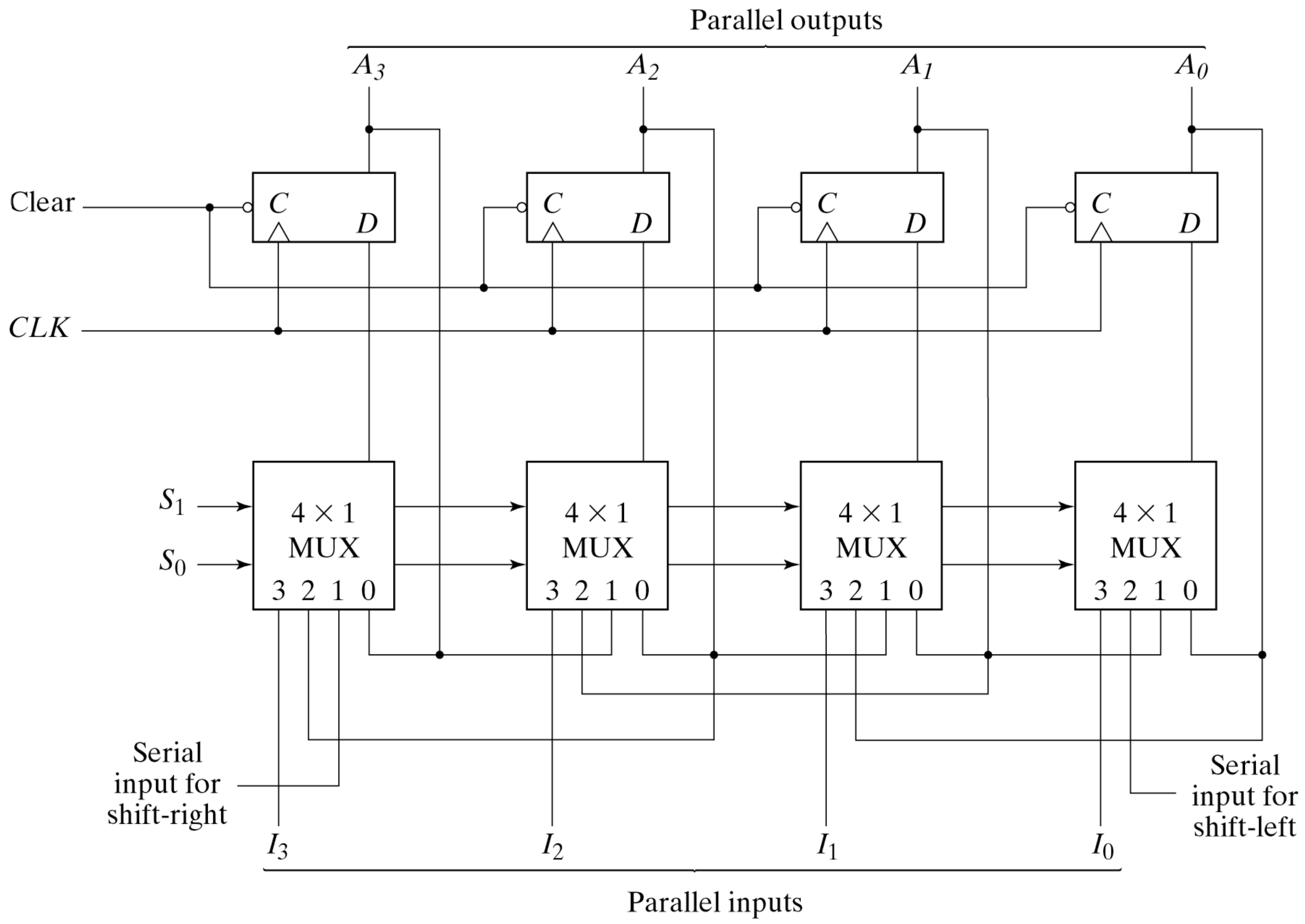


Fig. 6-7 4-Bit Universal Shift Register