Counters and Registers

Dr. Anurag Srivastava

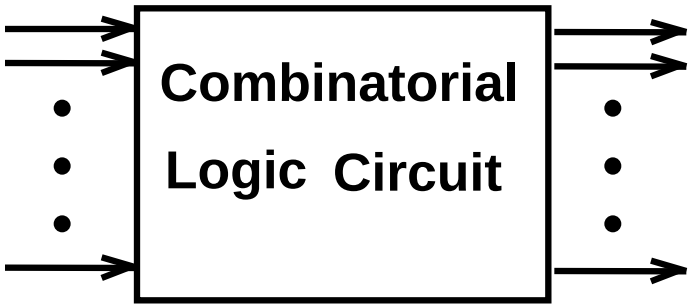
Basic of digital design

- Basic gates (AND ,OR, NOR, NAND and INVERTER)
- Universal gates (NAND and NOR)
- Design of any function using basic gates
- F=xy+(xyz+zx+xz)xy
- Truth table verification

- K-map for solving a complex functions

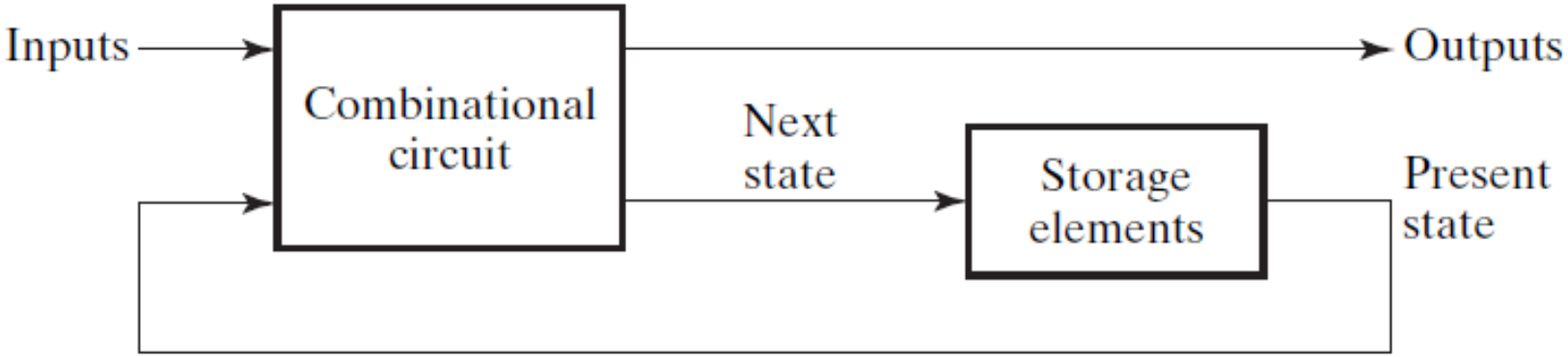
- Difference between the Combinational circuits and Sequential Circuits

Combinational functional blocks



m Boolean Inputs

n Boolean Outputs



Block of sequential

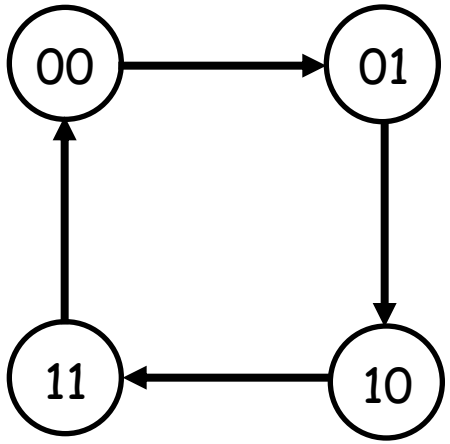
Introducing counters

- Counters are a specific type of sequential circuit
- The state serves as the "output" (Moore)
- A counter that follows the binary number sequence is called a binary counter
  - n-bit binary counter: n flip-flops, count in binary from 0 to 2^n-1
- Counters are available in two types:
  - Synchronous Counters
  - Ripple Counters
- Synchronous Counters:
  - A common clock signal is connected to the C input of each flip-flop

Synchronous Binary Up Counter

- The output value increases by one on each clock cycle
- After the largest value, the output "wraps around" back to 0
- Using two bits, we'd get something like this:

Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

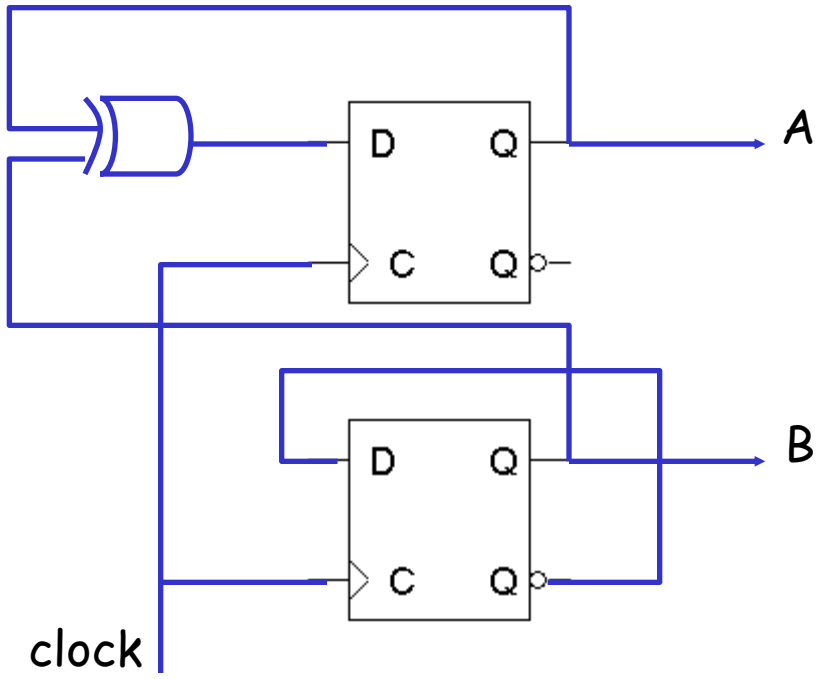


Synchronous Binary Up Counter

Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

D1= A'B + AB'

D0= B'

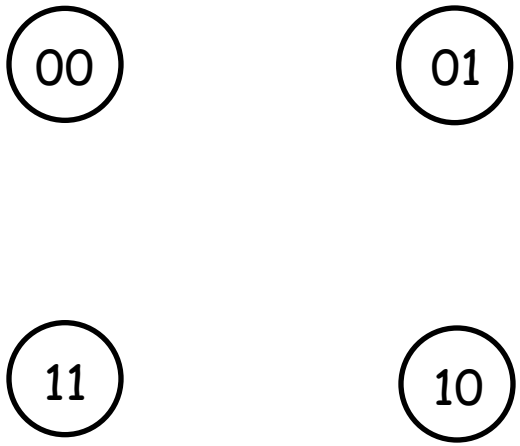


What good are counters?

- Counters can act as simple clocks to keep track of "time"
- You may need to record how many times something has happened
  - How many bits have been sent or received?
  - How many steps have been performed in some computation?
- All processors contain a program counter, or PC
  - Programs consist of a list of instructions that are to be executed one after another (for the most part)
  - The PC keeps track of the instruction currently being executed
  - The PC increments once on each clock cycle, and the next program instruction is then executed.

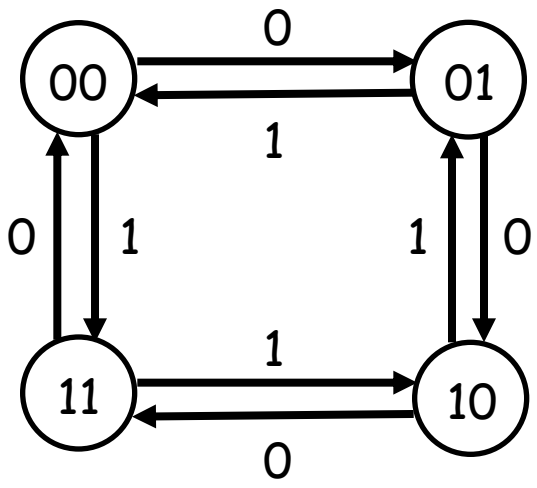
Synch Binary Up/Down Counter

- 2-bit Up/Down counter
  - Counter outputs will be 00, 01, 10 and 11
  - There is a single input, X.
    - X= 0, the counter counts up
    - X= 1, the counter counts down
- We'll need two flip-flops again. Here are the four possible states:



The complete state diagram and table

- Here's the complete state diagram and state table for this circuit



Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

D flip-flop inputs

- If we use D flip-flops, then the D inputs will just be the same as the desired next states
- Equations for the D flip-flop inputs are shown at the right
- Why does $D_0 = Q_0'$ make sense?

Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

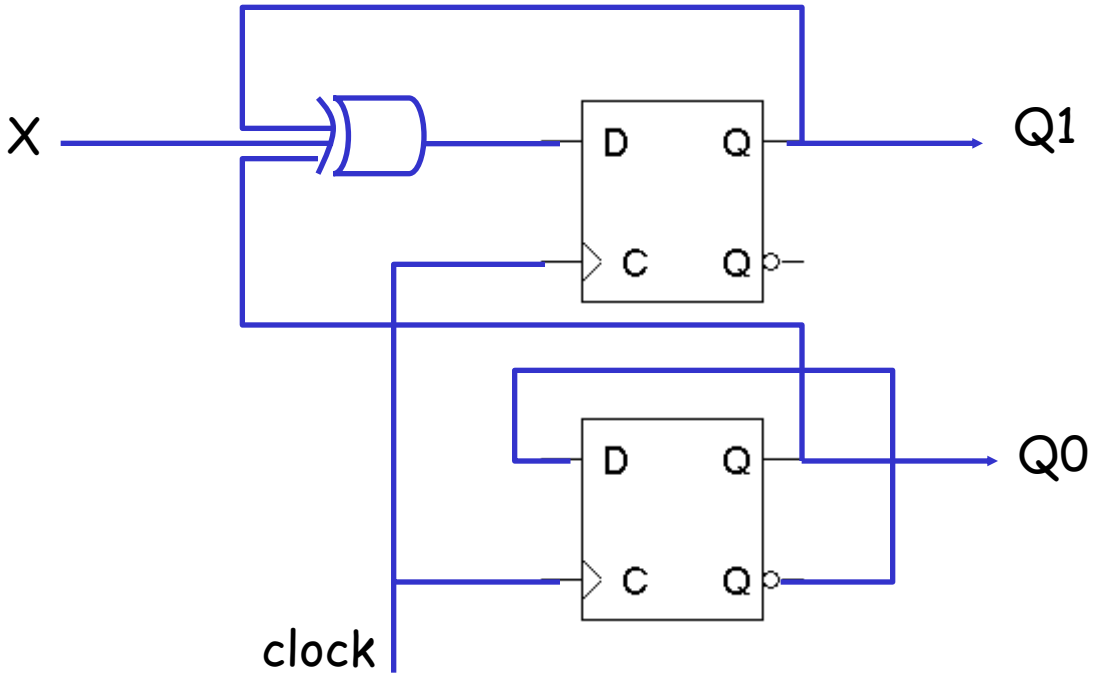
Q_1 \ Q_0	0	1	0	1
1	0	1	1	0
		X		

$D_1 = Q_1 \oplus Q_0 \oplus X$

Q_1 \ Q_0	1	1	0	0
1	1	1	0	0
		X		

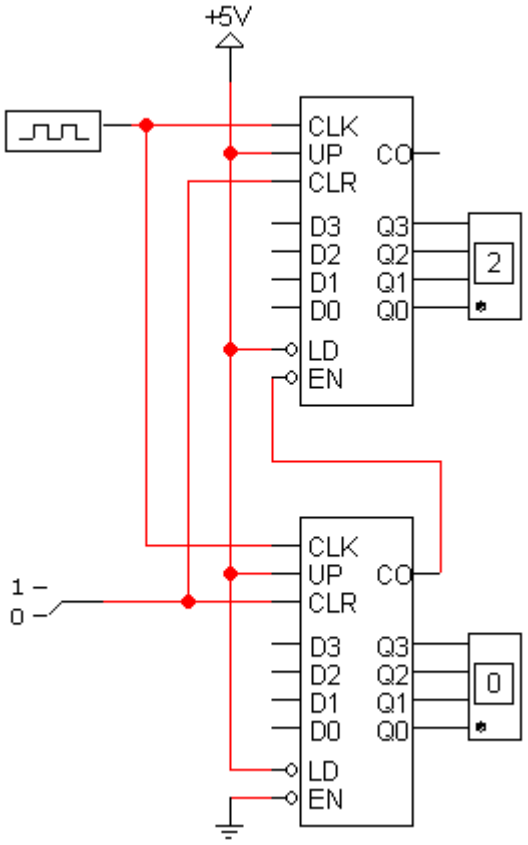
$D_0 = Q_0'$

Synchronous Binary Up/Down Counter

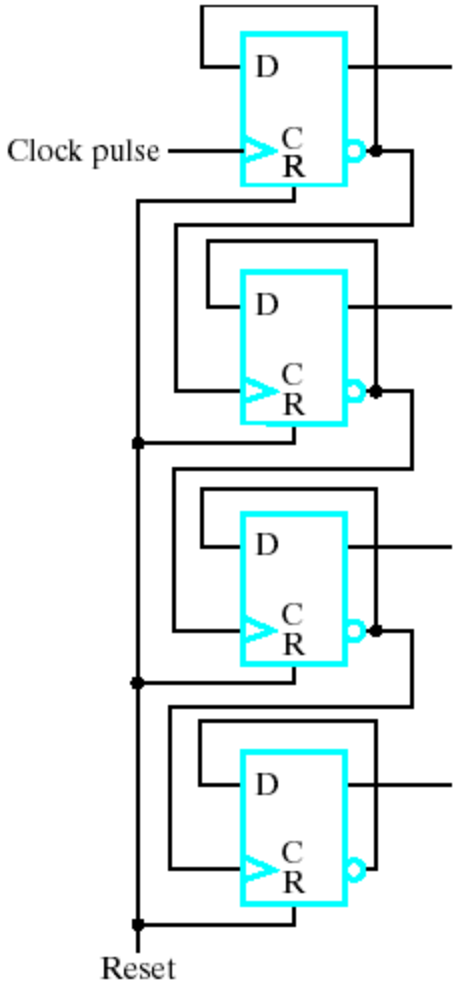


An 8-bit counter

- As you might expect by now, we can use these general counters to build other counters
- Here is an 8-bit counter made from two 4-bit counters
  - The bottom device represents the least significant four bits, while the top counter represents the most significant four bits
  - When the bottom counter reaches 1111 (i.e., when *CO* = 0), it enables the top counter for one cycle
- Other implementation notes:
  - The counters share clock and clear signals
  - Hex displays are used here



Ripple Counter



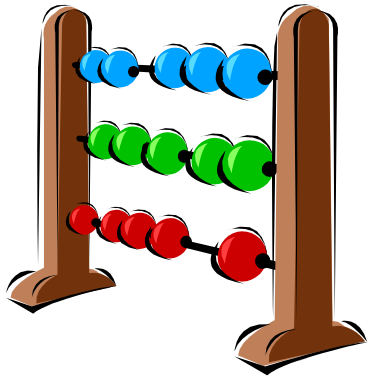
Upward Counting Sequence

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

Simple, yet asynchronous circuits !!!

Summary

- Counters serve many purposes in sequential logic design
- There are lots of variations on the basic counter
  - Some can increment or decrement
  - An enable signal can be added
  - The counter's value may be explicitly set



- There are also several ways to make counters
  - You can follow the sequential design principles to build counters from scratch
  - You could also modify or combine existing counter devices

Registers

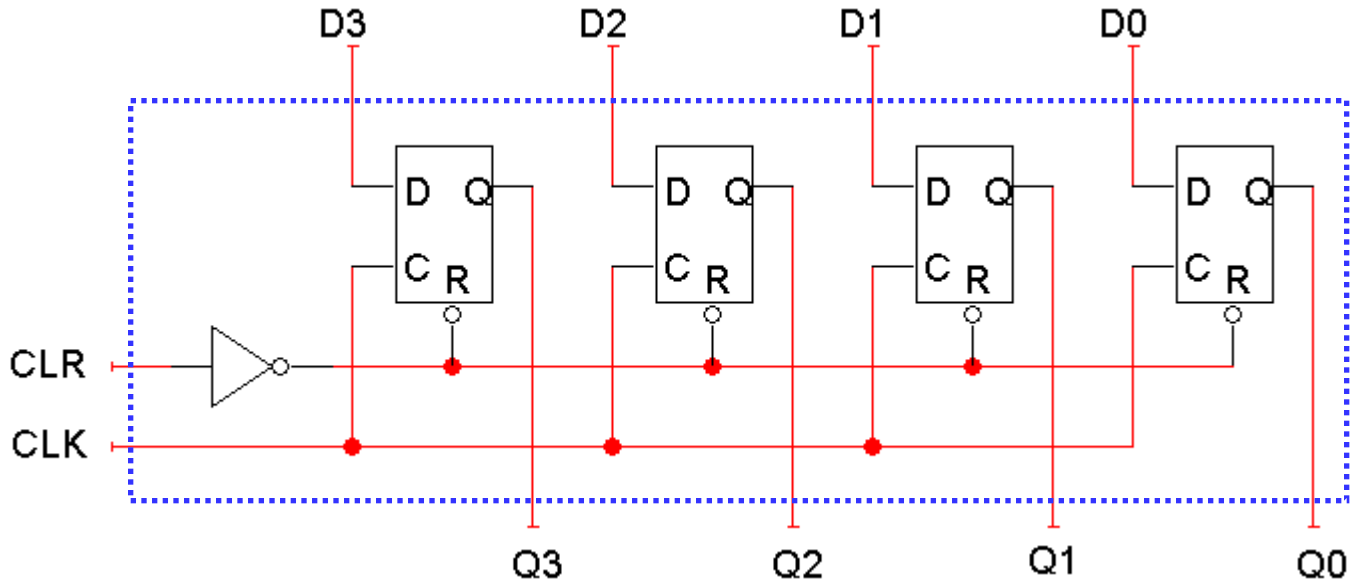
- A common sequential device: Registers
  - They're a good example of sequential analysis and design
  - They are also frequently used in building larger sequential circuits
- Registers hold larger quantities of data than individual flip-flops
  - Registers are central to the design of modern processors
  - There are many different kinds of registers
  - We'll show some applications of these special registers

What good are registers?

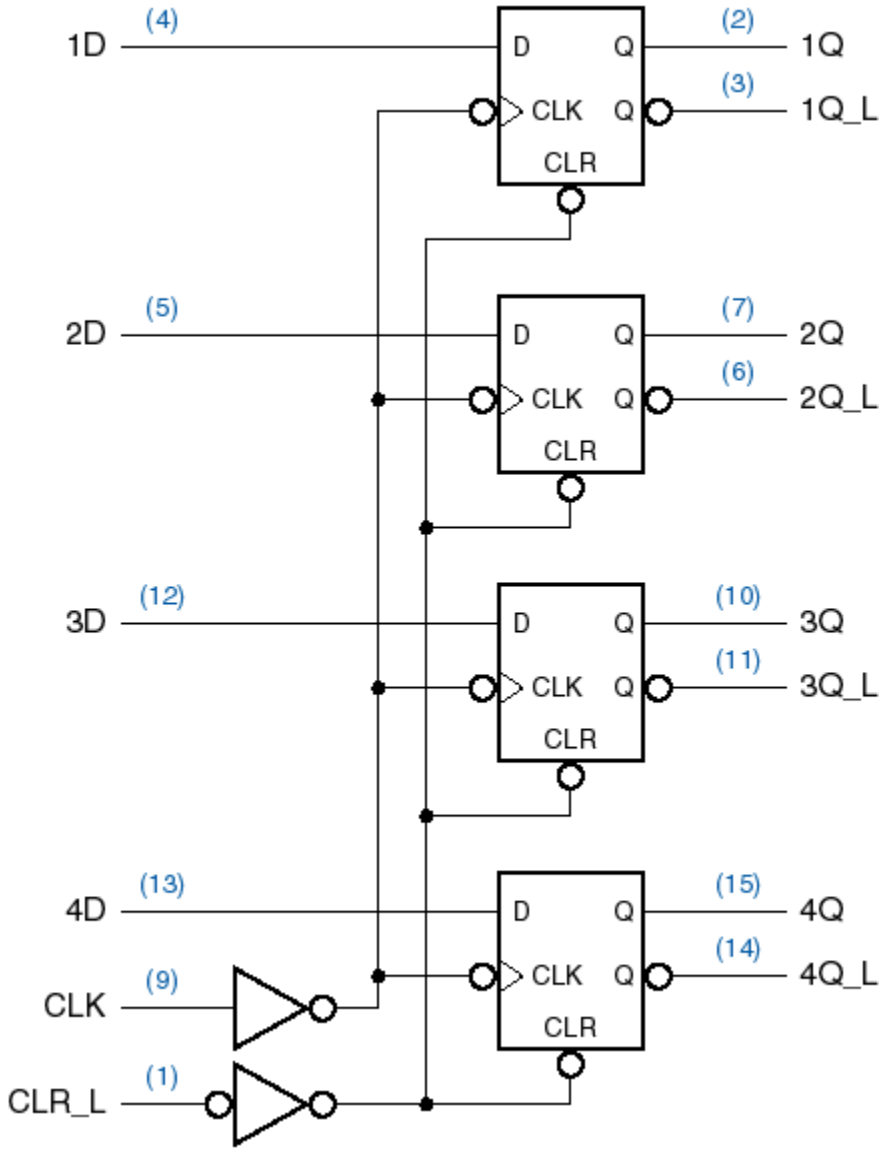
- Flip-flops are limited because they can store only one bit
  - We had to use two flip-flops for our two-bit counter examples
  - Most computers work with integers and single-precision floating-point numbers that are 32-bits long
- A register is an extension of a flip-flop that can store multiple bits
- 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

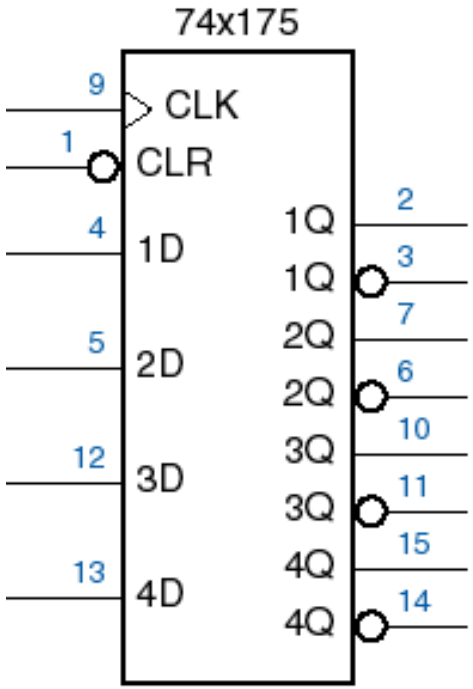
A basic register

- 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 given 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



74x175 – 4-bit register

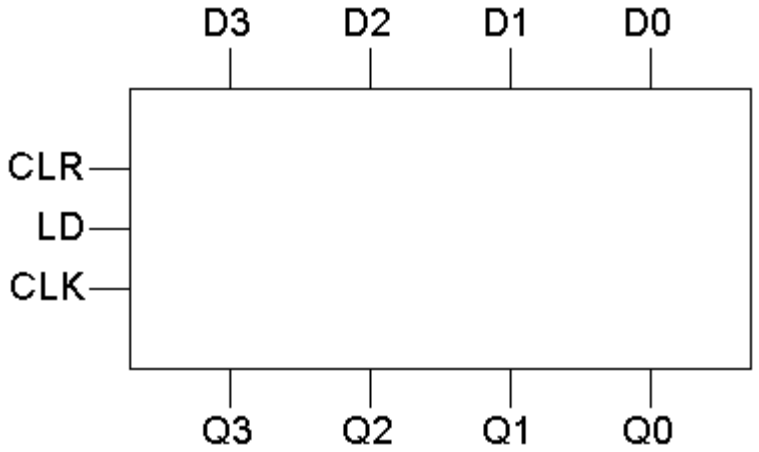




Adding a parallel load operation

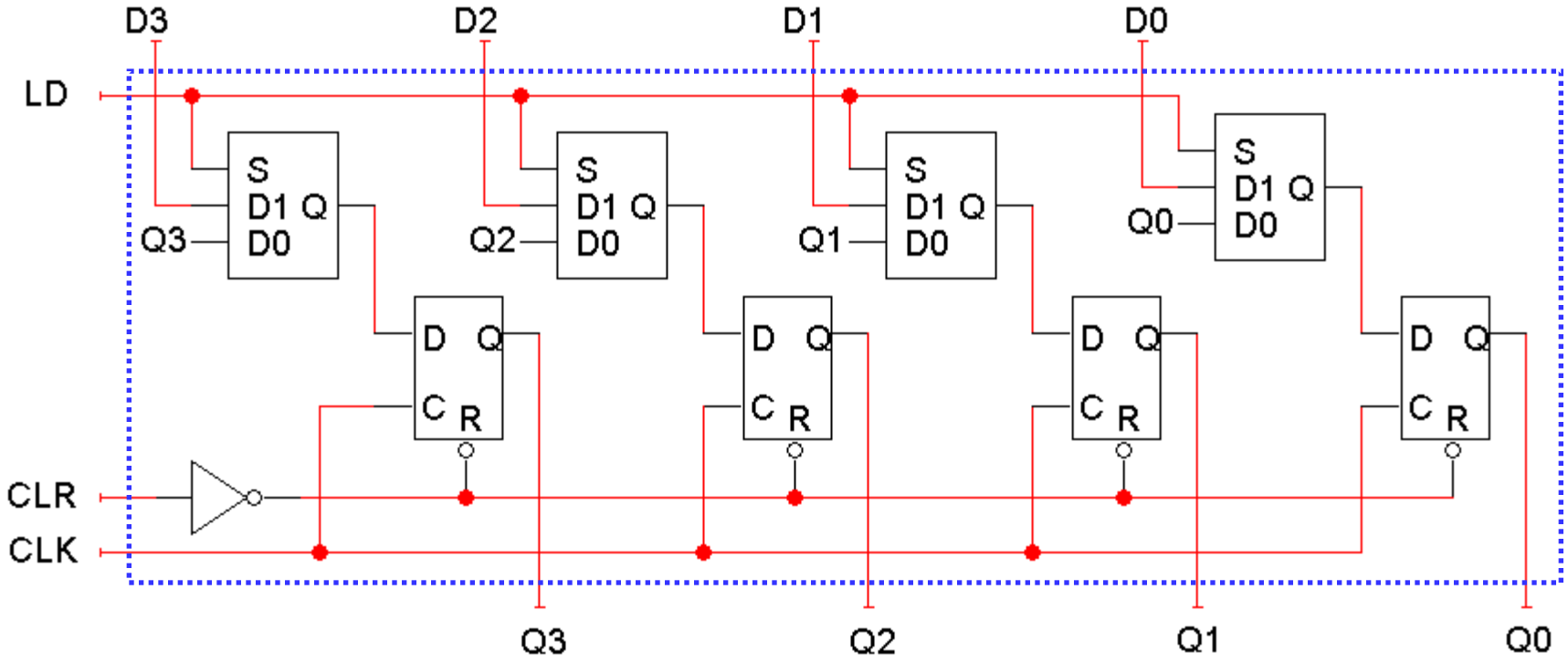
- The input D_3-D_0 is copied to the output Q_3-Q_0 on *every* clock cycle
- How can we store the current value for more than one cycle?
- Let's add a load input signal LD to the register
  - If LD = 0, the register keeps its current contents
  - If LD = 1, the register stores a new value, taken from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

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



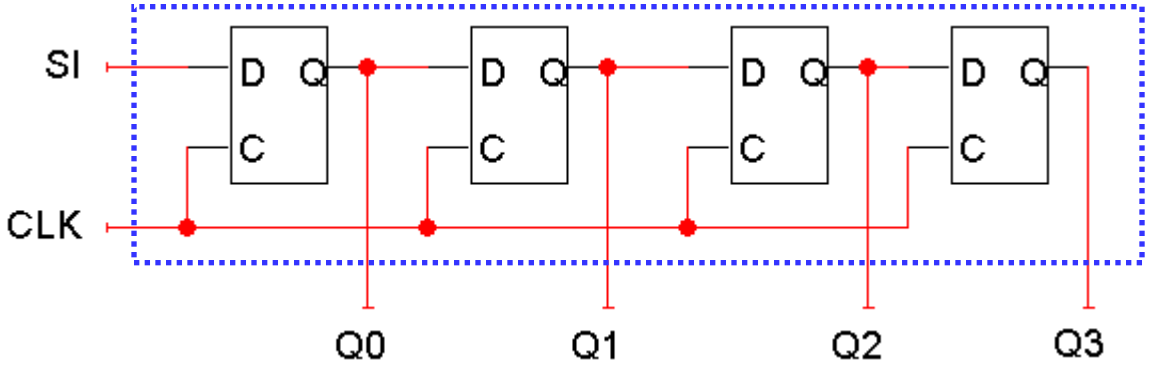
Register with parallel load

- 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 Register

- A shift register "shifts" its output once every clock cycle.



$Q_0(t+1) = SI$
$Q_1(t+1) = Q_0(t)$
$Q_2(t+1) = Q_1(t)$
$Q_3(t+1) = Q_2(t)$

- SI is an input that supplies a new bit to shift "into" the register
- For example, if on some positive clock edge we have:

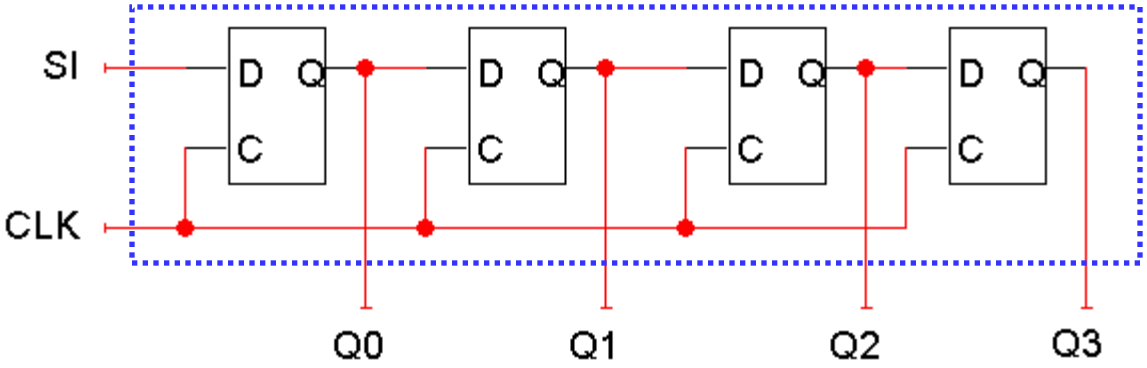
$$SI = 1$$
$$Q_0\text{-}Q_3 = 0110$$

then the next state will be:

$$Q_0\text{-}Q_3 = 1011$$

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

Shift direction



$Q_0(t+1) = SI$
$Q_1(t+1) = Q_0(t)$
$Q_2(t+1) = Q_1(t)$
$Q_3(t+1) = Q_2(t)$

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

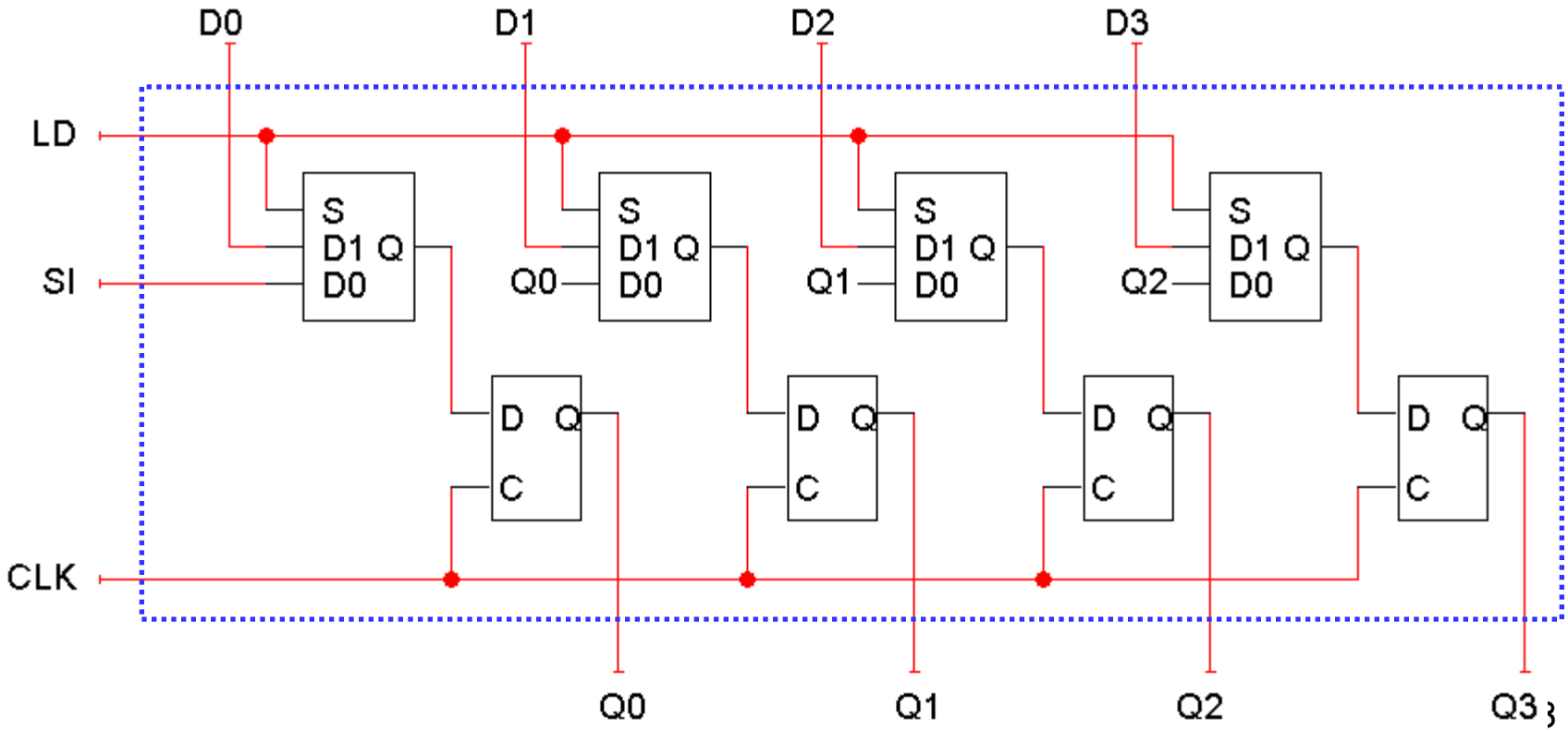
Present Q_0-Q_3	SI	Next Q_0-Q_3
ABCD	X	XABC

- 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 register 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

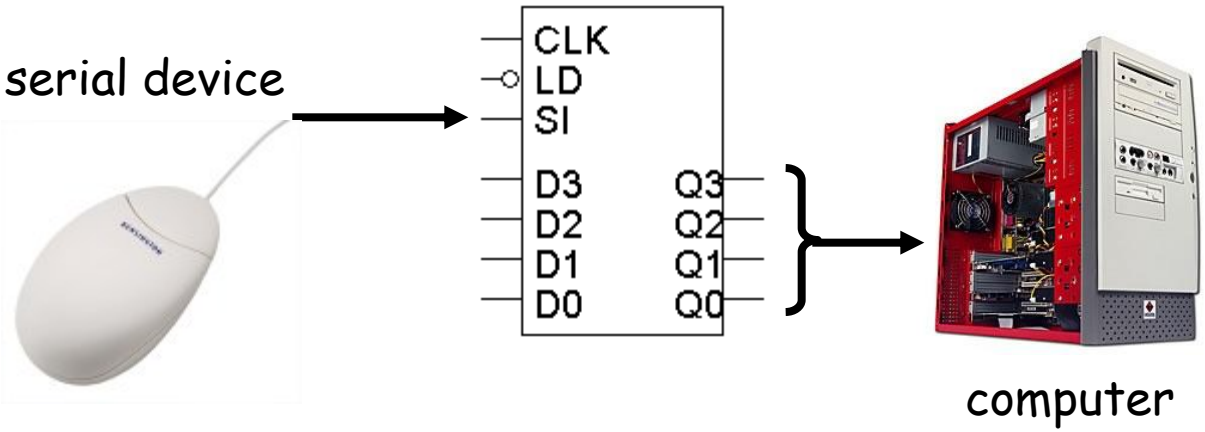


Serial data transfer

- 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 keyboards and mice
  - 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

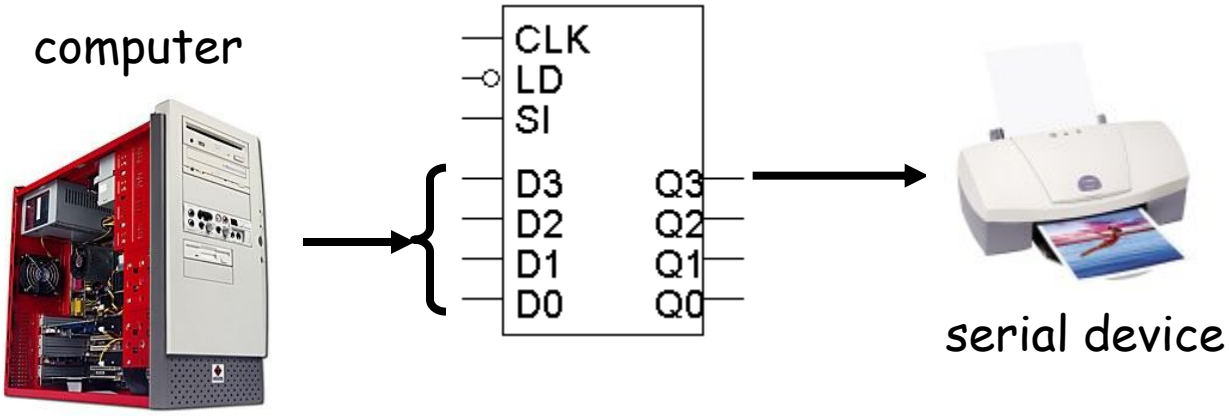
Receiving serial data

- 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.

Registers summary

- A register is a special state machine that stores multiple bits of data
- Several variations are possible:
  - Parallel loading to store data into the register
  - Shifting the register contents either left or right
  - Counters are considered a type of register too!
- One application of shift registers is converting between serial and parallel data
- Most programs need more storage space than registers provide
  - We'll introduce RAM to address this problem
- Registers are a central part of modern processors