SEQUENTIAL CIRCUITS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

THE UNIVERSITY OF UTAH

CS/ECE 3810: Computer Organization

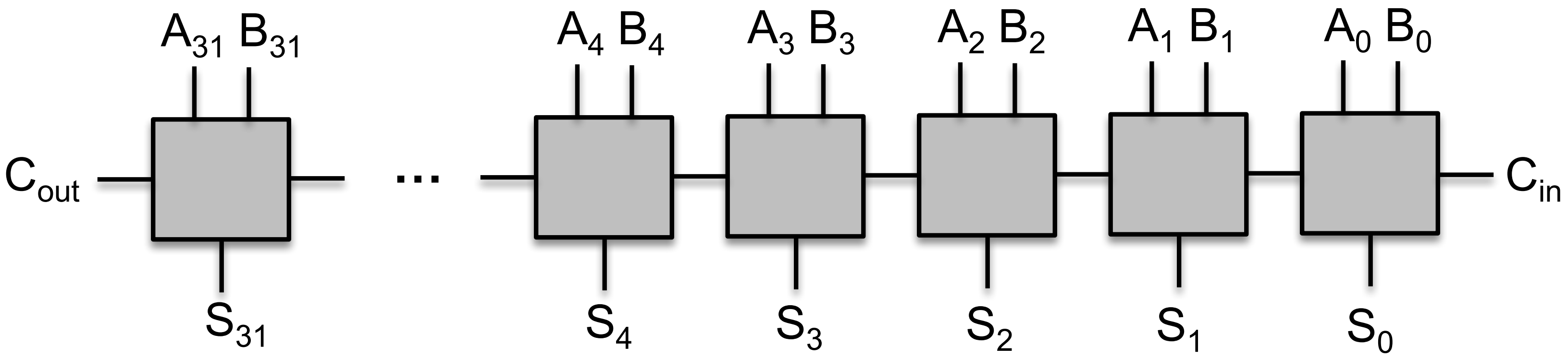
Overview

- Notes
  - Homework 7 is due March 21st
    - Verify your submitted file before midnight

- This lecture
  - Carry look-ahead Adder
  - Clock and sequential circuits

Recall: Carry Ripple Adder

- Simplest design by cascading 1-bit boxes
  - Each 1-bit box sequentially implements AND and OR
  - Critical path: the total delay is the time to go through 64 gates
- How to make a 32-bit addition faster?



Carry Ripple Adder

- Simplest design by cascading 1-bit boxes
  - Each 1-bit box sequentially implements AND and OR
  - Critical path: the total delay is the time to go through 64 gates
- How to make a 32-bit addition faster?
- Recall: any logic equation can be expressed as the sum of products (only 2 gate levels!)
  - Challenges: many parallel gates with very large inputs
  - Solution: we'll find a compromise

Fast Adder

- Computing carry-outs
  - CarryIn1 = b0.CarryIn0 + a0.CarryIn0 + a0.b0
  - CarryIn2 = b1.CarryIn1 + a1.CarryIn1 + a1.b1
  - = b1.b0.c0 + b1.a0.c0 + b1.a0.b0 +
  - a1.b0.c0 + a1.a0.c0 + a1.a0.b0 + a1.b1
  - …
  - CarryIn32 = a really large sum of really large products
- Each gate is enormous and slow!

Fast Adder

- Computing carry-outs: equation re-phrased
- $C_{i+1} = a_i.b_i + a_i.C_i + b_i.C_i$
- $= (a_i.b_i) + (a_i + b_i).C_i$

- Generate signal = $a_i.b_i$
  - The current pair of bits will generate a carry if they are both 1
- Propagate signal = $a_i + b_i$
  - The current pair of bits will propagate a carry if either is 1

- Therefore, $C_{i+1} = G_i + P_i \,.\, C_i$

Fast Adder

□ Computing carry-outs: example

$$c1 = g0 + p0.c0$$

$$c2 = g1 + p1.c1 \ = g1 + p1.g0 + p1.p0.c0$$

$$c3 = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0$$

$$c4 = \underbrace{g3}_{(1)} + \underbrace{p3.g2}_{(2)} + \underbrace{p3.p2.g1}_{(3)} + \underbrace{p3.p2.p1.g0}_{(4)} + \underbrace{p3.p2.p1.p0.c0}_{(4)}$$

Either,

(1) a carry was just generated, or

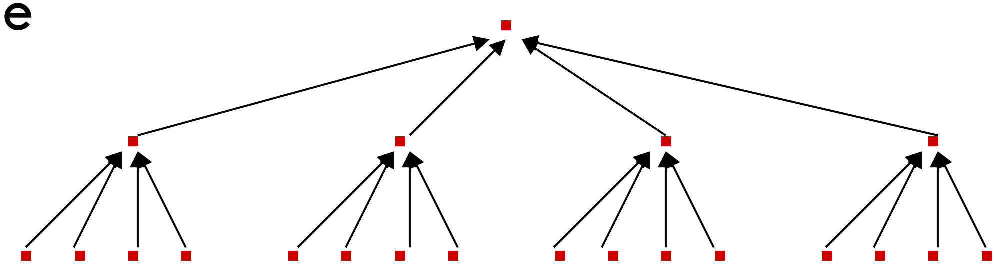
(2) a carry was generated in the last step and was propagated, or

(3) a carry was generated two steps back and was propagated by both the next two stages, or

(4) a carry was generated N steps back and was propagated by every single one of the N next stages

Fast Adder

- Divide and Conquer
  - Challenge: for the 32nd bit, we must AND every single propagate bit to determine what becomes of c0 (among other things)
  - Solution: the bits are broken into groups (of 4) and each group computes its group-generate and group-propagate
  - For example, to add 32 numbers, you can partition the task as a tree



Fast Adder

- P and G for 4-bit blocks
  - Compute P0 and G0 (super-propagate and super-generate) for the first group of 4 bits (and similarly for other groups of 4 bits)
    - P0 = p0.p1.p2.p3
    - G0 = g3 + g2.p3 + g1.p2.p3 + g0.p1.p2.p3

  - Carry out of the first group of 4 bits is
    - C1 = G0 + P0.c0
    - C2 = G1 + P1.G0 + P1.P0.c0
    - C3 = G2 + (P2.G1) + (P2.P1.G0) + (P2.P1.P0.c0)
    - C4 = G3 + (P3.G2) + (P3.P2.G1) + (P3.P2.P1.G0) + (P3.P2.P1.P0.c0)

  - By having a tree of sub-computations, each AND, OR gate has few inputs and logic signals have to travel through a modest set of gates (equal to the height of the tree)

Fast Adder

- Example

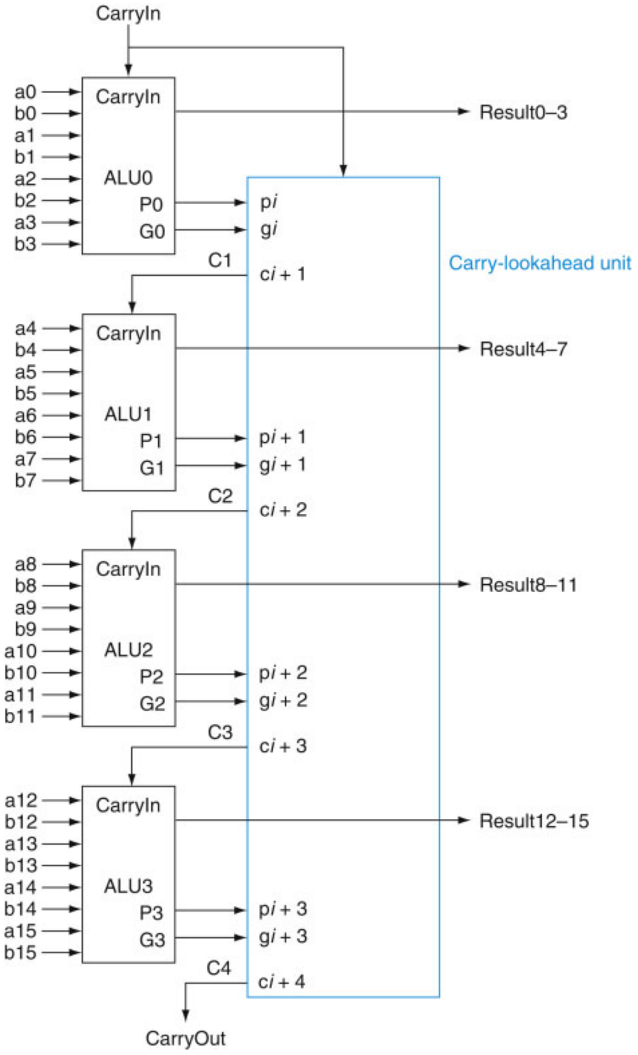
Add	A	0001	1010	0011	0011
	B	1110	0101	1110	1011
	g	0000	0000	0010	0011
	p	1111	1111	1111	1011
	P	1	1	1	0
	G	0	0	1	0

C4 = 1

Fast Adder: Carry Look-Ahead

- 16-bit Ripple-carry takes 32 steps

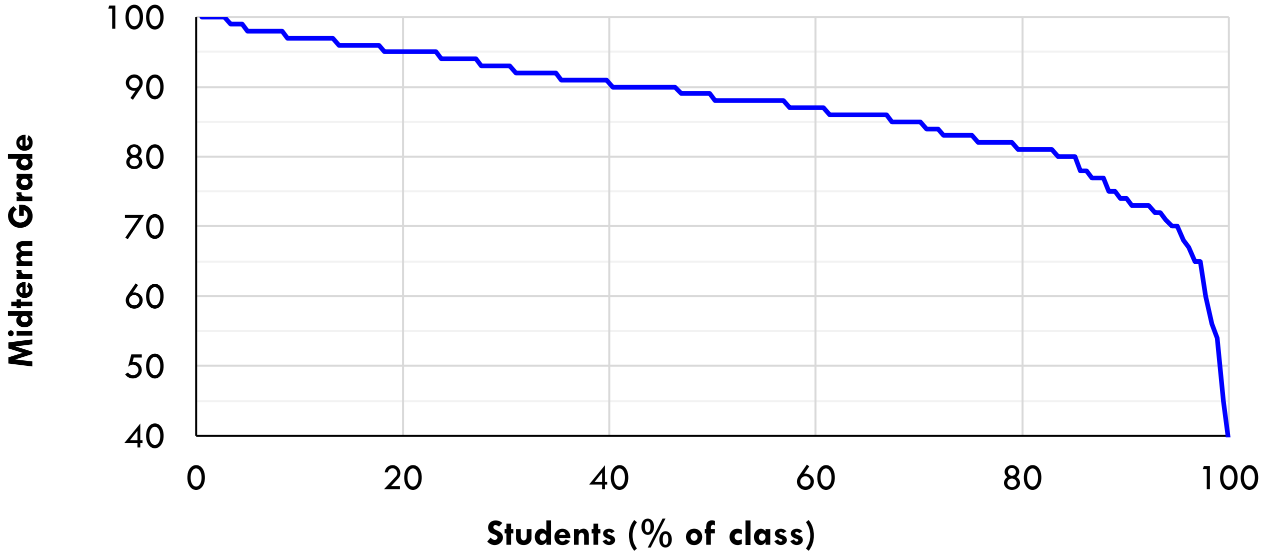
- This design takes how many steps?



Break: Remarks on Your Midterm

- Overall, the class did well!
  - 22% of the class earned 95 and above.
    - Five students got 100/100!
  - The average is ~87.

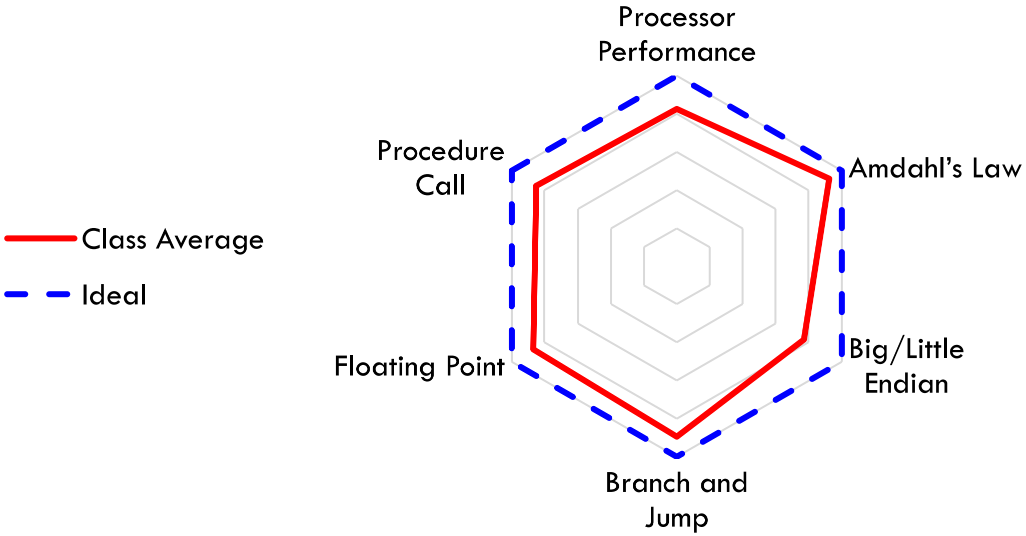
- Class overview (midterm exam only)



Break: Remarks on Your Midterm

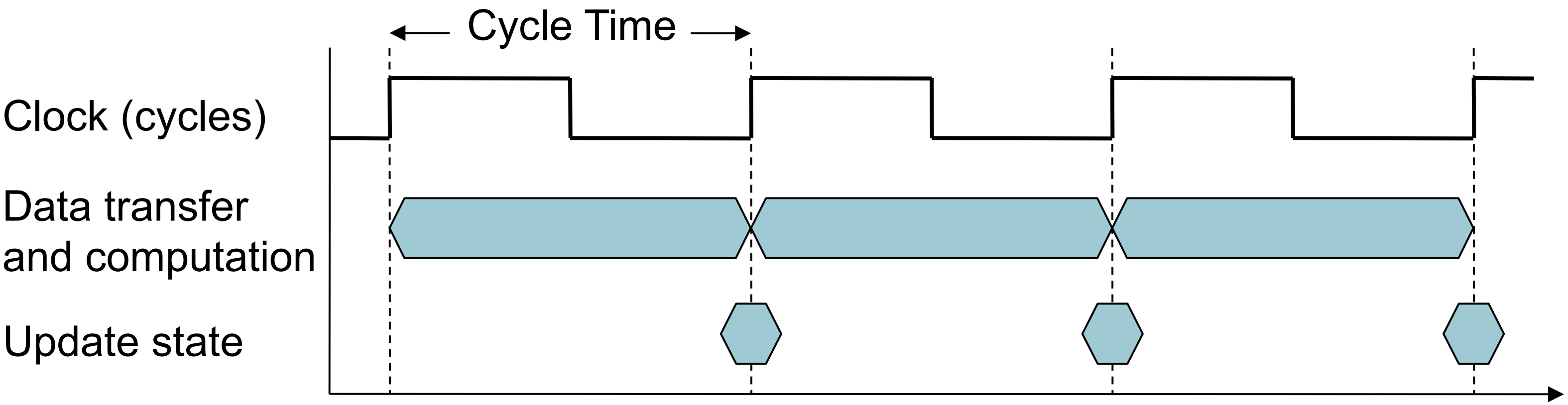
- Question: how to improve the final grade?
  - 1. Come to class regularly if you do not ☺
  - 2. I typically do a final upgrade if I see significant improvements in your grades after the midterm
  - 3. I drop one of your HWs with the least grade

- A different view of the class based on your midterm grades



Recall: Clocking and Cycle Time

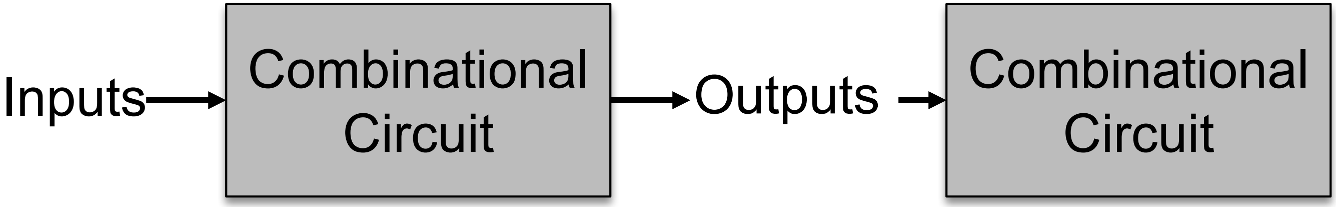
- Operation of digital hardware governed by a constant-rate clock



- A microprocessor consists of many circuits operating simultaneously, each of which
  - takes in inputs at time T_{input},
  - takes time $T_{execute}$ to execute the logic, and
  - produces outputs at time T_{output}

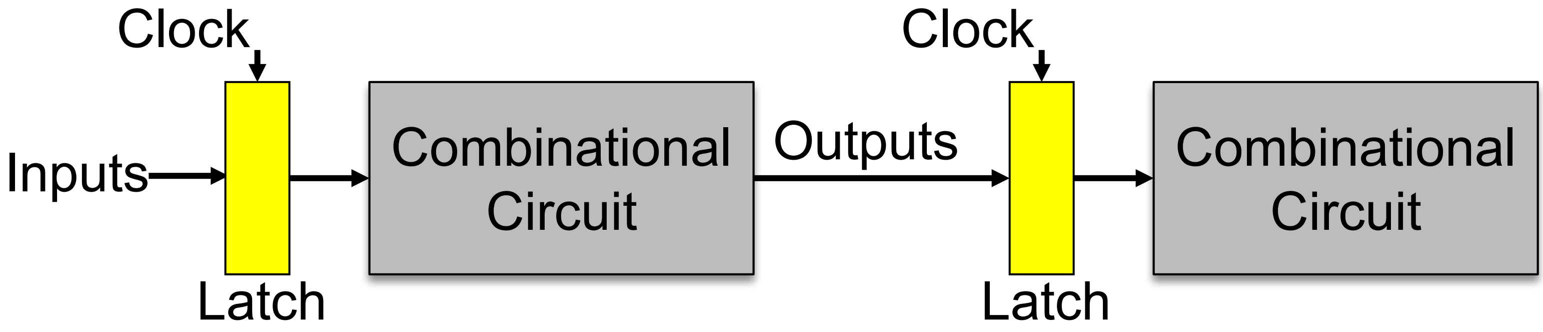
Combinational Circuits

- Circuits we have seen were combinational
  - when inputs change, the outputs change after a while (time = logic delay thru circuit)
  - Example: adder



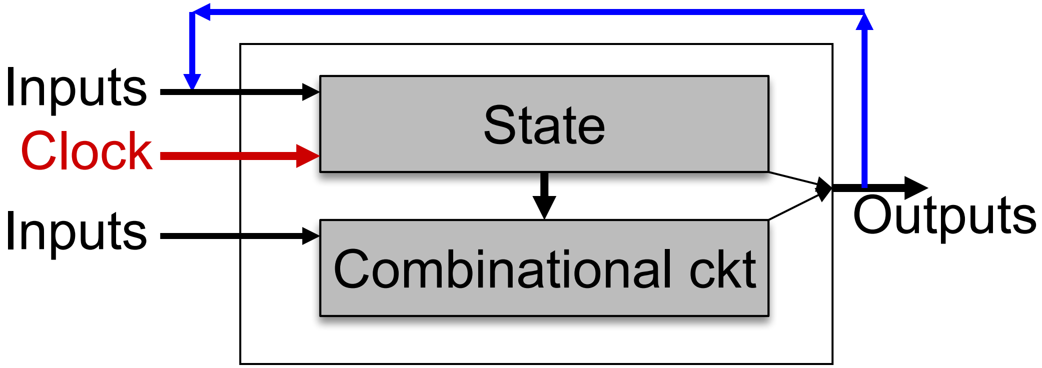
Sequential Circuits

- Sequential circuit consists of combinational circuit and a storage element (latch)
- The clock acts like a start and stop signal
  - The latch ensures that the inputs to the circuit do not change during a clock cycle



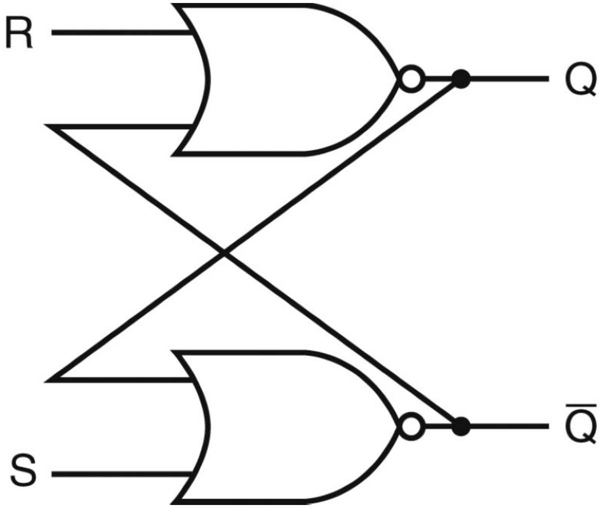
Sequential Circuits

- At the start of the clock cycle, the rising edge causes the “state” storage to store some input values
- This state will not change for an entire cycle
- The combinational circuit has some time to accept the value of “state” and “inputs” and produce “outputs”
- Some of the outputs (for example, the value of next “state”) may feed back (but through the latch so they’re only seen in the next cycle)



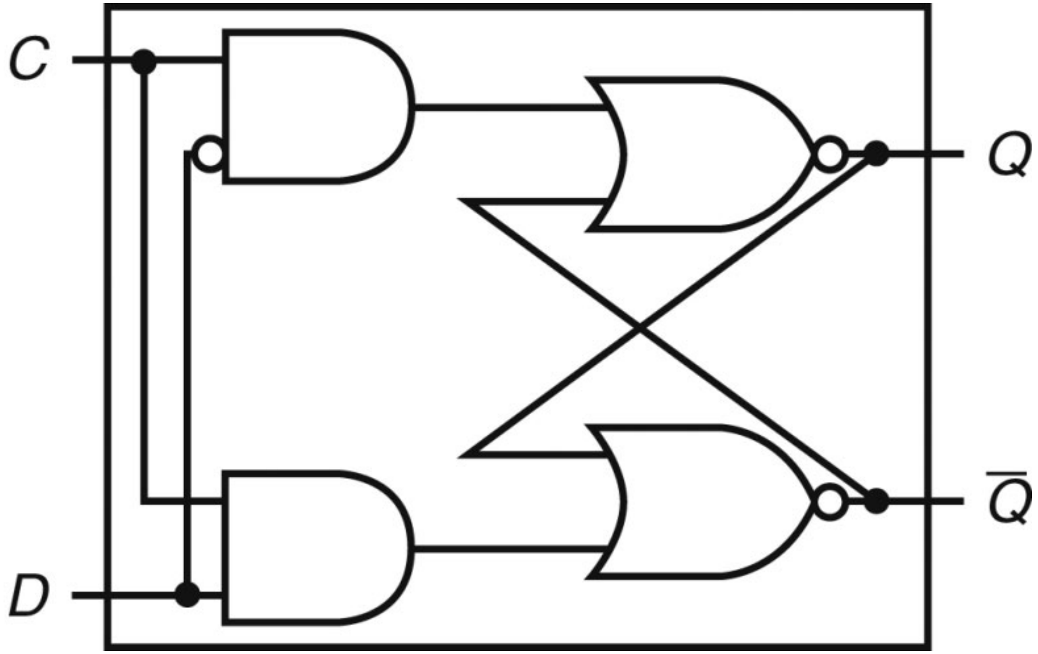
Design of an S-R Latch

- An S-R latch: set-reset latch
  - When Set is high, a 1 is stored
  - When Reset is high, a 0 is stored
  - When both are low, the previous state is preserved (hence, known as a storage or memory element)
  - Both are high – this set of inputs is not allowed



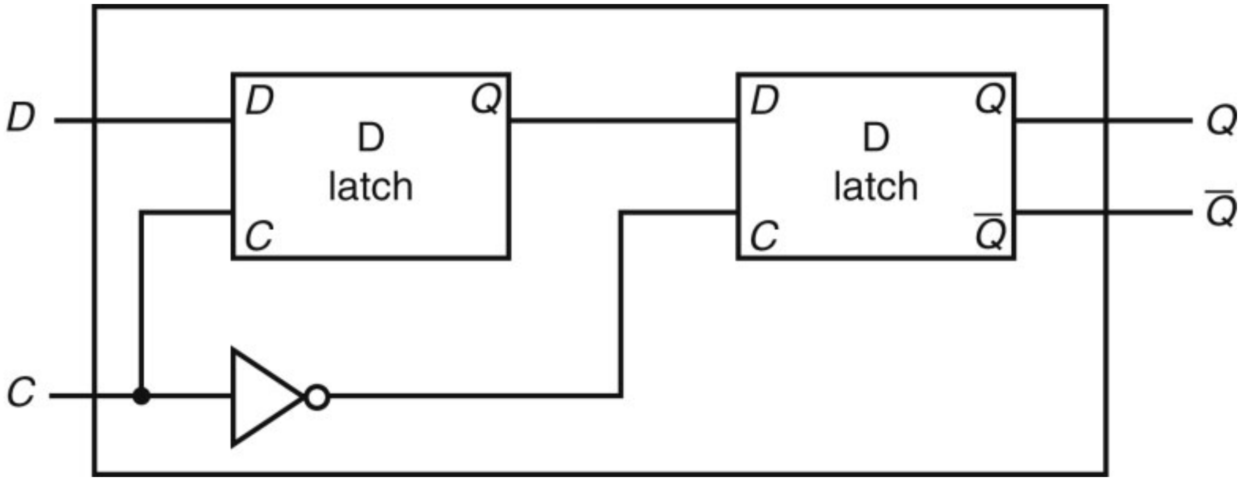
Design of a D Latch

- The value of the input D signal (data) is stored only when the clock is high – the previous state is preserved when the clock is low



Design of a D Flip Flop

- Latch vs. Flip Flop
  - Latch: outputs can change any time the clock is high (asserted)
  - Flip flop: outputs can change only on a clock edge
    - Technically, two D latches in series


D
C
D
Q
D
latch
C
D
Q
D
latch
C
$\overline{Q}$
Q
$\overline{Q}$