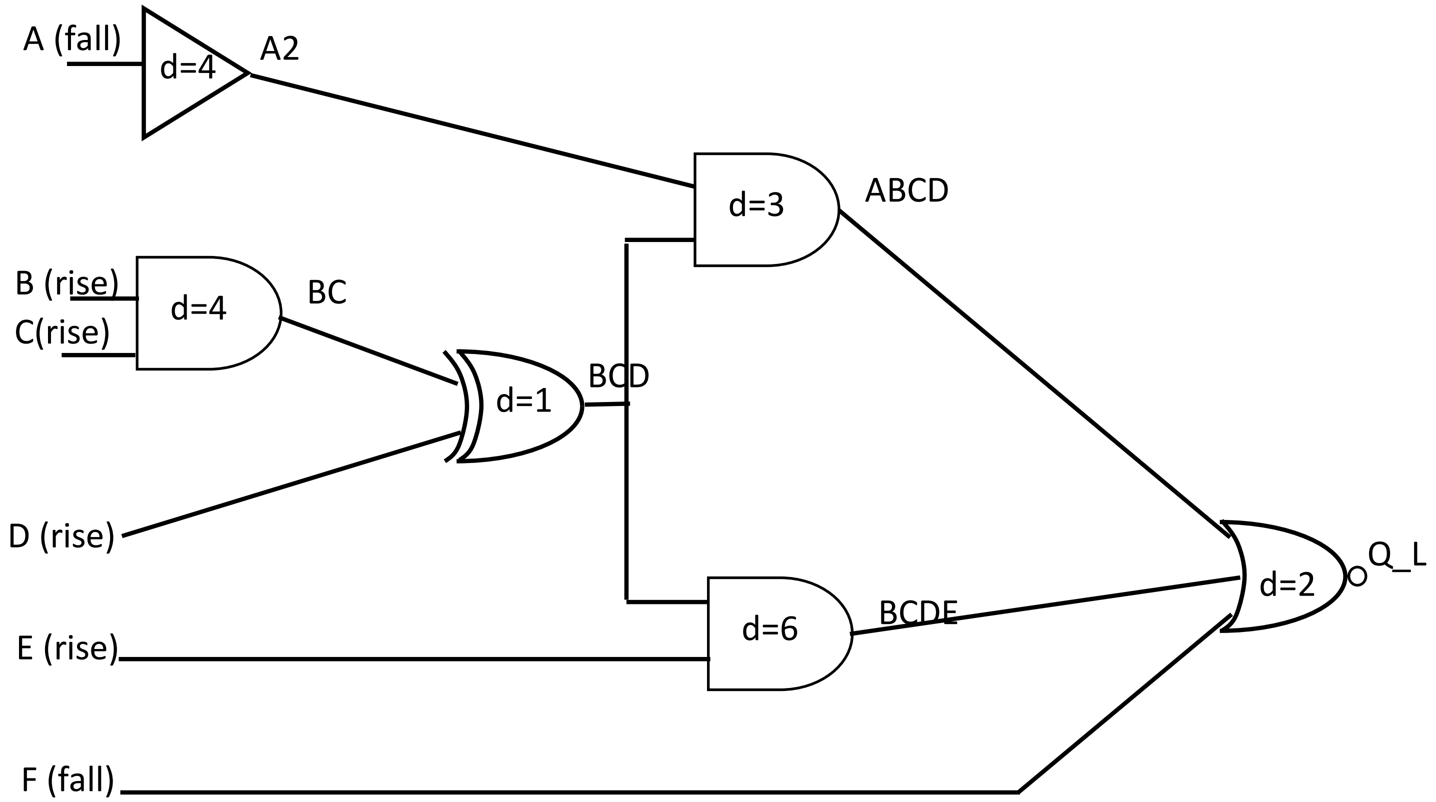
Problem #1. All inputs transition at t=0. Show what events propagate through the system.



In other words, assume that the circuit has first settled in the initial condition (with A=F=1 and B=C=D=E=0). Then at t=0, the inputs change as noted. You thus have a set of initial events at the inputs all stamped with t=0.

Please present your answer using arrows to show which event(s) caused which. For example, you might have

$$\begin{tikzcd} \text{A=0@t=0} \arrow[ddd, bend left] \\ \text{B=1@t=0} \\ \dots \\ \text{A2=0@t=4} \\ \dots \end{tikzcd}$$

Note that people who do the discrete-event programming homework are also simulating this same network. Feel free to check your results against theirs, but please try it on your own first.

Problem 2. We are going to try and make our simulation more efficient in two ways. First, we will use levelized compiled code, as discussed in class.

Second, we are going to take advantage of the fact that a machine word in most computers is 64 bits wide, and use that to simulate multiple input patterns at the same time. We will assign one bit of each variable to a different simulation, and thus be able to run 64 simulations at once. For example, a single simulation for an AND gate C=A&B might just AND 0 and 1 to get 0. However, we could let A=1100 and B=1010. Then bit #0 would have A=0 and B=0, bit #1 would have A=0 and B=1, bit #2 would have A=1 and B=0, and bit #3 would have A=1 and B=1. Our single AND instruction would then compute C=A&B= 1100 & 1010 = 1000. This would tell us that the output (C) is 1 only for the case when A=1 and B=1 (i.e., bit #3), and is 0 for the other three cases (bits #2, #1 and #0).

Assume the same network as in problem #1. In principle, with 6 inputs, we could fit all 64 possible input patterns into one 64-bit word and do them all in one simulation. However, to save your time, we will only simulate the following four input patterns:

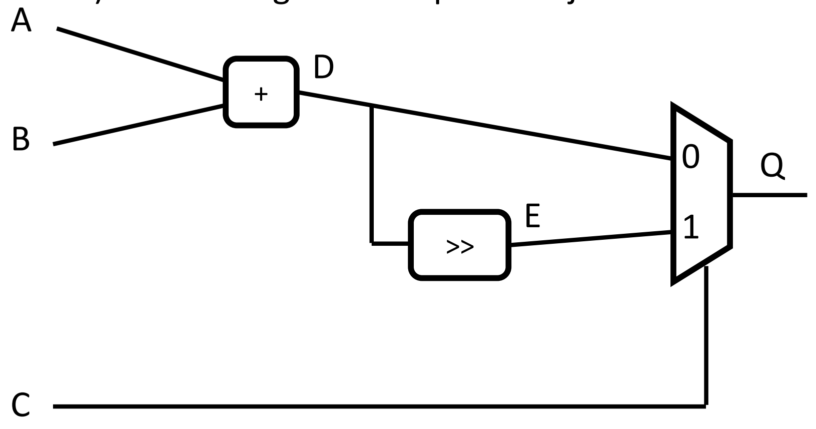
	Pattern #1	Pattern #2	Pattern #3	Pattern #4
A	0	1	1	0
B	0	1	0	1
C	0	1	1	1
D	0	1	1	1
E	0	1	0	0
F	0	1	1	0

Each row of the table is one signal (i.e., one variable). Each column is one input pattern. In your efficient simulation, computing the results of all four input patterns at once, you should use bit #0 for the first pattern, bit #1 for the second pattern, etc. Thus you would need variables that are at least 4 bits long, which of course is easy for a 64-bit machine.

In the table below, show the resulting four-bit values for the inputs A-F as well as the other nodes (BC, BCD, ABCD, BCDE and Q). Note that the values above the thick line are the inputs, for which you assign values; the values below the thick line are the results computed by the simulation. You may find it easier to compute these values with a short computer program in your favorite language and just copy them here.

A	0110
B	0101
C	0111
D	0111
E	0100
F	0110
BC	
BCD	
ABCD	
BCDE	
Q	

Problem 3. Another way to make a simulation run faster is to again take advantage of 64-bit computing, but this time to do it in a different way. Now, we will just run one simulation at a time, but we will simulate large structures with one machine instruction. I.e., we will use “wide objects” (like 64-bit adders or shifters) instead of gates that produce just one bit. Consider the network below:



Assume that A and B are 64-bit inputs, and C is one bit. D, E and Q are also 64 bits wide.

a) Write a short snippet of code (in your favorite language) to compute Q with levelized compiled code (you do not need to actually need to compile and run it). Also, do not bother with overflow or other error conditions.
b) We have just shown how wide objects can speed up levelized compiled code, where exact delays do not really matter. In principle, you could use wide objects for an event-oriented simulation, where delay does matter. What are some reasons why it would or would not make sense? (Consider the fact that, e.g., a 64-bit adder typically has different delays from the inputs to different output bits).

Please turn in your assignment via the Provide cgi interface. You can use any reasonable format (including writing your answers by hand and taking a picture of the page with your phone).