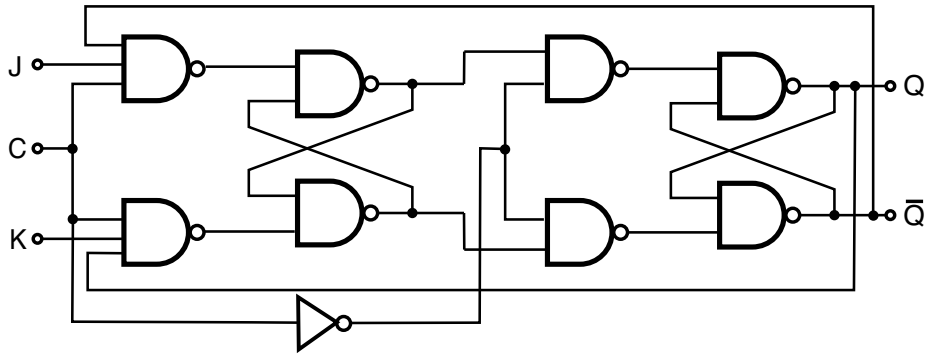
ICS 2022 Problem Sheet #9

Problem 9.1: *JK flip-flops* (1+1+1+1 = 4 points)

JK flip-flops, also colloquially known as jump/kill flip-flops, augment the behaviour of SR flip-flops. The letters J and K were presumably picked by Eldred Nelson in a patent application.

The sequential digital circuit shown below shows the design of a JK flip-flop based on two SR NAND latches. Assume the circuit's output is $Q = 0$ and that the inputs are $J = 0$ and $K = 0$, and that the clock input is $C = 0$. (You can make use of the fact that we already know how an SR NAND latch behaves.)



a) Suppose J transitions to 1 and C transitions to 1 soon after. Create a copy of the drawing and indicate for each line whether it carries a 0 or a 1.

b) Some time later, C transitions back to 0 and soon after J transitions to 0 as well. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.

c) Some time later, J and K both transition to 1 and C transitions to 1 soon after. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.

d) Finally, C transitions back to 0 and soon after J and K both transition to 0 as well. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.

Problem 9.2: *fold function duality theorems* (2+2+2 = 6 points)

The fold functions compute a value over a list (or some other type that is foldable) by applying an operator to the list elements and a neutral element. The foldl function assumes that the operator is left associative, the foldr function assumes that the operatore is right associative. For example, the function application

```
foldl (+) 0 [3,5,2,1]
```

results in the computation of ((((0+3)+5)+2)+1) and the function application

```
foldr (+) 0 [3,5,2,1]
```

results in the computation of (3+(5+(2+(1+0)))). The value computed by the fold functions may be more complex than a simple scalar. It is very well possible to construct a new list as part of the fold. For example:

```haskell
map' :: (a -> b) -> [a] -> [b]
map' f xs = foldr ((:) . f) [] xs
```

The evaluation of `map' succ [1,2,3]` results in the list `[2,3,4]`. There are several duality theorems that can be stated for fold functions. Prove the following three duality theorems:

a) Let `op` be an associative operation with `e` as the neutral element:

`op` is associative: `(x op y) op z = x op (y op z)`
`e` is neutral element: `e op x = x` and `x op e = x`

Then the following holds for finite lists `xs`:

```haskell
foldr op e xs = foldl op e xs
```

b) Let `op1` and `op2` be two operations for which

```haskell
x `op1` (y `op2` z) = (x `op1` y) `op2` z
x `op1` e           = e `op2` x
```

holds. Then the following holds for finite lists `xs`:

```haskell
foldr op1 e xs = foldl op2 e xs
```

c) Let `op` be an associative operation and `xs` a finite list. Then

```haskell
foldr op a xs = foldl op' a (reverse xs)
```

holds with

```haskell
x op' y = y op x
```