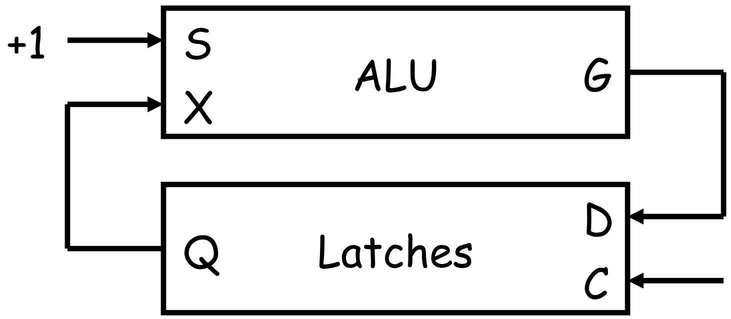
## Flip-Flops

- Last time, we saw how latches can be used as memory in a circuit
- Latches introduce new problems:
  - We need to know when to enable a latch
  - We also need to quickly disable a latch
  - In other words, it's difficult to control the timing of latches in a large circuit
- We solve these problems with two new elements: clocks and flip-flops
  - Clocks tell us when to write to our memory
  - Flip-flops allow us to quickly write the memory at clearly defined times
  - Used together, we can create circuits without worrying about the memory timing

## Using latches in real life

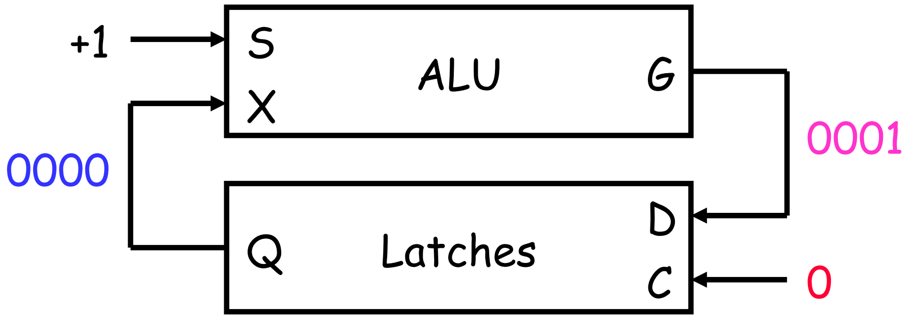
- We can connect some latches, acting as memory, to an ALU



- Let's say these latches contain some value that we want to increment
  - The ALU should read the current latch value
  - It applies the "G = X + 1" operation
  - The incremented value is stored back into the latches
- At this point, we have to stop the cycle, so the latch value doesn't get incremented again by accident
- One convenient way to break the loop is to disable the latches

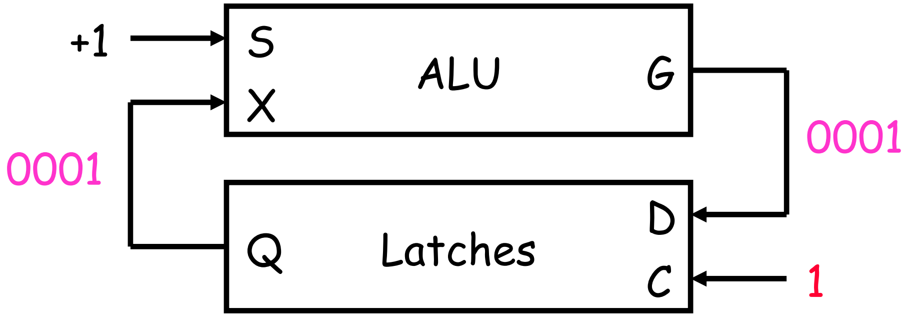
# Making latches work right

- Our example used latches as memory for an ALU
  - Let's say there are four latches initially storing 0000
  - We want to use an ALU to increment that value to 0001
- Normally the latches should be disabled, to prevent unwanted data from being accidentally stored
  - In our example, the ALU can read the current latch contents, 0000, and compute their increment, 0001
  - But the new value cannot be stored back while the latch is disabled

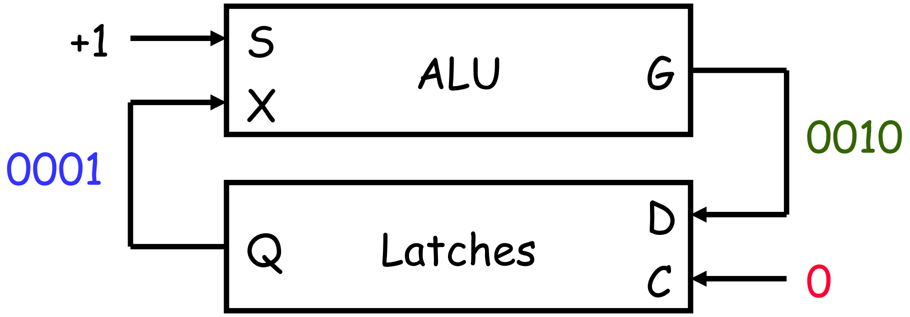


## Writing to the latches

- After the ALU has finished its increment operation, the latch can be enabled, and the updated value is stored.



- The latch must be quickly disabled again, *before* the ALU has a chance to read the new value 0001 and produce a new result 0010.



## Two main issues

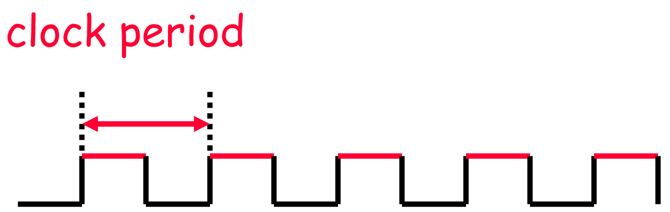
- So to use latches correctly within a circuit, we have to:
  - Keep the latches disabled until new values are ready to be stored
  - Enable the latches just long enough for the update to occur
- There are two main issues we need to address:
  - How do we know exactly when the new values are ready?

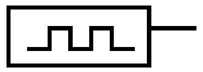
    We'll add another signal to our circuit. When this new signal becomes 1, the latches will know that the ALU computation has completed and data is ready to be stored
  - How can we enable and then quickly disable the latches?

    This can be done by combining latches together in a special way, to form what are called flip-flops

## Clocks and synchronization

- A clock is a special device whose output continuously alternates between 0 and 1.





- The time it takes the clock to change from 1 to 0 and back to 1 is called the clock period, or clock cycle time
- The clock frequency is the inverse of the clock period. The unit of measurement for frequency is the hertz
- Clocks are often used to synchronize circuits
  - They generate a repeating, predictable pattern of 0s and 1s that can trigger certain events in a circuit, such as writing to a latch
  - If several circuits share a common clock signal, they can coordinate their actions with respect to one another
- This is similar to how humans use real clocks for synchronization.

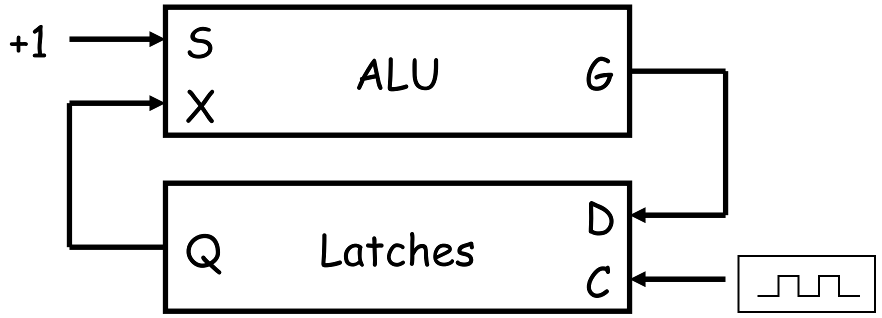
## More about clocks

- Clocks are used extensively in computer architecture
- All processors run with an internal clock
  - Modern chips run at frequencies up to 3.2 GHz.
  - This works out to a cycle time as little as 0.31 ns!

- Memory modules are often rated by their clock speeds too—examples include "PC133" and "DDR400" memory
- Be careful…higher frequencies do not always mean faster machines!
  - You also have to consider how much work is actually being done during each clock cycle
  - How much stuff can really get done in just 0.31 ns?

# Synchronizing our example

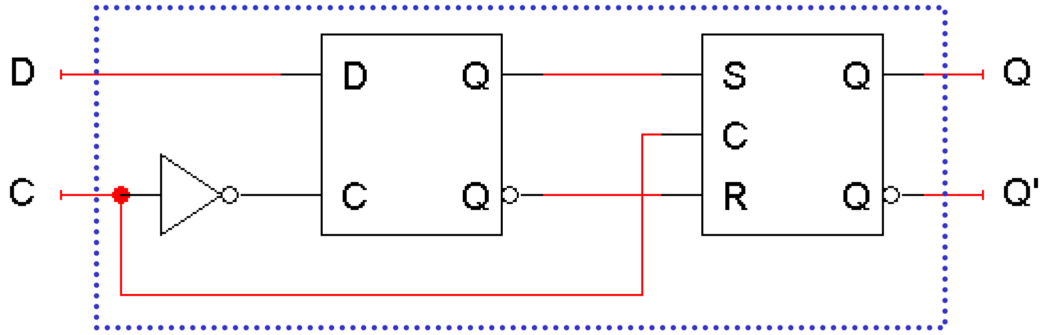
- We can use a clock to synchronize our latches with the ALU
  - The clock signal is connected to the latch control input *C*
  - The clock controls the latches. When it becomes 1, the latches will be enabled for writing



- The clock period must be set appropriately for the ALU
  - It should not be too short. Otherwise, the latches will start writing before the ALU operation has finished
  - The faster the ALU runs, the shorter the clock period can be
  - But in the current design, if the clock period is too large, it's a problem too..

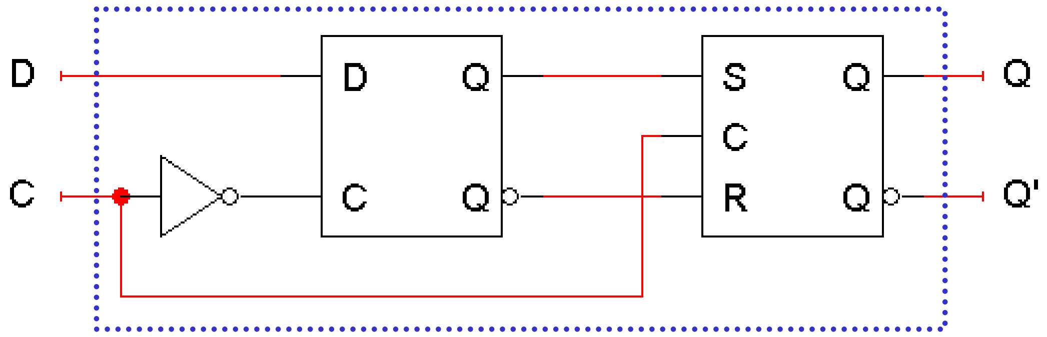
## Flip-flops

- The second issue was how to enable a latch for just an instant
- Here is the internal structure of a D flip-flop
  - The flip-flop inputs are C and D, and the outputs are Q and Q'
  - The D latch on the left is the master, while the SR latch on the right is called the slave



- Note the layout here
  - The flip-flop input D is connected directly to the master latch
  - The master latch output goes to the slave
  - The flip-flop outputs come directly from the slave latch

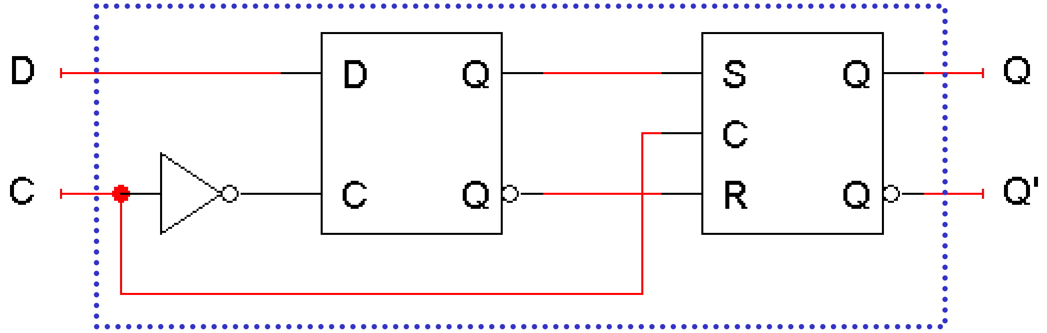
## D flip-flops when C=0



- The D flip-flop's control input C enables *either* the D latch or the SR latch, but not both

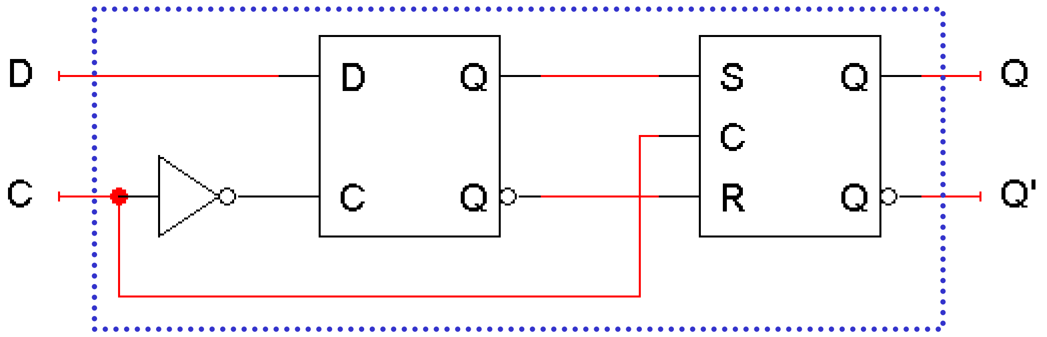
- When C = 0:
  - The master latch is enabled, and it monitors the flip-flop input D. Whenever D changes, the master's output changes too
  - The slave is disabled, so the D latch output has no effect on it. Thus, the slave just maintains the flip-flop's current state

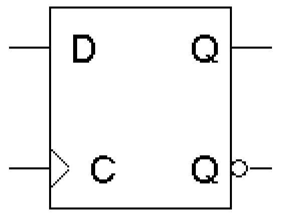
## D flip-flops when C=1



- *As soon as C becomes 1, (i.e. on the rising edge of the clock)*
  - The master is disabled. Its output will be the *last* D input value seen just before C became 1
  - Any subsequent changes to the D input while C = 1 have no effect on the master latch, which is now disabled
  - The slave latch is enabled. Its state changes to reflect the master's output, which again is the D input value from right when C became 1

## Positive edge triggering



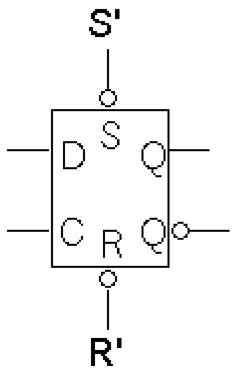


- This is called a positive edge-triggered flip-flop
  - The flip-flop output Q changes *only* after the positive edge of C
  - The change is based on the flip-flop input values that were present right at the positive edge of the clock signal
- The D flip-flop's behavior is similar to that of a D latch except for the positive edge-triggered nature, which is not explicit in this table

| C | D | Q |
|---|---|---|
| 0 | x | No change |
| 1 | 0 | 0 (reset) |
| 1 | 1 | 1 (set) |

## Direct inputs

- One last thing to worry about… what is the starting value of Q?
- We could set the initial value synchronously, at the next positive clock edge, but this actually makes circuit design more difficult
- Instead, most flip-flops provide direct, or asynchronous, inputs that let you immediately set or clear the state
  - You would "reset" the circuit once, to initialize the flip-flops
  - The circuit would then begin its regular, synchronous operation



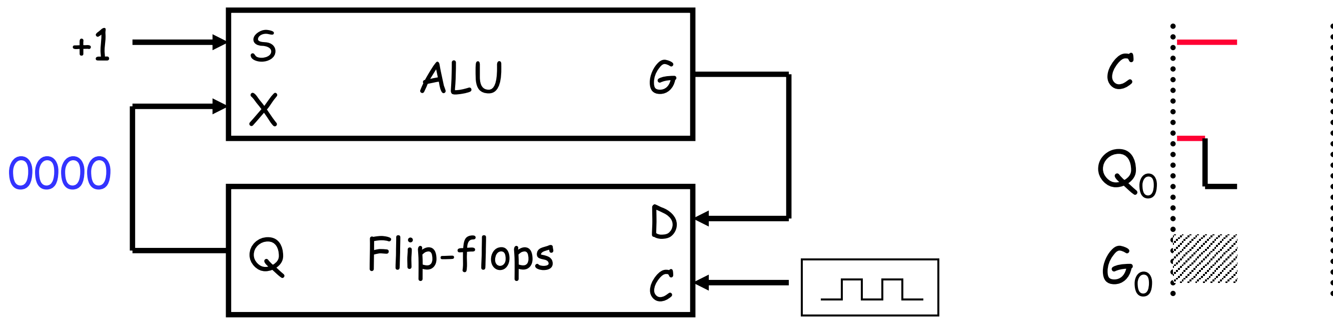
| S' | R' | C | D | Q |
|---|---|---|---|---|
| 0 | 0 | x | x | Avoid! |
| 0 | 1 | x | x | 1 (set) |
| 1 | 0 | x | x | 0 (reset) |
| 1 | 1 | 0 | x | No change |
| 1 | 1 | 1 | 0 | 0 (reset) |
| 1 | 1 | 1 | 1 | 1 (set) |

} Direct inputs to set or reset the flip-flop

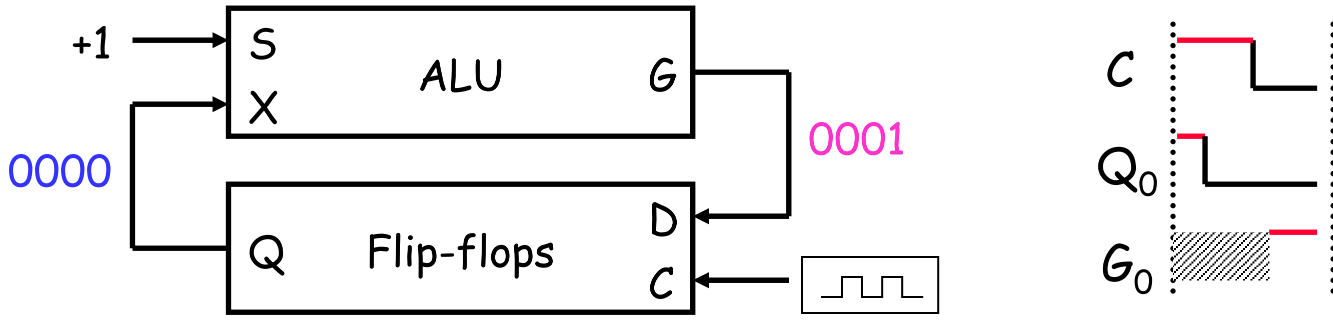
} S'R' = 11 for "normal" operation of the D flip-flop

# Our example with flip-flops

- We can use the flip-flops' direct inputs to initialize them to 0000

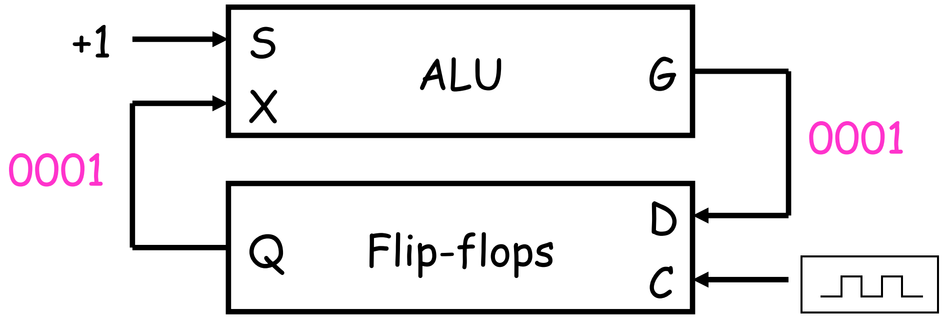


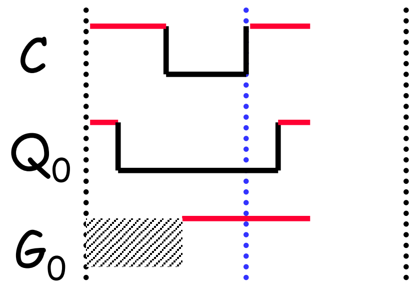
- During the clock cycle, the ALU outputs 0001, but this does not affect the flip-flops yet



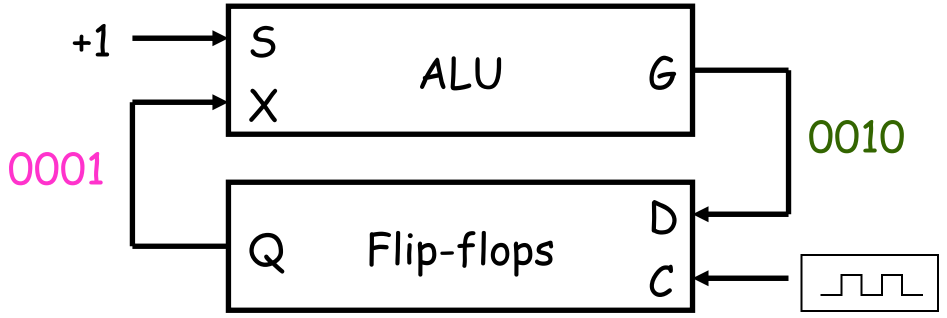
## Example continued

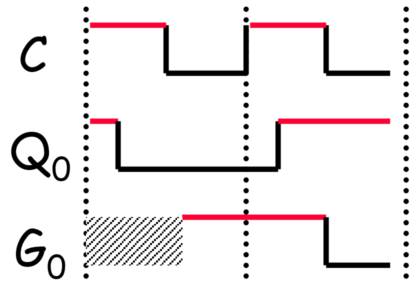
- The ALU output is copied into the flip-flops at the next positive edge of the clock signal.





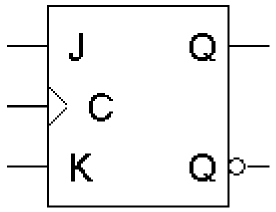
- The flip-flops automatically "shut off," and no new data can be written until the next positive clock edge... even though the ALU produces a new output.





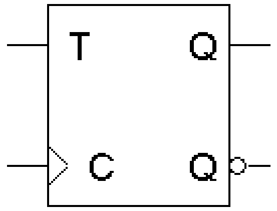
## Flip-flop variations

- We can make different versions of flip-flops based on the D flip-flop, just like we made different latches based on the S'R' latch
- A JK flip-flop has inputs that act like S and R, but the inputs JK=11 are used to *complement* the flip-flop's current state



| C | J | K | $Q_{next}$ |
|---|---|---|---|
| 0 | x | x | No change |
| 1 | 0 | 0 | No change |
| 1 | 0 | 1 | 0 (reset) |
| 1 | 1 | 0 | 1 (set) |
| 1 | 1 | 1 | $Q'_{current}$ |

- A T flip-flop can only maintain or complement its current state.



| C | T | $Q_{next}$ |
|---|---|---|
| 0 | x | No change |
| 1 | 0 | No change |
| 1 | 1 | $Q'_{current}$ |

## Characteristic tables

- The tables that we've made so far are called characteristic tables
  - They show the next state Q(t+1) in terms of the current state Q(t) and the inputs
  - For simplicity, the control input *C* is not usually listed.
  - Again, these tables don't indicate the positive edge-triggered behavior of the flip-flops that we'll be using.

| D | Q(t+1) | Operation |
|---|---|---|
| 0 | 0 | Reset |
| 1 | 1 | Set |

| J | K | Q(t+1) | Operation |
|---|---|---|---|
| 0 | 0 | Q(t) | No change |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | Q'(t) | Complement |

| T | Q(t+1) | Operation |
|---|---|---|
| 0 | Q(t) | No change |
| 1 | Q'(t) | Complement |

## Characteristic equations

- We can also write characteristic equations, where the next state Q(t+1) is defined in terms of the current state Q(t) and inputs

| D | Q(t+1) | Operation |
|---|---|---|
| 0 | 0 | Reset |
| 1 | 1 | Set |

$$Q(t+1) = D$$

| J | K | Q(t+1) | Operation |
|---|---|---|---|
| 0 | 0 | Q(t) | No change |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | Q'(t) | Complement |

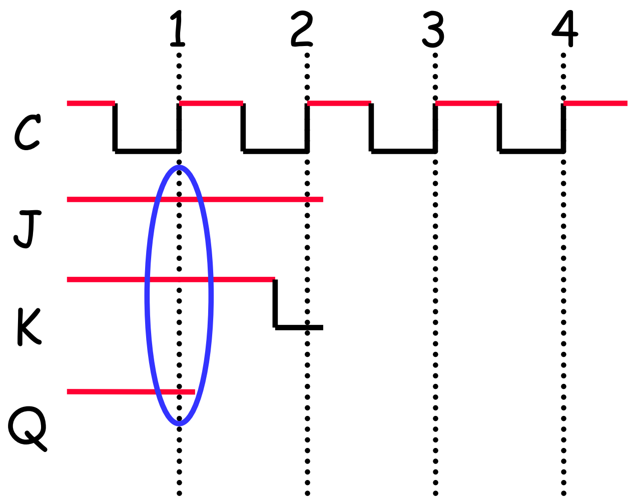
$$Q(t+1) = K'Q(t) + JQ'(t)$$

| T | Q(t+1) | Operation |
|---|---|---|
| 0 | Q(t) | No change |
| 1 | Q'(t) | Complement |

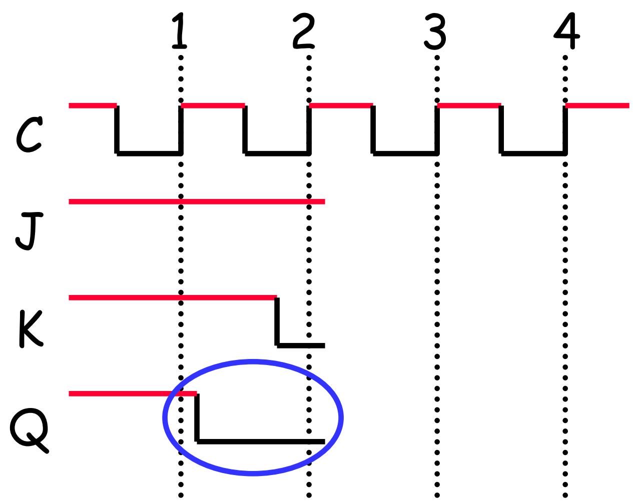
$$\begin{aligned} Q(t+1) &= T'Q(t) + TQ'(t) \\ &= T \oplus Q(t) \end{aligned}$$

# Flip flop timing diagrams

- "Present state" and "next state" are relative terms
- In the example JK flip-flop timing diagram on the left, you can see that at the first positive clock edge, J=1, K=1 and Q(1) = 1
- We can use this information to find the "next" state, Q(2) = Q(1)'
- Q(2) appears right after the first positive clock edge, as shown on the right. It will not change again until after the second clock edge



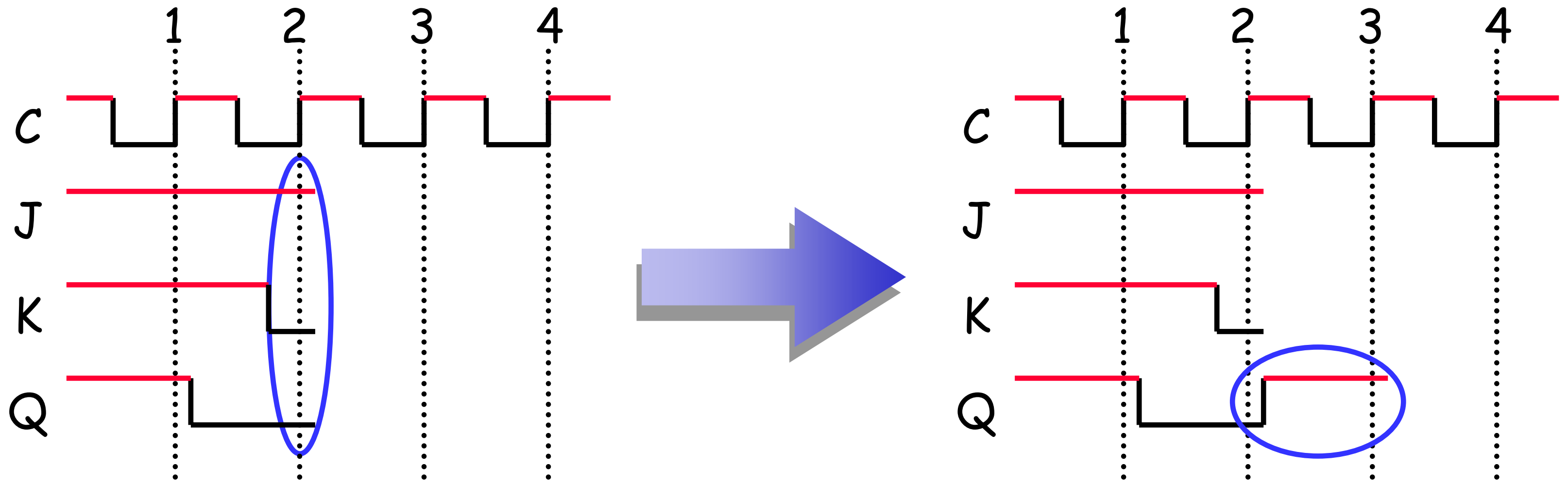
These values at clock cycle 1...



... determine the "next" Q

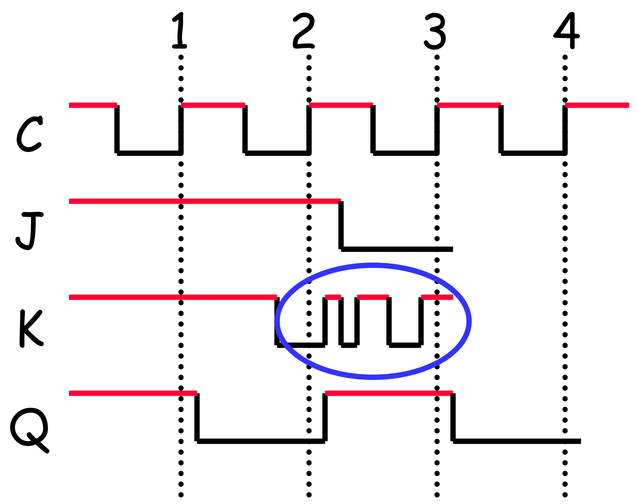
# “Present” and “next” are relative

- Similarly, the values of J, K and Q at the second positive clock edge can be used to find the value of Q during the third clock cycle
- When we do this, Q(2) is now referred to as the “present” state, and Q(3) is now the “next” state



## Positive edge triggered

- One final point to repeat: the flip-flop outputs are affected only by the input values *at the positive edge*
  - In the diagram below, K changes rapidly between the second and third positive edges
  - But it's only the input values at the third clock edge (K=1, and J=0 and Q=1) that affect the next state, so here Q changes to 0
- This is a fairly simple timing model. In real life there are "setup times" and "hold times" to worry about as well, to account for internal and external delays



## Summary

- To use memory in a larger circuit, we need to:
  - Keep the latches disabled until new values are ready to be stored
  - Enable the latches just long enough for the update to occur
- A clock signal is used to synchronize circuits. The cycle time reflects how long combinational operations take
- Flip-flops further restrict the memory writing interval, to just the positive edge of the clock signal
  - This ensures that memory is updated only once per clock cycle
  - There are several different kinds of flip-flops, but they all serve the same basic purpose of storing bits
- Next week we'll talk about how to analyze and design sequential circuits that use flip-flops as memory