Registers and Counters

Register

- Register is built with gates, but has memory.
- The only type of flip-flop required in this class – the D flip-flop
  - Has at least two inputs (both 1-bit): D and clk
  - Has at least one output (1-bit): Q
  - At the rising edge of clk (when clk is changing from 0 to 1), Q <= D.
  - Q does not change value at ANY OTHER TIME except at the rising edge of clk

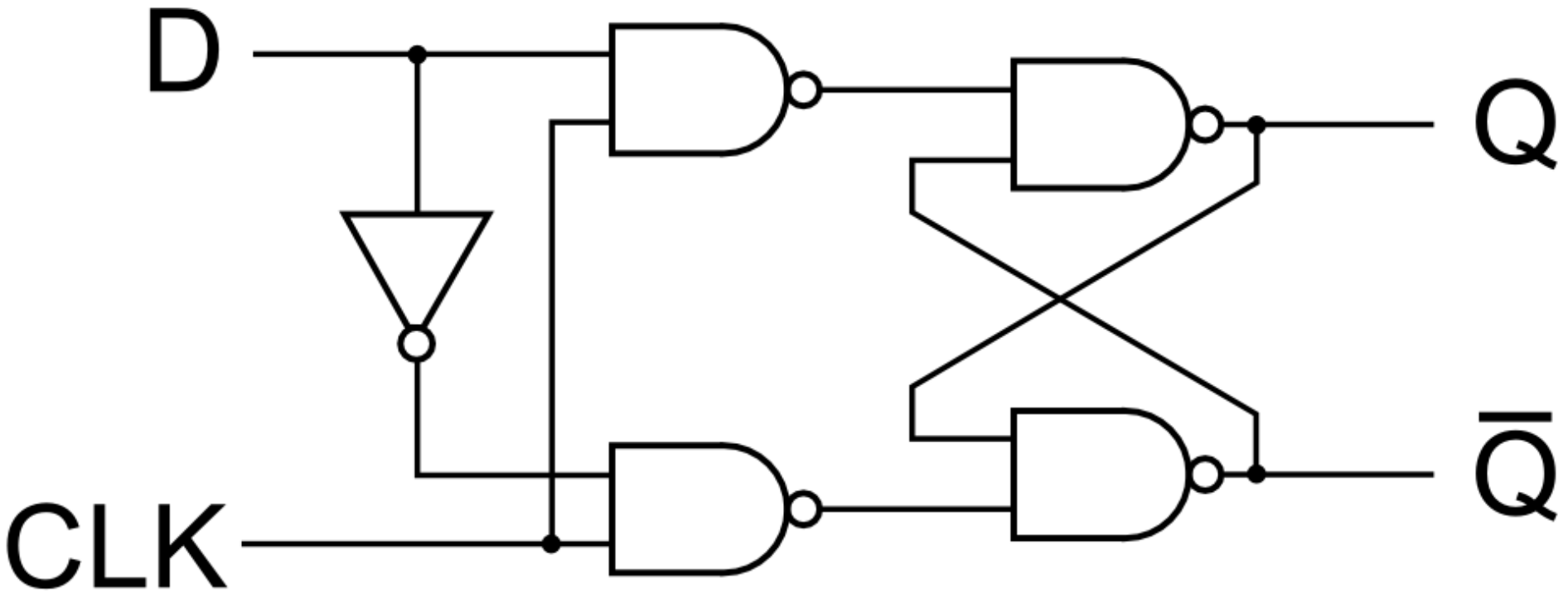
D flip-flop

```
module Dff (D, clk, Q);
  input D, clk;
  output reg Q;

  always @(posedge clk) begin
     Q = D;
  end

endmodule
```

D flip-flop

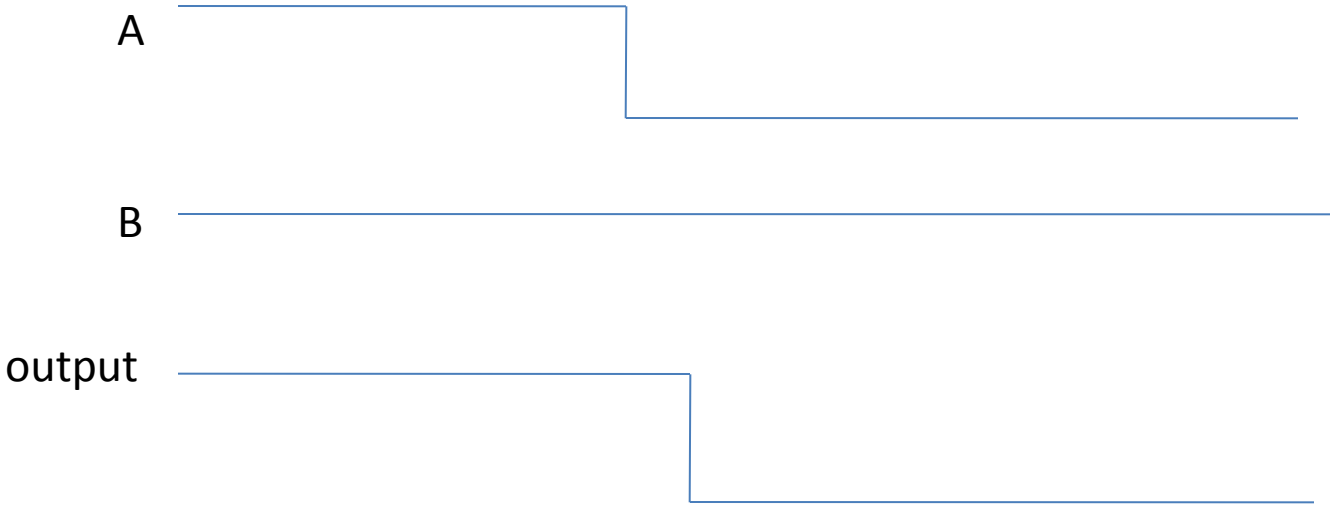


D flip-flop can hold value

- Note that the output of the D flip-flop (Q) does not follow the change of the input (D). It holds the value until the next time it is allowed to change – the rising edge of the clock.
- This is why you can write to a register and expect that the next time you want to use it the value is still there. Your MIPS code won't work if the values in the registers can change at random time.
- Now you know another piece of MIPS processor – MIPS has 32 registers, each register is 32 bits, so basically you have 1024 D-flip-flops.

Delay

- Real circuits have delays caused by charging and discharging.
- So, once the input to a gate changes, the output will change after a delay, usually in the order of nano seconds. An and gate:



Delay

- A more realistic D flip-flop:

```
module Dff1 (D, clk, Q, Qbar);
        input D, clk;
        output reg Q, Qbar;

        initial begin
                Q = 0;
                Qbar = 1;
        end

        always @(posedge clk) begin
                #1
                Q = D;
                #1
                Qbar = ~Q;
        end
endmodule
```

What happens if…

What happens if I connect a Dff like this?

```
wire Q2, Qbar2;
Dff1 D2 (Qbar2, clk, Q2, Qbar2);
```

What happens if …

- We connect three D-flip-flops in a chain?

Implementing a 3-bit counter

- A 3-bit counter changes value at every rising edge of the clock, and counts from 0 to 7 and then back to 0.
- We are going to implement it with D flip-flops and some combinatorial logic.

Any suggestions?

- How many D flip-flips do we need?
- How to control the D flip-flops to realize this function?

The last bit

- The output bit can be dealt with one by one
- Let’s work with something simple first.
- How to implement the last bit?

How about the other two bits?

- The only thing we can control of the D flip-flop is the D signal, because the clk should always be connected to the ``true clk.''
  - In hardware, special cares are given to the clk signal to make sure that they don't have glitches and other stuff

States

- The counter has 8 ``states,'' from 0 to 7. It moves from the current state to the next state at the clk.
- State is represented by the current value of Q.
- What the next state is is totally determined by the current state.

D

- To go from the current state to the next state, we tell the D flip-flop what the next Q should be by setting D to appropriate values, that is, D=next Q.

Q2	Q1	Q0	D2	D1	D0
0	0	0			1
0	0	1			0
0	1	0			1
0	1	1			0
1	0	0			1
1	0	1			0
1	1	0			1
1	1	1			0

D

Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

D

- The key part is: D is determined by the current state. That is, D2, D1, D0 are all functions of Q2, Q1, and Q0.
- We know that
  - D0 = ~Q0.
- How to get the functions for D2 and D1?

D1

- Based on the truth table.

Q0 \ Q2Q1	00	01	11	10
0	**0**	**1**	**1**	**0**
1	1	0	0	1

- So, D1 = (~Q1&Q0)|(Q1&~Q0) = Q1^Q0.

D2

- Based on the truth table.

Q0 \ Q2Q1	00	01	11	10
0	**0**	**0**	**1**	**1**
1	0	1	0	1

- So, D2 = (Q2&~Q1)|(Q2&~Q0)|(~Q2&Q1&Q0)

Load

- Sometimes, we wish to load a certain value to the counter.
- The counter will have a ``load'' input and a ``L'' input. When load is 1, at the next clock, Q=L.
- How to make this happen?

Load

- Sometimes, we wish to load a certain value to the counter.
- The counter will have a ``load'' input and a ``L'' input. When load is 1, at the next clock, Q=L.
- How to make this happen?
- Use a 2-1 selector in front of each D input. Use load as the select signal. One of the input is the D signal from the counter, the other is L.

Program Counter (PC)

- So we have basically implemented the program counter for mips.
- Remember that the PC will
  - Increment by 4 if there no jump or branch
  - Or be loaded with the address of the instruction to be jumped to if there is a jump or a branch

The next state table

- How to implement a “counter”, which will count as 0,2,3,1,4,5,7,6,0,2,3,……

Q2	Q1	Q0	D2	D1	D0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

The next state table

- How to implement a counter, which will count as 0,2,3,1,4,5,7,6,0,2,3,……

Q2	Q1	Q0	D2	D1	D0
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	1	1	0

D0

- D0 =

Q0 \ Q2Q1	00	01	11	10
0	**0**	**1**	**0**	**1**
1	0	1	0	1

= (~Q2&Q1) | (Q2&~Q1) = Q2^Q1

D1

- D1 =

Q0 \ Q2Q1	00	01	11	10
0	1	1	0	0
1	0	0	1	1

= (~Q0 & ~Q2) | (Q0 & Q2) = ~(Q0^Q2)

D2

- D2 =

Q0 \ Q2Q1	00	01	11	10
0	**0**	**0**	**0**	**1**
1	1	0	1	1

= (Q2 & ~Q1) | (Q2 & Q0) | (Q0 & Q1)