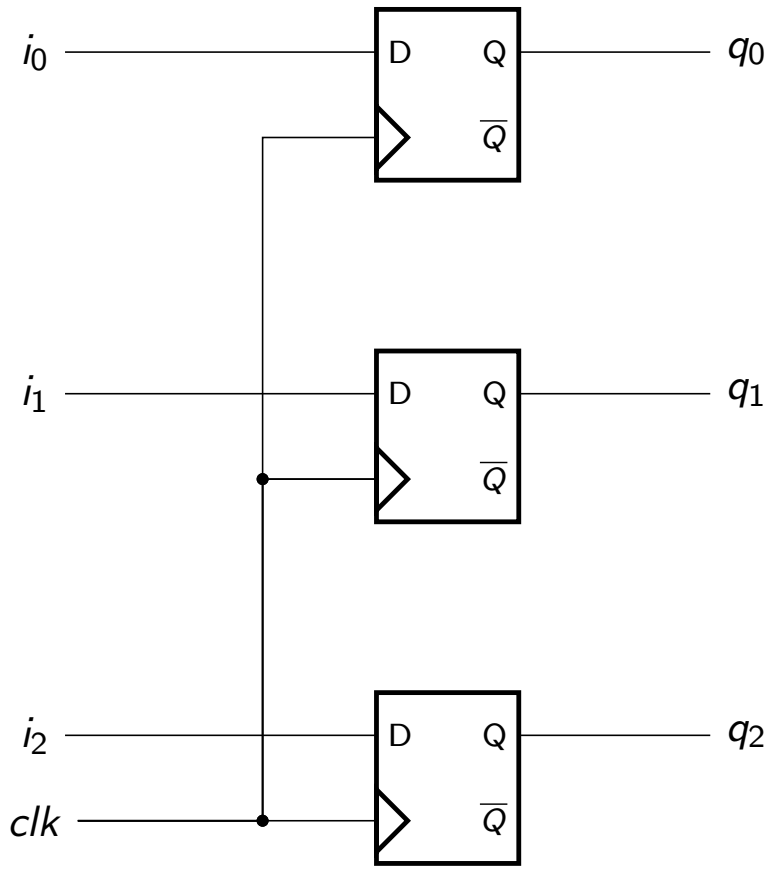
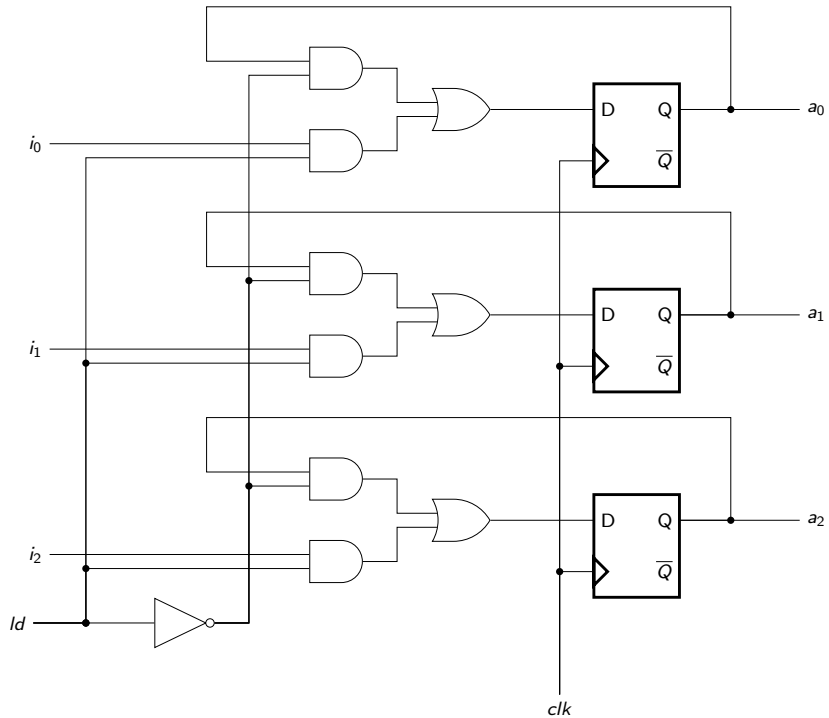
Registers

- A register is nothing more than n flip flops together in a group in order to perform some task.
- Flip flops in a register all use the same clock.
- The simple example... a 3-bit register which will load new data on every active clock edge.



Register with parallel load and hold

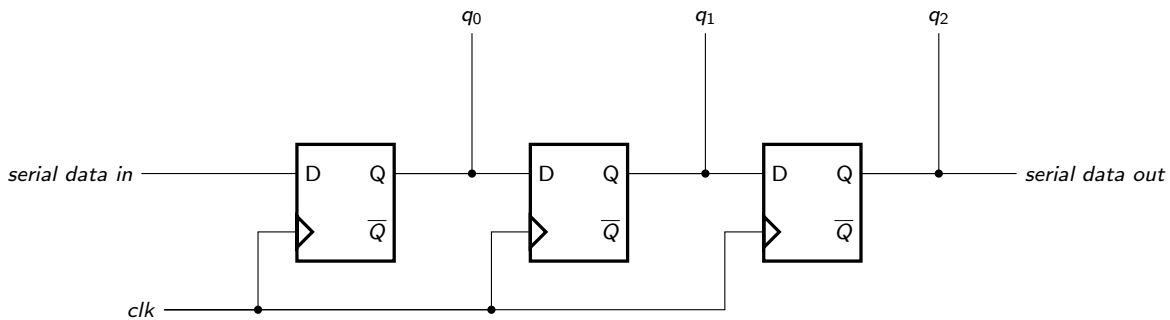
- Design a 3-bit register which has the ability to both load new data as well as hold current data.
- The idea is to "get the correct value to the input of each flip flop depending on the operation to be performed".



- When $ld = 1$, the register loads new data. When $ld = 0$, the register holds its current value.
- The logic placed in front of the *DFF*s is nothing more than a multiplexer with *ld* as a select line and the appropriate value connected to each data input of the multiplexer.

Shift registers

- The purpose of a shift register is to accept input and to shift it one bit over on every active clock edge.
- Such a device can, for instance, be used to take "serial data" and convert it to "parallel data" or visa versa (assuming we have a shift register with parallel load).
- Simple shift register...



- As the active clock edge arrives, data present at the *serial data in* gets transferred towards the *serial data out*; so the data gets shifted to the right each time an active clock edge arrives.

Universal shift registers

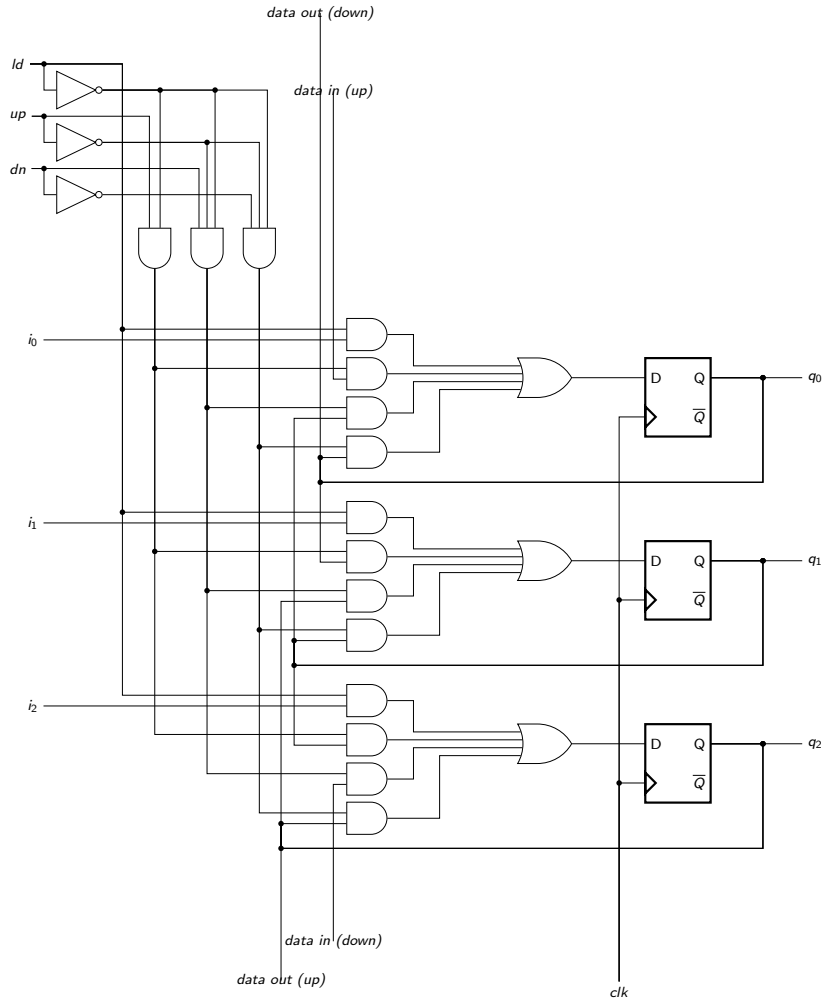
- Make an n-bit register that can shift up, shift down, do a parallel load or hold data.
- Table describing how things should work...

Inputs			**Action**
load	*up*	*dn*	
0	0	0	*hold*
0	0	1	*shift down*
0	1	X	*shift up*
1	X	X	*load*

- By *shift up*, we mean that data moves from q_0 towards q_{n-1}. By *shift down*, we mean that data moves from q_{n-1} towards q_0.

Universal shift registers

- The circuit for 3 bits...



- The extra gates and control logic are effectively performing a multiplexer operation to get the correct signal to the input of the flip flop.

Universal shift registers

- The extra logic... Consider the j-th DFF input...

$$
\begin{aligned}
d_j &= \underbrace{(ld)i_j}_{\text{load}} + \underbrace{(\overline{ld})upq_{i-1}}_{\text{shift up}} + \underbrace{(\overline{ld})(\overline{up})(dn)q_{i+1}}_{\text{shift down}} + \underbrace{(\overline{ld})(\overline{up})(\overline{dn})q_i}_{\text{hold}} \\
&= (ld)(i_j) + +(\overline{ld})((up)q_{i-1} + +(\overline{up})(dn)q_{i+1} + (\overline{up})(\overline{dn})q_i) \\
&= (ld)(i_j) + +(\overline{ld})((up)(q_{i-1}) + (\overline{up})((dn)q_{i+1} + (\overline{dn})q_i)) \\
&= (ld)(i_j) + +(\overline{ld})((up)(q_{i-1}) + (\overline{up})(\underbrace{(dn)q_{i+1} + (\overline{dn})q_i}_{\text{MUX}})) \\
&\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{MUX}} \\
&\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{MUX}}
\end{aligned}
$$