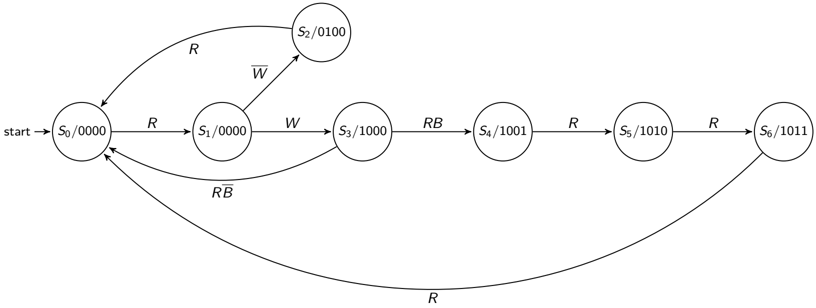
State assignment

- Recall... state assignment is when we assign binary values to represent symbolic states in a state diagram or state table. We need to do this prior to designing a circuit implementation.
- Since circuit outputs and next state functions depend on the current state, how we do state assignment can impact the complexity of our circuit.
- Further, since we are assigning binary patterns, there is nothing restricting us from using more flip flops than required.
- We will consider a few different ways to perform state assignment; each approach will have some *pros and cons*.

State assignment

- Consider the following state diagram for a problem that has 3 inputs R,B and W. The problem has 4 outputs A, B, C and D. Finally, the problem requires 7 states denoted as S_0 through S_6.
- Conditions on edges are shown as logic equations; if none of the conditions to leave a state are true, it's assumed that one remains in the current state. This is done to reduce the "clutter" in the state diagram.



- We will consider several ways to assign binary patterns to the states S_0 through S_6.

State assignment — minimum flip flop count

- First way to perform state assignment is to try and use the minimum number of flip flops. For n states, we require *at least* $\lceil \log n \rceil$ bits (or flip flops).
- In our example, we have 7 states and therefore require $\lceil \log 7 \rceil = 3$ flip flops; we don't have enough binary patterns available if we use less than 3 flip flops.
- With 3 flip flops, we have 8 different patterns available, although we only need 7 of them. The obvious assignment is to do the assignment sequentially: $S_0 = 000$, $S_1 = 001$, $S_2 = 010$, $S_3 = 011$, $S_4 = 100$, $S_5 = 101$, and $S_6 = 110$.
- We could have assigned binary patterns differently... Unfortunately, it is difficult to say which method will lead to the simplest circuit. We can neither predict the complexity of the next state functions nor the circuit output equations.
- The only benefit this approach offers is that we are using the minimum number of flip flops.

State assignment — output encoding

- Sometimes we can try to encode the states such that the outputs are equal to the current state.
- *Aside: The best example of output encoding is a counter in which the current count value is equal to the state.*
- For our example, we can make a table to see if output encoding is possible:

Current state	**Output**			
	A	B	C	D
S_0	0	0	0	0
S_1	0	0	0	0
S_2	0	1	0	0
S_3	1	0	0	0
S_4	1	0	0	1
S_5	1	0	1	0
S_6	1	0	1	1

- Since we have 4 outputs, we can consider whether or not we should implement this circuit using 4 flip flops and use the output values as the state assignment.
  - As tempting as this is, it will not work since it would require both S_0 and S_1 to be assigned the binary pattern 0000 and two states cannot have the same state assignment!

State assignment — output encoding

- We don't have to give up. What we need to achieve is the ability to get our circuit outputs by looking at the state *and ensuring that every state has a different binary pattern*.
- In this example, we can accomplish this task by adding an extra bit...

Current state	**Output**				**Extra bits**
	A	B	C	D	E
S_0	0	0	0	0	0
S_1	0	0	0	0	1
S_2	0	1	0	0	X
S_3	1	0	0	0	X
S_4	1	0	0	1	X
S_5	1	0	1	0	X
S_6	1	0	1	1	X

- A suitable encoding would be $S_0 = 00000$, $S_1 = 00001$, $S_2 = 01000$, $S_3 = 10000$, $S_4 = 10010$, $S_5 = 10010$, and $S_6 = 10011$. This encoding is suitable because every state is assigned a different pattern.
- This solution requires 5 flip flops which is 2 more than the minimum.
- We can't predict the complexity of the next state functions.
- However, *we can predict the complexity of the output logic* — there isn't any. The outputs for this circuit are taken directly from the first 4 flip flops. In other words, the output logic is simply wire.

State assignment — one hot encoding

- One hot encoding is specifically intended for use with *DFF*.
- The output of **only one** output can be 1 or "hot" at any given time.
  - This clearly implies that we will need one flip flop for each state.
  - Further, we can always tell which state we are in simply by looking to see if the *DFF* representing that state has an output of 1.
- In our example, we would require 7 flip flops. We would use the following state assignment: $S_0 = 0000001$, $S_1 = 0000010$, $S_2 = 0000100$, $S_3 = 0001000$, $S_4 = 0010000$, $S_5 = 0100000$, and $S_6 = 1000000$.
- One hot encoding requires a lot of flip flops which is a potential disadvantage.
- Conversely, when you use one hot encoding, you tend to get simple flip flop input equations. Further, the equations for the circuit outputs also tend to be very simple.

State assignment — one hot encoding

- Consider the following state table for a circuit with one input a, one output z and three states:

Current State	**Next State**		**Output (z)**	
	$a = 0$	$a = 1$	$a = 0$	$a = 1$
s_0	s_2	s_0	1	1
s_1	s_0	s_1	0	0
s_2	s_1	s_2	1	0

- We can use one hot encoding and do the assignment $s_0 = 001$, $s_1 = 010$ and $s_2 = 100$ (which requires 3 DFFs.
- New state table showing assignment:

Current State	**Next State**		**Output**	
	$a = 0$	$a = 1$	$a = 0$	$a = 1$
$q_2 q_1 q_0$	$q_2 q_1 q_0$	$q_2 q_1 q_0$	z	z
001	100	001	1	1
010	001	010	0	0
100	010	100	1	0

State assignment — one hot encoding

- Using DFF, we need to get the DFF inputs:

Current State	Next State		DFF Inputs	
	$a = 0$	$a = 1$	$a = 0$	$a = 1$
$q_2 q_1 q_0$	$q_2 q_1 q_0$	$q_2 q_1 q_0$	$d_2 d_1 d_0$	$d_2 d_1 d_0$
001	100	001	100	001
010	001	010	001	010
100	010	100	010	100

- You should be able to write down the *DFF* input equations by simply looking at the table. For the *i*-th flip flop input, you should ask yourself "when will we enter into state *i*".
- Flip flop input equations...

$$\begin{aligned} d_0 &= \bar{a}q_1 + aq_0 \\ d_1 &= \bar{a}q_2 + aq_1 \\ d_2 &= \bar{a}q_0 + aq_2 \end{aligned}$$

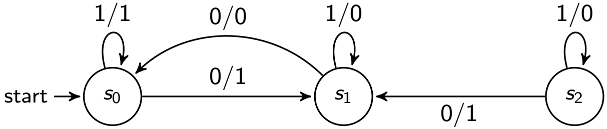
- The output equations are similarly simply to write down...

$$z = q_0 + \bar{a}z_2$$

- So not only are the various equations simple in form, they are also simple to write down directly from the state table.

State assignment — one hot encoding

- You can also write the equations down directly if given a state diagram...



- Just like with a state table, you ask for each state *i*, under what conditions do you enter state *i*. This is obtained by looking at the edges **entering** a state.
- Flip flop input equations...

$$\begin{aligned} d_0 &= \bar{a}q_1 + aq_0 \\ d_1 &= \bar{a}q_2 + aq_1 \\ d_2 &= \bar{a}q_0 + aq_2 \end{aligned}$$

- Output equations...

$$z = q_0 + \bar{a}z_2$$

- **Note:** When working with a state diagram, one does need to be careful that no information is “missing”; e.g., sometimes in a state diagram, one might leave out certain edges such as the self loops. To use one hot encoding directly from the state diagram, you need to be sure that **all edges entering each state** are shown otherwise your equations will be missing necessary logic.