Outline

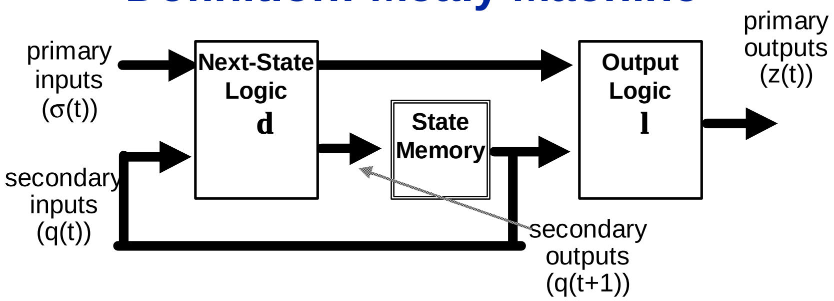
- ❍ Last time:
  - ➜ Introduction to number systems: sign/magnitude, ones complement, twos complement
  - ➜ Review of latches, flip flops, counters
- ❍ This lecture:
  - ➜ Review State Tables & State Transition Diagrams
  - ➜ Implementation Using D Flip-Flops
  - ➜ Machine Equivalence
  - ➜ Incompletely Specified Machines
  - ➜ State Assignment & State Coding Schemes
  - ➜ Design Example: Assign Codes to States
  - ➜ Design Example: Implement Using D flip-flops
  - ➜ Design Example: Implement Using T flip-flops

Clocked Synchronous Finite-State Machines

- ❍ Example:

Consider the student association coffee vending machine which sells coffee at 15¢/cup. The machine will accept nickles, dimes, and quarters, one at a time. The coffee release line will be set to true when 15¢ or more has been put into the machine and the machine will return the correct change.

Definition: Mealy Machine



❍ A sequential machine or *Mealy Machine* can be characterized by the quintuple: M = (S, Q, Z, d l) where

S = finite non-empty set of input symbols $\sigma_1, \sigma_2, ..., \sigma_i$

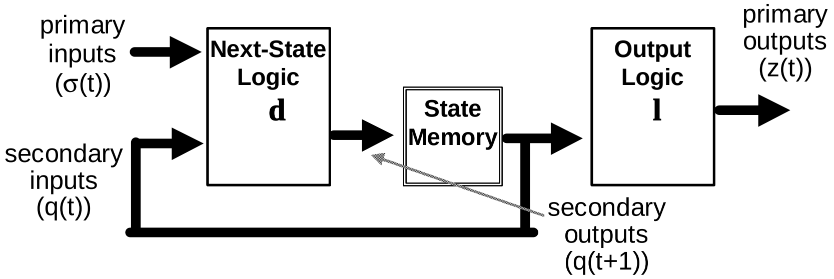
Q = finite non-empty set of states $q_1, q_2, ..., q_n$

Z = finite non-empty set of output symbols $z_1, z_2, ..., z_m$

d= *next-state* function, which maps Q ´ S ® Q

l = the *output* function, which maps Q ´ S ® Z

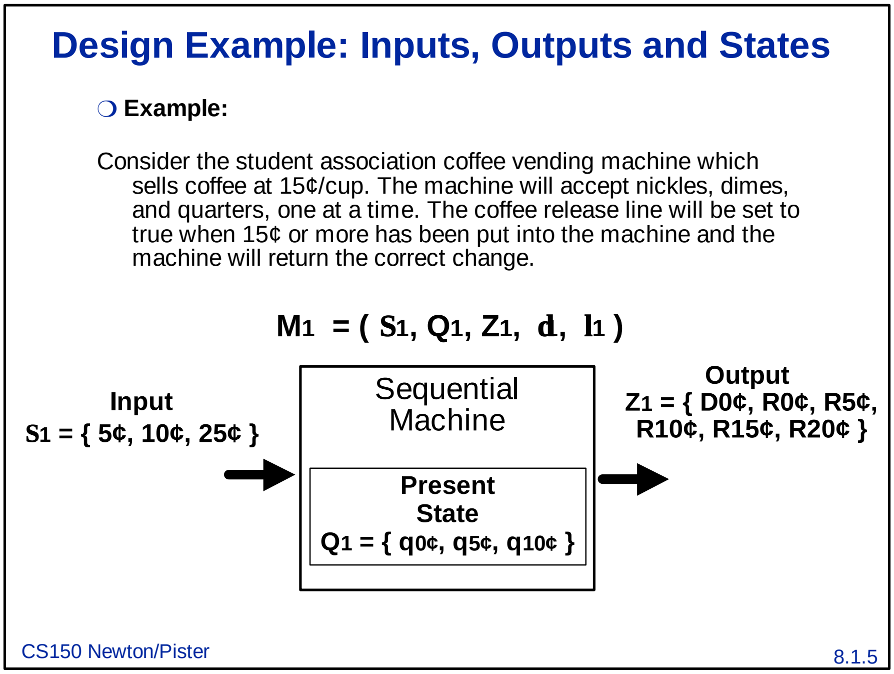
Definition: Moore Machine



❍ A sequential machine is said to be of the Moore type (*Moore Machine*) if its output function is a function only of its states (i.e. l : Q ® Z)

❍ Every Mealy Machine can be converted to a Moore Machine and vice versa.

❍ If the State Memory is clocked, the machines are Clocked, Synchronous Mealy and Moore machines respectively.



Design Example: Inputs, Outputs and States

❍ **Example:**

Consider the student association coffee vending machine which sells coffee at 15¢/cup. The machine will accept nickles, dimes, and quarters, one at a time. The coffee release line will be set to true when 15¢ or more has been put into the machine and the machine will return the correct change.

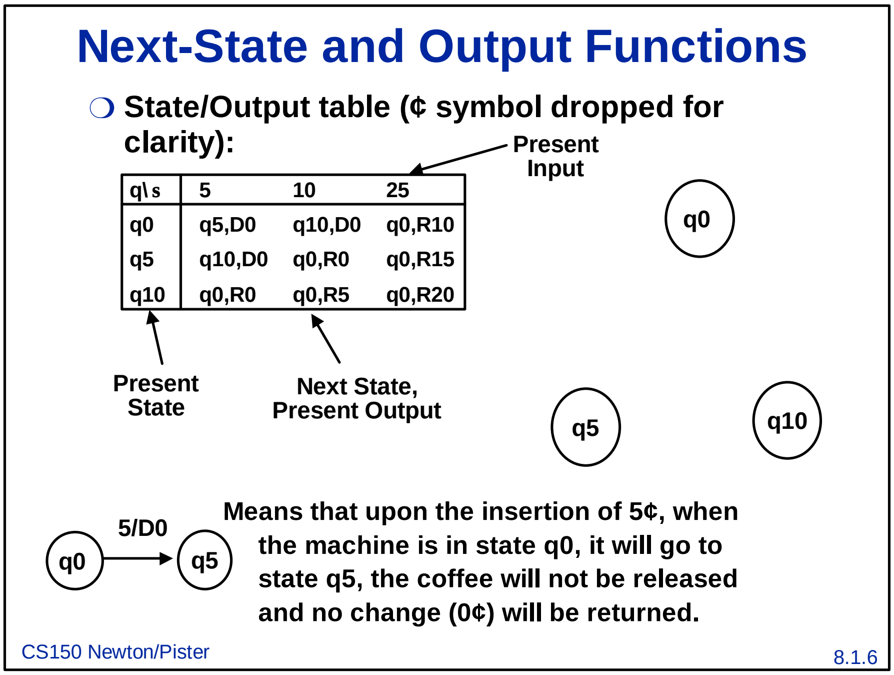
$$M_1 = (S_1, Q_1, Z_1, d, l_1)$$

Input
S_1 = { 5¢, 10¢, 25¢ }

Sequential
Machine

Present
State
Q_1 = { q0¢, q5¢, q10¢ }

Output
Z_1 = { D0¢, R0¢, R5¢,
R10¢, R15¢, R20¢ }



Next-State and Output Functions

❍ **State/Output table (¢ symbol dropped for clarity):**

Present Input

q\s	5	10	25
q0	q5,D0	q10,D0	q0,R10
q5	q10,D0	q0,R0	q0,R15
q10	q0,R0	q0,R5	q0,R20

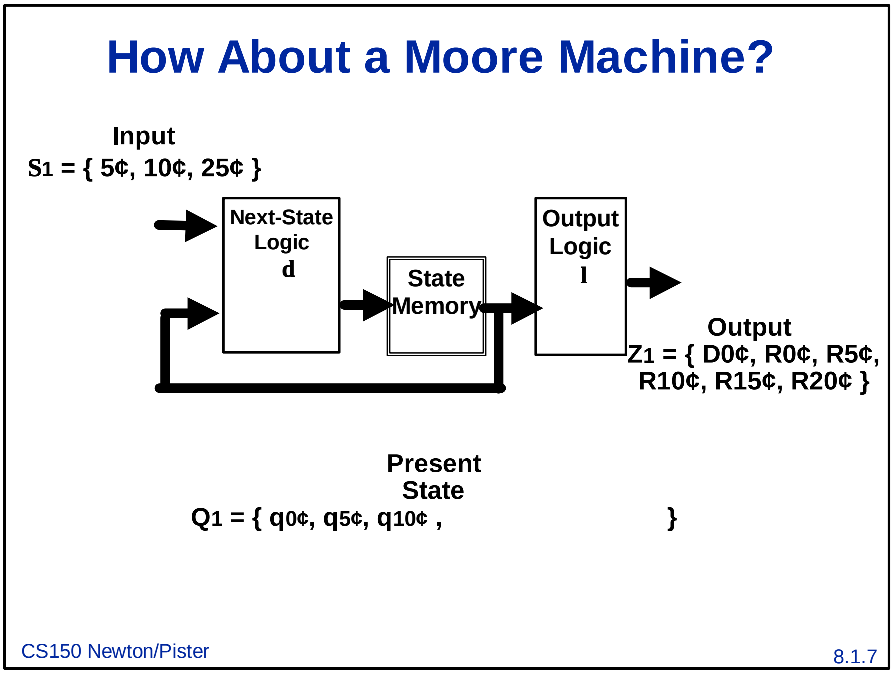
Present State

Next State, Present Output

q0
q5
q10

q0
5/D0
q5

Means that upon the insertion of 5¢, when the machine is in state q0, it will go to state q5, the coffee will not be released and no change (0¢) will be returned.



How About a Moore Machine?

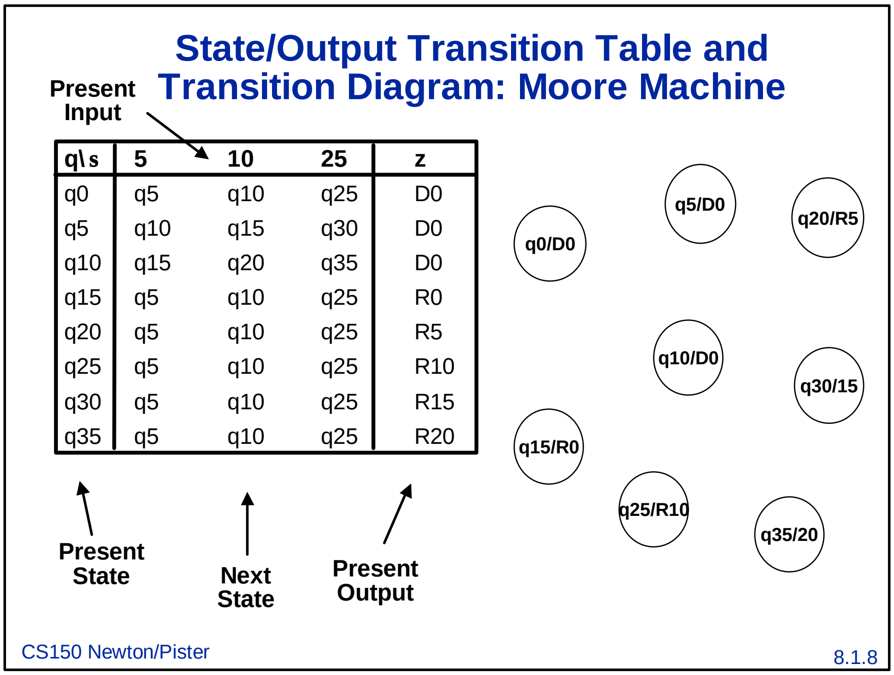
Input
S_1 = { 5¢, 10¢, 25¢ }

Output
Z_1 = { D0¢, R0¢, R5¢, R10¢, R15¢, R20¢ }

Present
State
Q_1 = { q0¢, q5¢, q10¢ , }

CS150 Newton/Pister

8.1.7



State/Output Transition Table and Transition Diagram: Moore Machine

Present
Input

q\ s	5	10	25	z
q0	q5	q10	q25	D0
q5	q10	q15	q30	D0
q10	q15	q20	q35	D0
q15	q5	q10	q25	R0
q20	q5	q10	q25	R5
q25	q5	q10	q25	R10
q30	q5	q10	q25	R15
q35	q5	q10	q25	R20

Present
State

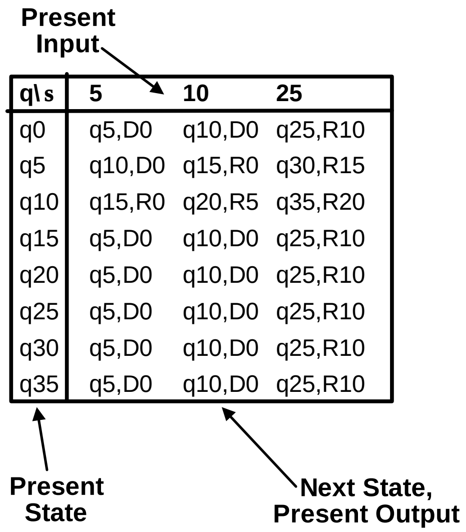
Next
State

Present
Output

CS150 Newton/Pister

8.1.8

Conversion to Mealy Machine



Present Input

q\s	5	10	25
q0	q5,D0	q10,D0	q25,R10
q5	q10,D0	q15,R0	q30,R15
q10	q15,R0	q20,R5	q35,R20
q15	q5,D0	q10,D0	q25,R10
q20	q5,D0	q10,D0	q25,R10
q25	q5,D0	q10,D0	q25,R10
q30	q5,D0	q10,D0	q25,R10
q35	q5,D0	q10,D0	q25,R10

Present State

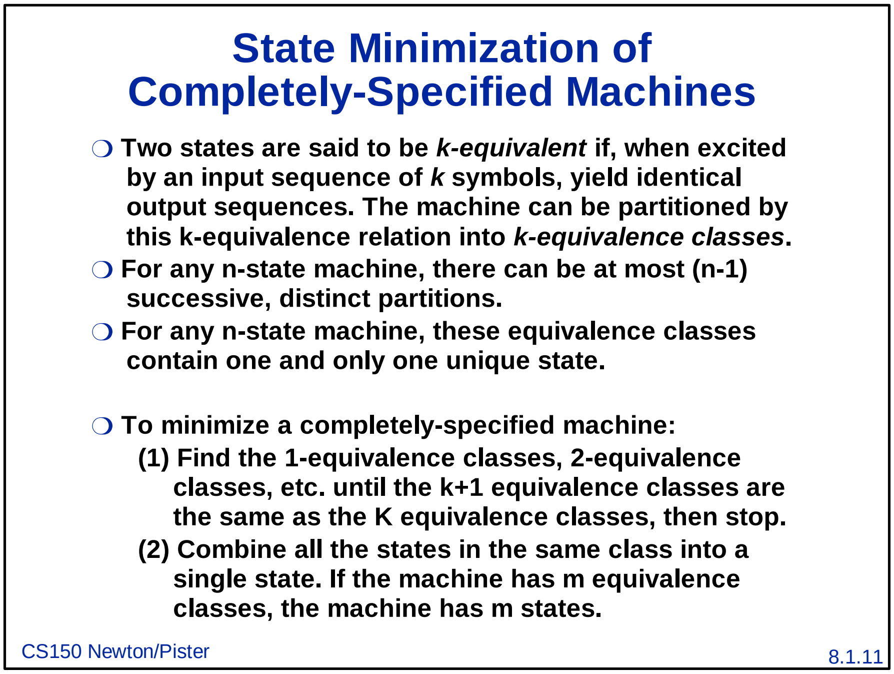
Next State, Present Output

Machine Equivalence

- **Let *qa* and *qb* be two states of machines *Ma* and *Mb* respectively. States *qa* and *qb* are said to be *equivalent* iff, starting with *qa* and *qb*, for *any* sequence of input symbols applied to the two machines, the output sequences are identical. If *qa* and i are not identical, they are said to be *distinguishable.***

- **Let *Ma* and *Mb* be two sequential machines. *Ma* and *Mb* are said to be *eqivalent* iff for every state of *Ma* there exists at least one equivalent state in *Mb*, and vice versa. Similarly, if *Ma* and *Mb* are not equivalent we say they are *distinguishable*.**

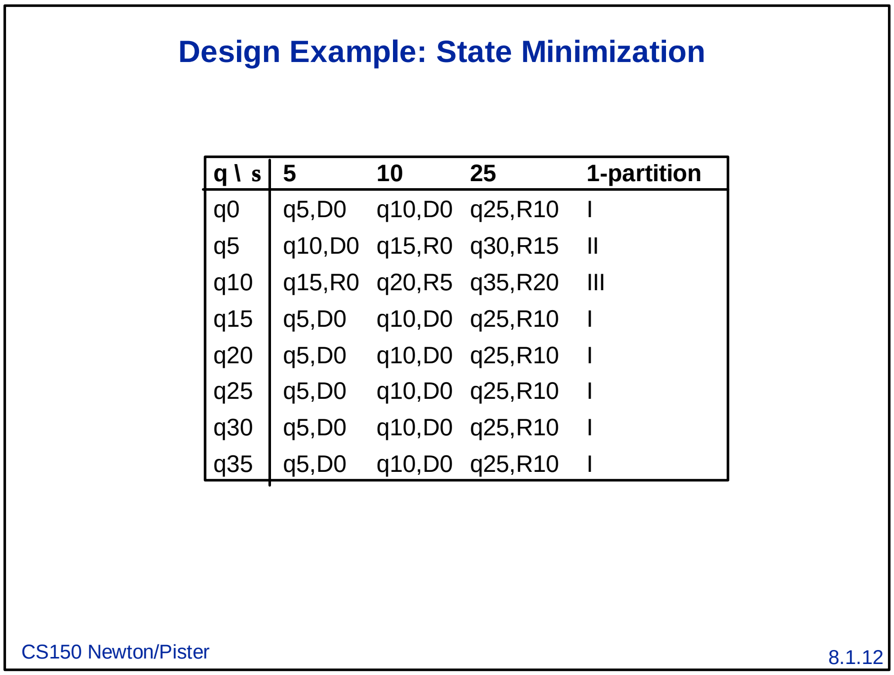
- **Two states *qa* and *qb* are equivalent if:**
  - **(1) *qa* and *qb* produce the same output values (for Mealy machines, they must produce the same outputs for all legal input conditions).**
  - **(2) For each input combination, *qa* and *qb* must have the same next state, or equivalent next states.**



State Minimization of Completely-Specified Machines

- ❍ Two states are said to be ***k-equivalent*** if, when excited by an input sequence of ***k*** symbols, yield identical output sequences. The machine can be partitioned by this k-equivalence relation into ***k-equivalence classes***.
- ❍ For any n-state machine, there can be at most (n-1) successive, distinct partitions.
- ❍ For any n-state machine, these equivalence classes contain one and only one unique state.

- ❍ To minimize a completely-specified machine:
  - (1) Find the 1-equivalence classes, 2-equivalence classes, etc. until the k+1 equivalence classes are the same as the K equivalence classes, then stop.
  - (2) Combine all the states in the same class into a single state. If the machine has m equivalence classes, the machine has m states.



Design Example: State Minimization

q \ s	5	10	25	1-partition
q0	q5,D0	q10,D0	q25,R10	I
q5	q10,D0	q15,R0	q30,R15	II
q10	q15,R0	q20,R5	q35,R20	III
q15	q5,D0	q10,D0	q25,R10	I
q20	q5,D0	q10,D0	q25,R10	I
q25	q5,D0	q10,D0	q25,R10	I
q30	q5,D0	q10,D0	q25,R10	I
q35	q5,D0	q10,D0	q25,R10	I

Design Example: State Minimization

1-partition	q \ s	5	10	25	2-partition
I	q0	q5,D0	q10,D0	q25,R10	
	q15	q5,D0	q10,D0	q25,R10	
	q20	q5,D0	q10,D0	q25,R10	
	q25	q5,D0	q10,D0	q25,R10	
	q30	q5,D0	q10,D0	q25,R10	
	q35	q5,D0	q10,D0	q25,R10	
II	q5	q10,D0	q15,R0	q30,R15	
III	q10	q15,R0	q20,R5	q35,R20	

State Assignment

q\s	5	10	25
q0	q5,D0	q10,D0	q0,R10
q5	q10,D0	q0,R0	q0,R15
q10	q0,R0	q0,R5	q0,R20

- **We must assign codes to symbolic values. Codes for input and output symbols are usually "given" so we must determine codes for the state symbols. This process is called state assignment or state coding. If binary storage elements are used we need:**

$$\lceil \log 2(N_s) \rceil < N_m < N_s$$

Design Example: State Assignment Minimum-Length Code

- For this example, $2 < N_m < 3$. If we choose $N_m = 2$, and assign codes randomly, then we have the state table:

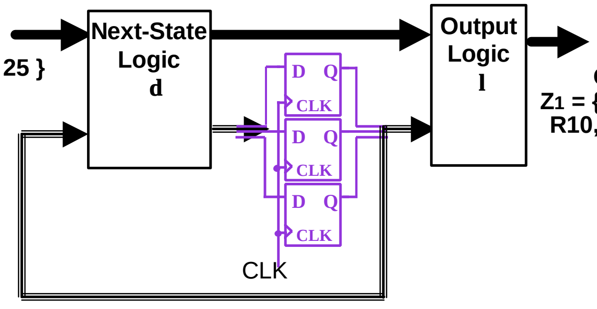
q\σ	5	10	25
00	**01**,D0	**11**,D0	**00**,R10
01	**11**,D0	**00**,R0	**00**,R15
11	**00**,R0	**00**,R5	**00**,R20
10	**??,??**	**??,??**	**??,??**

unused state

Implementation Using D Flip-Flops

- Can use positive-edge-triggered D flip-flop directly to implement storage element:

Input
S_1 = { 5, 10, 25 }



Output
Z_1 = { D0, R0, R5, R10, R15, R20 }

Present State
Q_1 = { 00,01,11}

Design Example: State Assignment One-Hot Code

- **For this example, 2 < N_m < 3. If we choose N_m = 3, and assign codes randomly but where exactly one bit of the code is "1" for each valid state, then we have the state table:**

q\s	5	10	25
001	010,D0	100,D0	001,R10
010	100,D0	001,R0	001,R15
100	001,R0	001,R5	001,R20
000	???,??	???,??	???,??
011	???,??	???,??	???,??
101	???,??	???,??	???,??
110	???,??	???,??	???,??
111	???,??	???,??	???,??

unused states

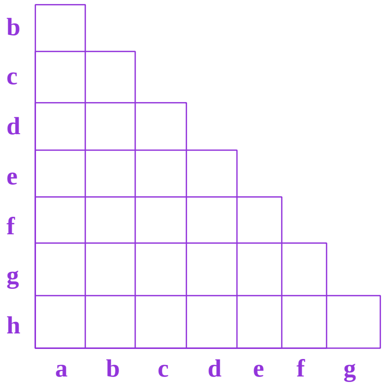
Steps to FSM Design

- Construct a state/output table from the word description (or a state graph).
- *State Minimization:* Minimize the number of states (usually helps).
- *State Assignment:* Coose a set of state variables and assign codes to named states.
- Substitute the state-variable combinations into the state/output table to create a *transition/output table* that shows the desired next-state variable combination for each state/input combination.
- Choose a flip-flop type (e.g. D, J-K, T) for the state memory.
- Construct an *excitation table* that shows the excitation values required to obtain the desired next-state value for each state/input combination.
- Derive *excitation equations* from excitation table.
- Derive *output equations* from transition/output table.
- Draw *logic diagram* that shows combinational next-state and output functions as well as flip-flops.

State Minimization Using an Implication Table

❍ Build a compatibility checking table in a ladder shape, as shown, and label each row q2, q3, ... qn and column q1, q2, qn-1 (no need for diagonal).

q\s	0	1	z
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



State Minimization Using am Implication Table: Summary of Approach

➔ Construct an implication table which contains a square for each pair of states. Label each row q2, q3, ... q_n and column q1, q2, q_{n-1} (no need for diagonal).

➔ Compare each each pair of rows in the state table. If the outputs associated with states i and j are different, put an ✗ in square i-j to indicate that i ≢ j (trivial non-equivalence). If the outputs and the next states are the same, put a ✓ in square i-j to indicate i ≡ j (trivial equivalence).

➔ In all other squares, put state-pairs that must be equivalent if states i-j are to be equivalent (if the next states of i and j are m and n for some input $\sigma 1$, then m-n is an implied pair and goes in square i-j).

➔ Go through the non-✓ and non-✗ squares, one at a time. If square i-j contains an implied pair and square m-n contains an ✗ , then i ≡ j so put an ✗ in i-j as well.

➔ If any ✗ 's were added in the last step, repeat it until no more ✗ 's are added. For each square i-j which not containing an ✗ , i ≢ j.

Implication Table Example: Pass 0

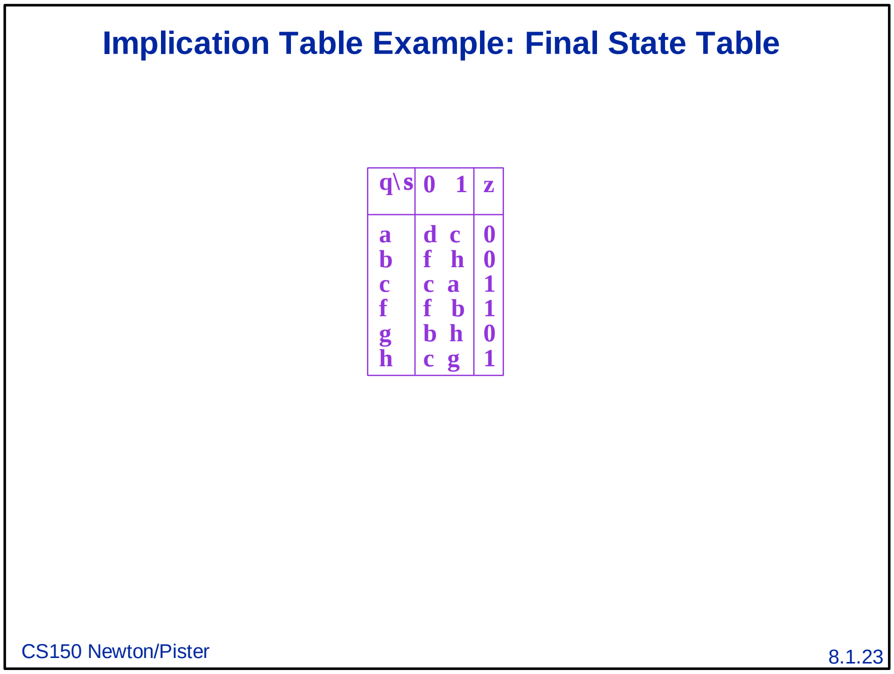
q\s	0	1	z
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

b	d-f c-h						
c	×	×					
d	c-e	a-f e-h	×				
e	×	×	a-d	×			
f	×	×	e-f b-d	×	c-f a-b		
g	b-d c-h	b-f	×	a-b e-h	×	×	
h	×	×	c-e d-g	×	a-g	c-f b-g	×
	a	b	c	d	e	f	g

Implication Table Example: Pass 1 and Pass 2

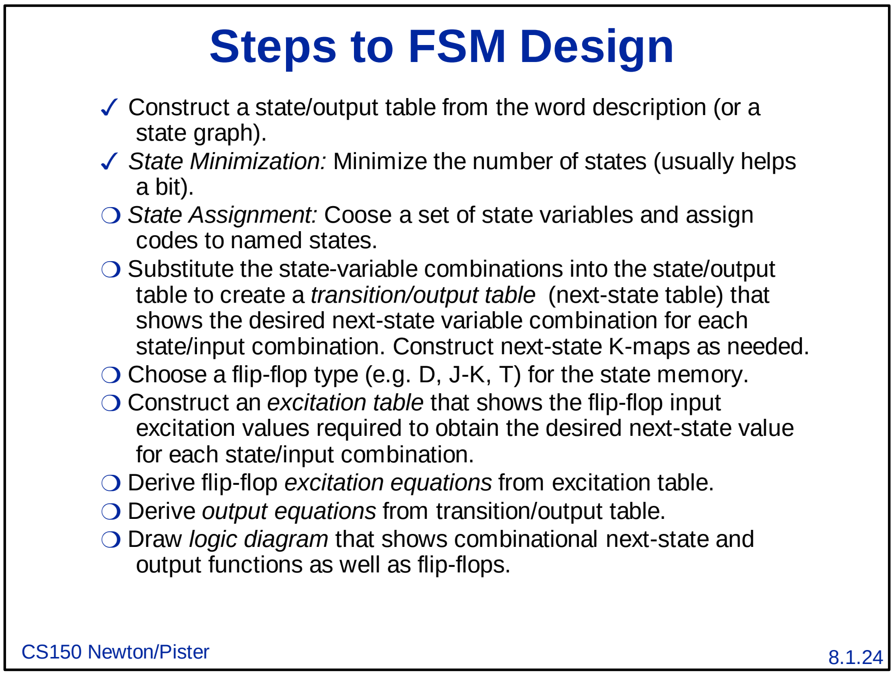
b	~~d-f~~ ~~c-h~~						
c	×	×					
d	c-e	~~a-f~~ ~~e-h~~	×				
e	×	×	a-d	×			
f	×	×	~~e-f~~ ~~b-d~~	×	~~c-f~~ ~~a-b~~		
g	b-d c-h	~~b-f~~	×	~~a-b~~ ~~e-h~~	×	×	
h	×	×	c-e d-g	×	a-g	~~c-f~~ ~~b-g~~	×
	a	b	c	d	e	f	g

b	~~d-f~~ ~~c-h~~						
c	×	×					
d	c-e	~~a-f~~ ~~e-h~~	×				
e	×	×	a-d	×			
f	×	×	~~e-f~~ ~~b-d~~	×	~~c-f~~ ~~a-b~~		
g	~~b-d~~ ~~c-h~~	~~b-f~~	×	~~a-b~~ ~~e-h~~	×	×	
h	×	×	~~c-e~~ ~~d-g~~	×	~~a-g~~	~~c-f~~ ~~b-g~~	×
	a	b	c	d	e	f	g



Implication Table Example: Final State Table

q\s	0	1	z
a	d	c	0
b	f	h	0
c	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



Steps to FSM Design

- ✓ Construct a state/output table from the word description (or a state graph).
- ✓ *State Minimization:* Minimize the number of states (usually helps a bit).
- ❍ *State Assignment:* Coose a set of state variables and assign codes to named states.
- ❍ Substitute the state-variable combinations into the state/output table to create a *transition/output table* (next-state table) that shows the desired next-state variable combination for each state/input combination. Construct next-state K-maps as needed.
- ❍ Choose a flip-flop type (e.g. D, J-K, T) for the state memory.
- ❍ Construct an *excitation table* that shows the flip-flop input excitation values required to obtain the desired next-state value for each state/input combination.
- ❍ Derive flip-flop *excitation equations* from excitation table.
- ❍ Derive *output equations* from transition/output table.
- ❍ Draw *logic diagram* that shows combinational next-state and output functions as well as flip-flops.

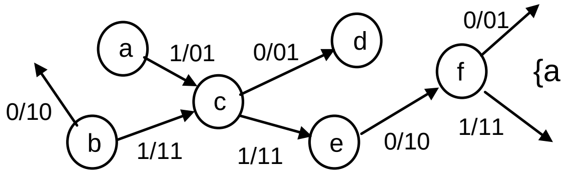
Guidelines for State Assignment

❍ The idea of the following heuristics is to try to get the 1's together (in the same implicant) on the flip-flop input maps. This method does not apply to all problems and even when it is applicable it does not guarantee a minimum soultion.

➔ States which have the same next state, for a given input, should be given adjacent assignments ("fan-out oriented").

➔ States which are the next states of the same state should be given adjacent assignments ("fan-in oriented").

➔ Third priority, to simplify the output function, states which have the same output for a given input should be given adjacent assignments (this will help put the 1's together in the output K-maps; "output oriented").



{a,b}; {d,e}; {c,f}

But How Do You Actually Do It?

❍ Write down all of the states that should be given adjacent assignments according to the criteria above ("assignment constraints", or "face embedding constraints.") Then, using a Karnaugh-map, try to satisfy as many of them as possible (or use a computer program which does it: Kiss, Nova, Mustang, Jedi). Some guidelines to help are:

➔ Assign the starting state to the "0" square on the map (picking a different square doesn't help, since all squares have the same number of adjacencies and it's easier to reset to "0").

➔ Fanout-oriented guidelines and adjacency conditions required more than once should be satisfied first.

➔ When guidelines require that 3 or 4 states be adjacent, these states should be placed within a group of 4 on the assignment map.

➔ If there are only a few outputs, the output guideline should be applied last. If there are lots of outputs and only a few states, then give more weight to the third guideline.

State Assignment: Design Example

q \ s	0	1
q0	q1,0	q2,0
q1	q3,0	q2,0
q2	q1,0	q4,0
q3	q5,0	q2,0
q4	q1,0	q6,0
q5	q5,1	q2,0
q6	q1,0	q6,1

- ❍ Consider the state table opposite:
- ➔ *Guideline 1:* {q0,q2,q4,q6} since all have q1 as next-state with input 0. Similary {q0,q1,q3,q5}; {q3,q5}; {q4,q6}.
- ➔ *Guideline 2:* {q1,q2} since next-states of q0. Similarly {q2,q3}; {q1,q4}; {q2,q5} twice; {q1,q6} twice.
- ➔ *Guideline 3:* would not be worth using here. We already have a lot of constraints and their is only one output, mostly 0.

CS150 Newton/Pister

8.1.27

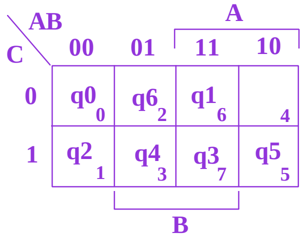
State Assignment: Design Example

- ❍ Given the adjacency constraints:

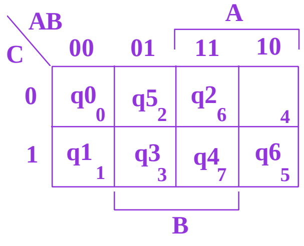
1: {q0,q2,q4,q6}; {q0,q1,q3,q5}; {q3,q5}; {q4,q6}.

2: {q1,q2}; {q2,q3}; {q1,q4}; {q2,q5} twice; {q1,q6} twice.

- ➔ Choose number of flip-flops: 6 states so need at least 3 and no more than 6. Try with 3 -A, B, C say.
- ➔ Task is now to choose assignment of 3-bit (ABC) state codes to q1-q6 so that as many of the above constraints as possible are satisfied, in the order stated earlier.



C \ AB	00	01	11	10
0	q0 $_{0}$	q6 $_{2}$	q1 $_{6}$	$_{4}$
1	q2 $_{1}$	q4 $_{3}$	q3 $_{7}$	q5 $_{5}$



C \ AB	00	01	11	10
0	q0 $_{0}$	q5 $_{2}$	q2 $_{6}$	$_{4}$
1	q1 $_{1}$	q3 $_{3}$	q4 $_{7}$	q6 $_{5}$

CS150 Newton/Pister

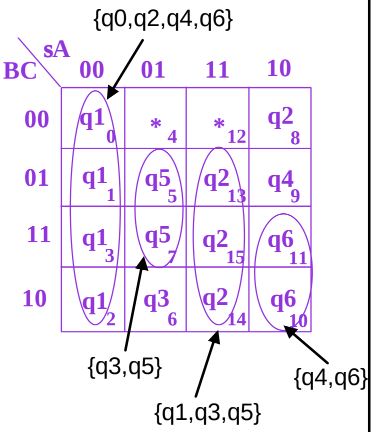
8.1.28

State Assignment: Design Example

- ❍ Assignments achieved by trial-and-error (question: would we have been able to satisfy more constraints using 4 flip-flops instead of 3?).
- ❍ Top assignment leads to codes:

q0 = 000, q1 = 110, q2 = 001, q3 = 111, q4 = 011, q5 = 101, q6 = 010

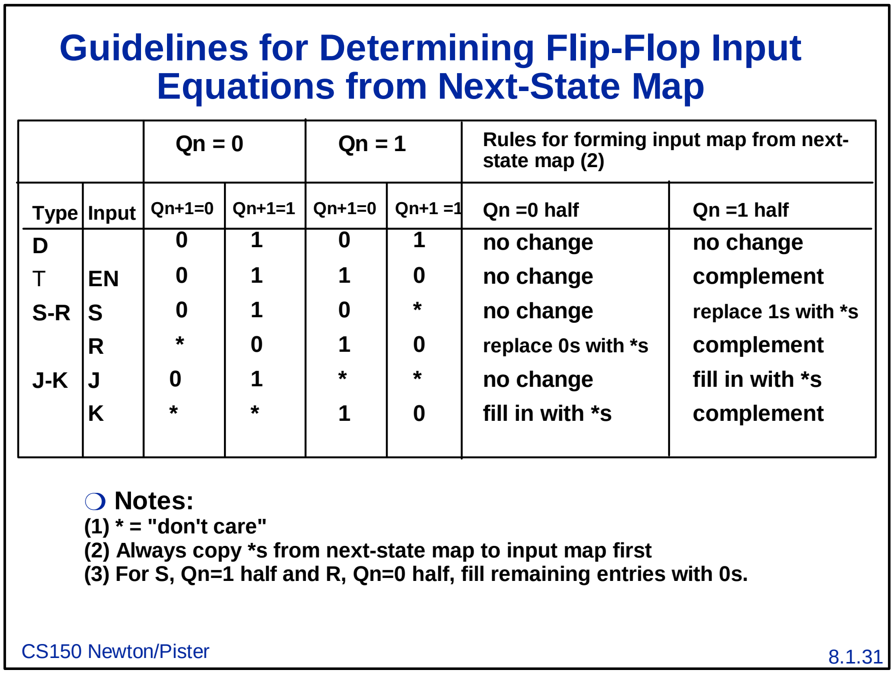
- ❍ Now we can construct the next-state maps for the assignment.



BC \ sA	00	01	11	10
00	q1 0	* 4	* 12	q2 8
01	q1 1	q5 5	q2 13	q4 9
11	q1 3	q5 7	q2 15	q6 11
10	q1 2	q3 6	q2 14	q6 10

Steps to FSM Design

- ✓ Construct a state/output table from the word description (or a state graph).
- ✓ *State Minimization:* Minimize the number of states (usually helps a bit).
- ✓ *State Assignment:* Coose a set of state variables and assign codes to named states.
- ✓ Substitute the state-variable combinations into the state/output table to create a *transition/output table* (next-state table) that shows the desired next-state variable combination for each state/input combination. Construct next-state K-maps as needed.
- ❍ Choose a flip-flop type (e.g. D, J-K, T) for the state memory.
- ❍ Construct an *excitation table* that shows the flip-flop input excitation values required to obtain the desired next-state value for each state/input combination.
- ❍ Derive flip-flop *excitation equations* from excitation table.
- ❍ Derive *output equations* from transition/output table.
- ❍ Draw *logic diagram* that shows combinational next-state and output functions as well as flip-flops.



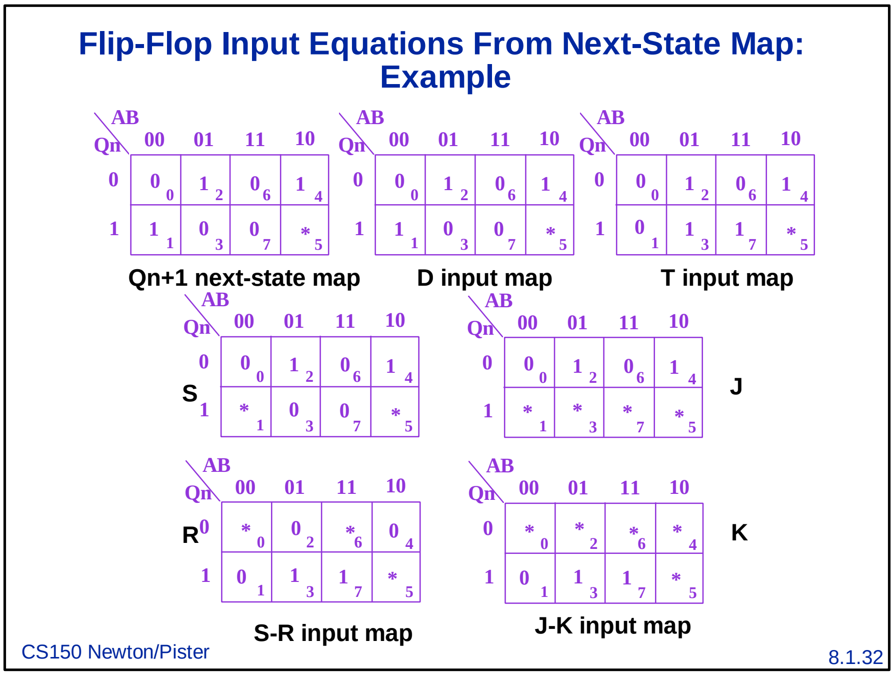
Guidelines for Determining Flip-Flop Input Equations from Next-State Map

		Qn = 0		Qn = 1		Rules for forming input map from next-state map (2)	
Type	Input	Qn+1=0	Qn+1=1	Qn+1=0	Qn+1 =1	Qn =0 half	Qn =1 half
D		0	1	0	1	no change	no change
T	EN	0	1	1	0	no change	complement
S-R	S	0	1	0	*	no change	replace 1s with *s
	R	*	0	1	0	replace 0s with *s	complement
J-K	J	0	1	*	*	no change	fill in with *s
	K	*	*	1	0	fill in with *s	complement

❍ **Notes:**

(1) * = "don't care"
(2) Always copy *s from next-state map to input map first
(3) For S, Qn=1 half and R, Qn=0 half, fill remaining entries with 0s.

CS150 Newton/Pister 8.1.31



Flip-Flop Input Equations From Next-State Map: Example

Qn+1 next-state map

Qn \ AB	00	01	11	10
0	0_{0}	1_{2}	0_{6}	1_{4}
1	1_{1}	0_{3}	0_{7}	$*_{5}$

D input map

Qn \ AB	00	01	11	10
0	0_{0}	1_{2}	0_{6}	1_{4}
1	1_{1}	0_{3}	0_{7}	$*_{5}$

T input map

Qn \ AB	00	01	11	10
0	0_{0}	1_{2}	0_{6}	1_{4}
1	0_{1}	1_{3}	1_{7}	$*_{5}$

S

Qn \ AB	00	01	11	10
0	0_{0}	1_{2}	0_{6}	1_{4}
1	$*_{1}$	0_{3}	0_{7}	$*_{5}$

J

Qn \ AB	00	01	11	10
0	0_{0}	1_{2}	0_{6}	1_{4}
1	$*_{1}$	$*_{3}$	$*_{7}$	$*_{5}$

R

Qn \ AB	00	01	11	10
0	$*_{0}$	0_{2}	$*_{6}$	0_{4}
1	0_{1}	1_{3}	1_{7}	$*_{5}$

K

Qn \ AB	00	01	11	10
0	$*_{0}$	$*_{2}$	$*_{6}$	$*_{4}$
1	0_{1}	1_{3}	1_{7}	$*_{5}$

S-R input map

J-K input map

CS150 Newton/Pister 8.1.32

Next-State Maps: Design Example

- ❍ Choose flip-flop types: D flip-flops
- ❍ Recall assignments:

  q0 = 000, q1 = 110, q2 = 001, q3 = 111, q4 = 011, q5 = 101, q6 = 010
- ❍ Construct D input maps from next-state map, substituting state codes.

BC \ A	00	01	11	10
00	1_{0}	$*_{4}$	$*_{12}$	0_{8}
01	1_{1}	1_{5}	0_{13}	0_{9}
11	1_{3}	1_{7}	0_{15}	0_{11}
10	1_{2}	1_{6}	0_{14}	0_{10}

An+1

BC \ A	00	01	11	10
00	1_{0}	$*_{4}$	$*_{12}$	0_{8}
01	1_{1}	0_{5}	0_{13}	1_{9}
11	1_{3}	0_{7}	0_{15}	1_{11}
10	1_{2}	1_{6}	0_{14}	1_{10}

Bn+1

BC \ sA	00	01	11	10
00	0_{0}	$*_{4}$	$*_{12}$	1_{8}
01	0_{1}	1_{5}	1_{13}	1_{9}
11	0_{3}	1_{7}	1_{15}	0_{11}
10	0_{2}	1_{6}	1_{14}	0_{10}

Cn+1

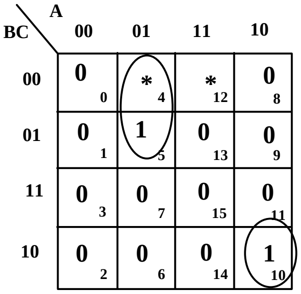
Steps to FSM Design

- ✓ Construct a state/output table from the word description (or a state graph).
- ✓ *State Minimization:* Minimize the number of states (usually helps a bit).
- ✓ *State Assignment:* Coose a set of state variables and assign codes to named states.
- ✓ Substitute the state-variable combinations into the state/output table to create a *transition/output table* (next-state table) that shows the desired next-state variable combination for each state/input combination. Construct next-state K-maps as needed.
- ✓ Choose a flip-flop type (e.g. D, J-K, T) for the state memory.
- ✓ Construct an *excitation table* that shows the flip-flop input excitation values required to obtain the desired next-state value for each state/input combination.
- ❍ Derive flip-flop *excitation equations* from excitation table.
- ❍ Derive *output equations* from transition/output table.
- ❍ Draw *logic diagram* that shows combinational next-state and output functions as well as flip-flops.

Next-State Maps: Summary of Example

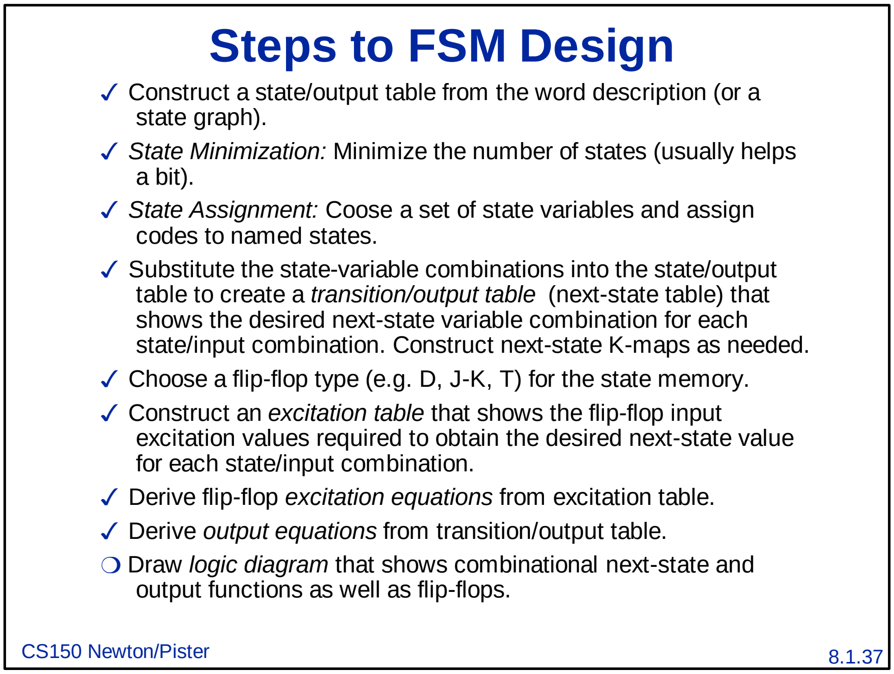
- Need 6 gates and 13 gate-inputs to implement the machine using this assignment.
- Straight binary assignment (q0=000, q1=001, etc.) would yield 10 gates and 39 gate-inputs.
- The approach gave good results in this example, but that is not always the case.

Derive Output Equations from Output Maps



BC \ A	00	01	11	10
00	0 0	* 4	* 12	0 8
01	0 1	1 5	0 13	0 9
11	0 3	0 7	0 15	0 11
10	0 2	0 6	0 14	1 10

Output map from Transition/Output Table



Steps to FSM Design

- ✓ Construct a state/output table from the word description (or a state graph).
- ✓ *State Minimization:* Minimize the number of states (usually helps a bit).
- ✓ *State Assignment:* Coose a set of state variables and assign codes to named states.
- ✓ Substitute the state-variable combinations into the state/output table to create a *transition/output table* (next-state table) that shows the desired next-state variable combination for each state/input combination. Construct next-state K-maps as needed.
- ✓ Choose a flip-flop type (e.g. D, J-K, T) for the state memory.
- ✓ Construct an *excitation table* that shows the flip-flop input excitation values required to obtain the desired next-state value for each state/input combination.
- ✓ Derive flip-flop *excitation equations* from excitation table.
- ✓ Derive *output equations* from transition/output table.
- ❍ Draw *logic diagram* that shows combinational next-state and output functions as well as flip-flops.

CS150 Newton/Pister 8.1.37