

## Sequential Logic Circuits – Part II (Realizing Sequential Circuits)

CIT 595  
Spring 2007

CIT 595

6 - 1

## Sequential Circuits

- Sequential Circuits are dependent on past behaviour
- They are built out of combinational logic and one or more memory/storage elements
- Examples seen so far: n-bit Register and m x n Memory
- Like combinational logic, we want a systematic way of analyzing/designing a sequential circuit

CIT 595

6 - 2

## Describing Behavior of Sequential Circuits

- We looked at how a fundamental memory element can store 1-bit of information
  - By grouping “n” 1-bit elements we can store n-bits of information
- We also observed that for 1-bit there are 2 possible values (states) that can be stored i.e. 0 or 1
  - Similarly, if we have 2-bit to store, then memory can be in any of the four states i.e. 00, 01, 10, 11 at particular point in time
- Therefore, we can conclude that for sequential system, the amount of information to be stored is “finite” and the “total number of different possible states” the information can be in also finite
  - Because of this a sequential circuit is also known as **Finite State Machine(FSM)** and the behavior of the circuit can be described using FSM model

CIT 595

6 - 3

## Finite State Machine (FSM)

- A Finite State Machine is an abstract model consisting of *finite set of states*, and the *transitions* between those states, along with the *actions* to be performed while in those state or during transitions
- The state stores information about the past i.e. it reflects input changes from the time system started to present moment
- State machines can be used for many other things beyond logic design and computer architecture
  - E.g. State machines can be designed to respond to a sequence of inputs, such as individual characters in a string
  - Broadly used in theory of computation (**D**eterministic **F**inite **A**utomaton )

CIT 595

6 - 4

## FSM for Computer Hardware Application

- For actual hardware implementation
  - requires memory element(s) to store state(s) and state is updated based on the present input with respect to clock
  - a block of combinational logic which determines the state transition
  - a second block of combinational logic that determines the output(s) of a FSM

CIT 595

6 - 5

## FSM Terminology

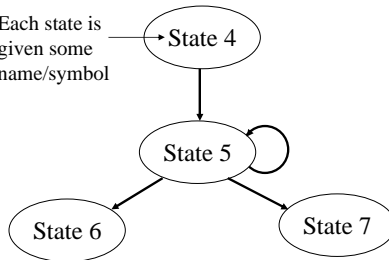
- *State Diagram*: Illustrates the form and function of a state machine. Usually drawn as a circle-and-arrow diagram
- *State*: is a unique configuration of information in a machine
- *Current/Present State*: configuration in which the machine is currently in
- *Next State*: The state to which the state machine makes the next transition, determined by the inputs present
- *Transition Arc*: A change from present state to next state

CIT 595

6 - 6

## State Diagram, State, Next State, Transition Arc

Each state is given some name/symbol



- For any given state, there is a **finite number of possible next states**

- For digital system, on each clock cycle, the FSM branches to the next state

- One of the possible next states becomes the new present state, depending on the inputs present on the clock cycle.

Note: From this diagram it can be deduced that if the present state is State 5, then the previous state was either State 4 or 5 and the next state must be either 5, 6, or 7.

CIT 595

6 - 7

## Two Kinds of FSM Model

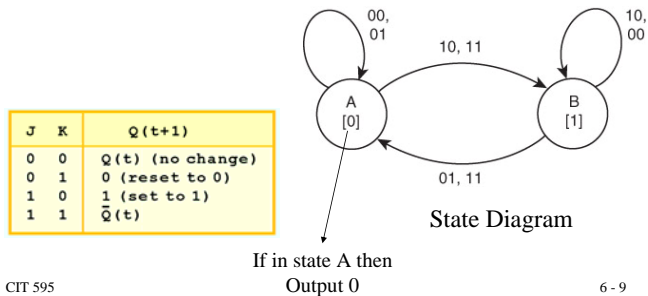
- Both machine model types stated below follow the basic characteristics of state machines, but differ in the way that outputs are produced
- **Moore Machine**
  - Outputs are independent of the inputs i.e. dependent on present state only
- **Mealy Machine**
  - Outputs are function of the present/current state and the present inputs

CIT 595

6 - 8

## Moore FSM

- Each state is associated with the output of the machine
  - Hence outputs are only dependent upon current state
- Example: JK Flip-Flop as Moore Machine

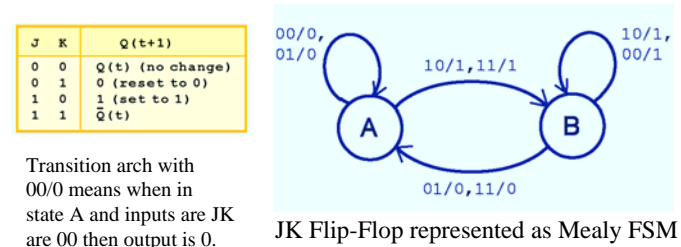


CIT 595

6 - 9

## Mealy FSM

- Outputs are produced as the machine makes a transition from one state to another
- Hence outputs are dependent on current state and current input



Transition arch with 00/0 means when in state A and inputs are JK are 00 then output is 0.

CIT 595

6 - 10

## Moore vs. Mealy

- Moore and Mealy FSMs can be functionally equivalent
- Mealy FSM has richer description and usually requires smaller number of states
  - Hence reduces the number of memory elements it needs
- Mealy FSM computes outputs as soon as inputs change
  - Mealy FSM responds one clock cycle sooner than equivalent Moore FSM

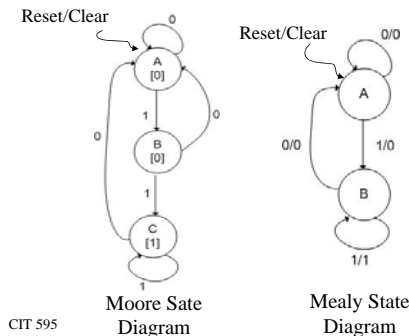
Lets see a real implementation ...

CIT 595

6 - 11

## Example: Sequence/Run Detector

- Lets say on serial line, a binary sequence is transmitted 1-bit at a time. At one end of the line there is a sequential circuit that has to output a "1" when it sees **at least two subsequent 1s**. E.g. 01101110001011 etc..



Here "A" is starting state in both Models. This allows the FSM to be set to known state at beginning (indicated by "reset").

CIT 595

6 - 12

## State Table

- Once you have conceptualized the problem in a state diagram (Moore or Mealy), you translate it to **State Table**
- Like Truth Table for Combinational Logic, a State Table enumerates the inputs, outputs with additional columns for current and next states of the sequential circuit/FSM

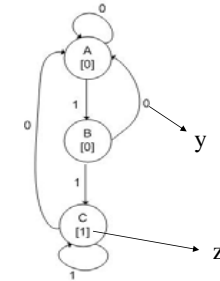
CIT 595

6 - 13

## State Table: Moore Sequence Detector

Input	Current State	Next State	Output
y	Q(t)	Q(t + 1)	z
0	A	A	0
1	A	B	0
0	B	A	0
1	B	C	0
0	C	A	1
1	C	C	1

Independent of y, only depends on current state



CIT 595

6 - 14

## State Encoding

- We need to convert letters A, B, C to particular binary combination of values such that they can be stored in flip-flops
  - Remember that our digital flip-flop is storing binary information
- Since there are 3 values (states), we can encode them in 2 bits
  - Implies that we have 2 Flip-Flops in sequential circuit
- Let A = 00, B = 01, C = 10
  - In this example we don't care about state 11

CIT 595

6 - 15

## State Table: Moore Sequence Detector (contd..)

y	Q(t)	Q(t + 1)	z
0	A	A	0
1	A	B	0
0	B	A	0
1	B	C	1
0	C	A	0
1	C	C	1



y	Q(t)		Q(t + 1)		z
	Q1	Q0	q1	q0	
0	0	0	0	0	0
1	0	0	0	1	0
0	0	1	0	0	0
1	0	1	1	0	0
0	1	0	0	0	1
1	1	0	1	0	1
0	1	1	x	x	x
1	1	1	x	x	x

**Q1 Q0: current state variables**

**q1 q0: next state variables**

**x - don't care**

Do you see where this is going???

• Now our State Table looking like Truth table

CIT 595

6 - 16

### Finding the Output and Next State Function

- Now, we draw up Kmaps like we did for combinational logic circuits to come up the function for output and next state
- But before that:
  - Need to pick the *type of flip-flop* to be used
  - The most straight forward choice is D Flip-Flops because for the flip-flop values to change to  $q_1q_0$  (next state),  $q_1$  and  $q_0$  values are simply clocked into the flip-flops as inputs to become the new state
  - Other than D Flip-Flop, need to do something extra – see an example later

CIT 595

6 - 17

### Kmap for Moore Sequence Detector (Using D Flip-Flop)

$q_1$

	$Q_1Q_0$			
$y$	00	01	11	10
0	0	0	x	0
1	0	1	x	1

$q_1 = yQ_0 + yQ_1$   
 $= y(Q_0 + Q_1)$

$y$	$Q(t)$		$Q(t+1)$		$z$
	$Q_1$	$Q_0$	$q_1$	$q_0$	
0	0	0	0	0	0
1	0	0	0	1	0
0	0	1	0	0	0
1	0	1	1	0	0
0	1	0	0	0	1
1	1	0	1	0	1
0	1	1	x	x	x
1	1	1	x	x	x

$q_0$

	$Q_1Q_0$			
$y$	00	01	11	10
0	0	0	x	0
1	1	0	x	0

$q_0 = y\bar{Q}_1\bar{Q}_0$

$z$

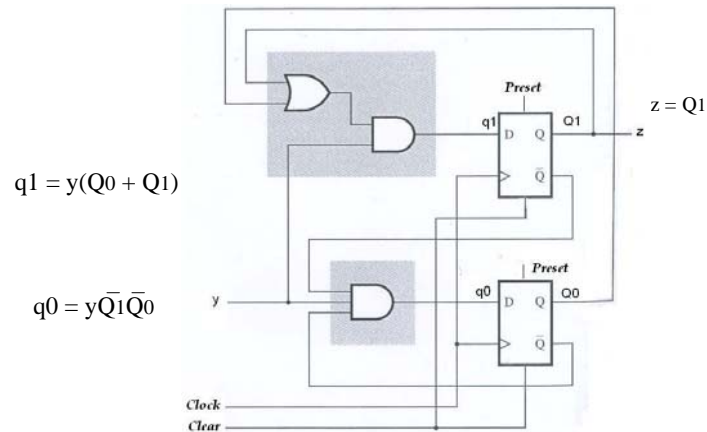
	$Q_0$	
	0	1
0	0	0
1	1	x

$z = Q_1$

CIT 595

6 - 18

### Logic Diagram of Moore Sequence Detector

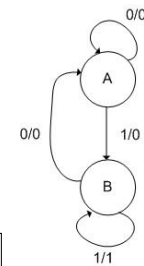


CIT 595

6 - 19

### State Table: Mealy Sequence Detector

$y$	$Q(t)$	$Q(t+1)$	$z$	Dependent on $y$ and current state
0	A	A	0	
1	A	B	0	
0	B	A	0	
1	B	B	1	



Inputs:  $y$  - 1 bit input,  $Q(t)$  - Current State of the Circuit  
Outputs:  $z$  - output,  $Q(t+1)$  - Next State of the Circuit

CIT 595

6 - 20

### State Table: Mealy Sequence Detector (contd..)

y	Q(t)	Q(t+1)	z
0	A	A	0
1	A	B	0
0	B	A	0
1	B	B	1

y	Q(t)	Q(t+1)	z
	Q	q	
0	0	0	0
1	0	1	0
0	1	0	0
1	1	1	1

**Q = current state variables**  
**q = next state variables**

Need only 1 flip-flop to represent 2 states.  
 Let A = 0 and B = 1.

CIT 595

6 - 21

### Kmap for Mealy Sequence Detector (Using D Flip-Flop)

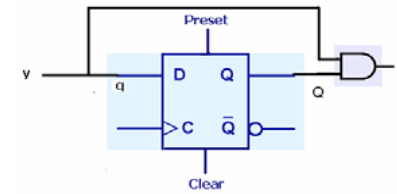
		q	
		0	1
y	0	0	0
	1	1	1

$q = y$

y	Q(t)	Q(t+1)	z
0	0	0	0
1	0	1	0
0	1	0	0
1	1	1	1

		q	
		0	1
z	0	0	0
	1	0	1

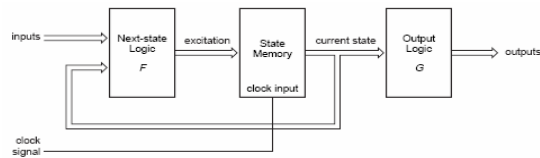
$z = yQ$



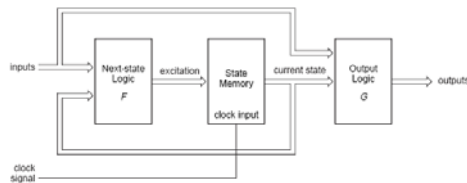
CIT 595

6 - 22

### General Mealy and Moore Machine Structure



General Moore Machine



General Mealy Machine

CIT 595

6 - 23

### For Flip-Flops other than D Flip-Flops

- We need to find the function for next state based on the required inputs to Flip-Flop
  - E.g. what should inputs JK for JK Flip-Flop be into order to make values  $q_1q_0$  to be the next state in sequence detector example?

(a) JK Flip-Flop				(b) SR Flip-Flop			
Q(t)	Q(t+1)	J	K	Q(t)	Q(t+1)	S	R
0	0	0	X	0	0	0	X
0	1	1	X	0	1	1	0
1	0	X	1	1	0	0	1
1	1	X	0	1	1	X	0

Excitation Table

CIT 595

6 - 24

## Sequence Detector Using JK Flip-Flop

(a) JK Flip-Flop

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

y	Q(t)		Q(t+1)		For q <sub>1</sub>		For q <sub>0</sub>		z
	Q <sub>1</sub>	Q <sub>0</sub>	q <sub>1</sub>	q <sub>0</sub>	J <sub>1</sub> :K <sub>1</sub>	J <sub>0</sub> :K <sub>0</sub>			
0	0	0	0	0	0:x	0:x	0		
1	0	0	0	1	0:x	1:x	0		
0	0	1	0	0	0:x	x:1	0		
1	0	1	1	0	1:x	x:1	0		
0	1	0	0	0	x:1	0:x	1		
1	1	0	1	0	x:0	0:x	1		
0	1	1	x	x	x:x	x:x	x		
1	1	1	x	x	x:x	x:x	x		

$J_1 = yQ_0$

$K_1 = \bar{y}Q_1$

$J_0 = y\bar{Q}_1$

$K_0 = 1$

CIT 595 6-25

## Steps for Designing Sequential Circuits

### Steps:

1. Conceptualize the problem into *state diagram*
2. Translate the state diagram into *State Table*
3. Decide the kind of *Flip-Flop* you want to use in order to determine next state function
  - If q is next state, then what should be input values of flip-flop in order to have the value q as next state (use excitation table)
4. Draw the Kmaps to determine the function
5. Draw the logic circuit based on Kmap expression (i.e. with flip-flops and combinational logic)

CIT 595

6-26

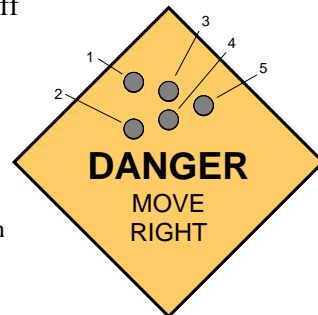
## Traffic Light Example

- A blinking traffic sign is controlled by a switch

➤ No lights if switch is off

➤ If switch is on then

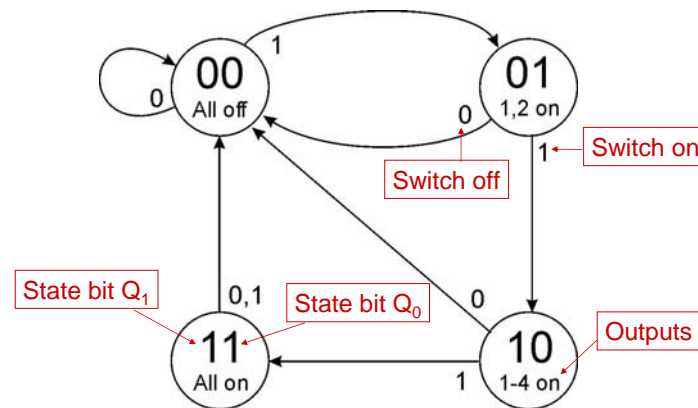
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- All off
- repeat as long as switch is turned on



CIT 595

6-27

## Traffic Sign State Diagram – Moore Machine

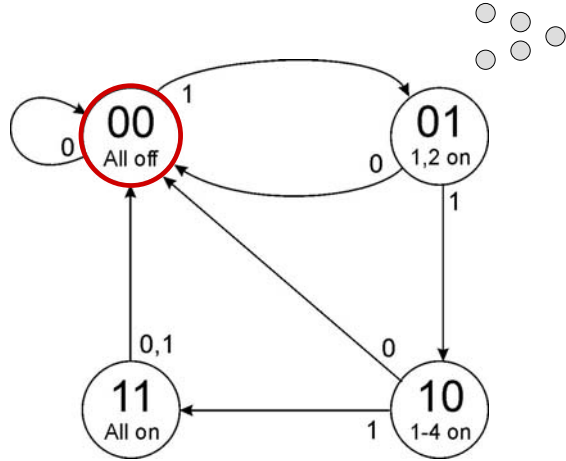


CIT 595

6-28

Transition on each clock cycle

Traffic Sign State Diagram: State 00

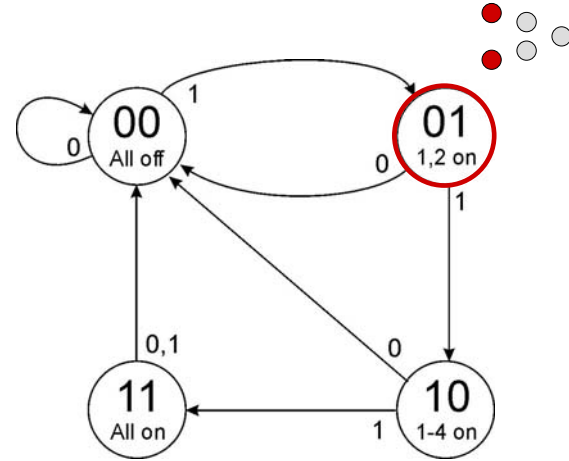


CIT 595

6 - 29

Transition on each clock cycle

Traffic Sign State Diagram: State 01

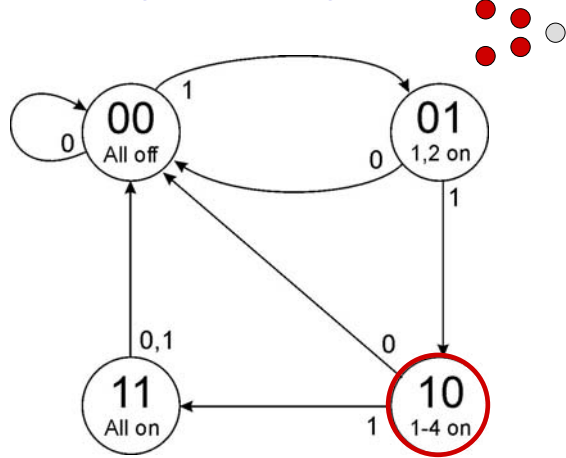


CIT 595

6 - 30

Transition on each clock cycle

Traffic Sign State Diagram: State 10

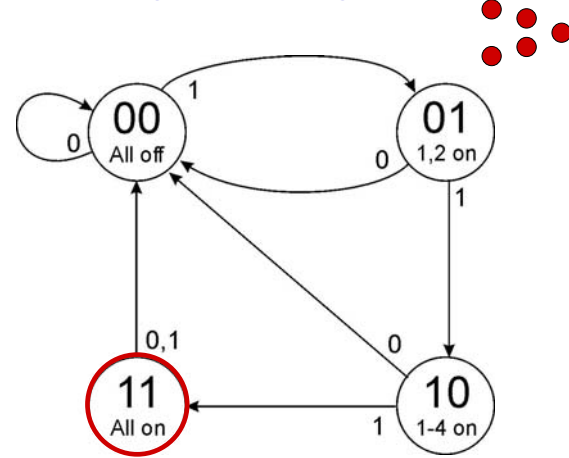


CIT 595

6 - 31

Transition on each clock cycle

Traffic Sign State Diagram: State 11

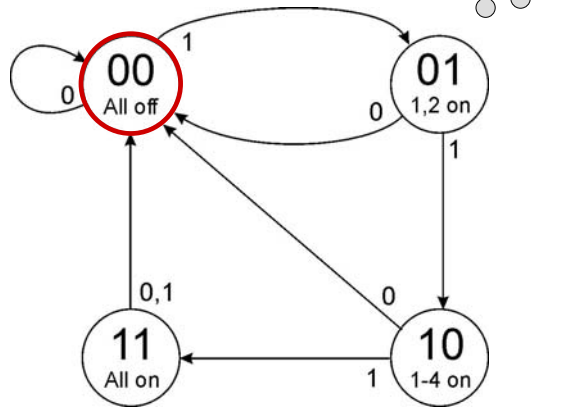


CIT 595

6 - 32

Transition on each clock cycle

### Traffic Sign State Diagram: State 00



CIT 595

6 - 33

Transition on each clock cycle

### Traffic Sign State Tables

Outputs depend only on current state:  $Q_1, Q_0$  (depend on state and input)

$Q_1$	$Q_0$	Z	Y	X
0	0	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	1	1	1

Next State:  $q_1, q_0$  (depend on state and input)

S	$Q_1$	$Q_0$	$q_1$	$q_0$
0	x	x	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

CIT 595

6 - 34

### Traffic Sign Kmap

$q_1$	S	$Q_1 Q_0$	00	01	11	10
0	0		0	0	0	0
1	1		0	1	0	1

$$q_1 = S \bar{Q}_1 Q_0 + S Q_1 \bar{Q}_0$$

$q_0$	S	$Q_1 Q_0$	00	01	11	10
0	0		0	0	0	0
1	1		1	0	0	1

$$q_0 = S \bar{Q}_0$$

Y	$Q_1$	$Q_0$	0	1
0	0		0	0
1	1		1	1

$$Y = Q_1$$

Z and X are straight forward:

$$Z = Q_1 + Q_0$$

$$X = Q_1 Q_0$$

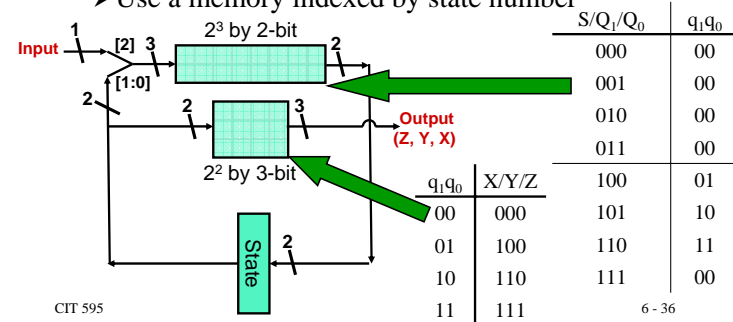
CIT 595

6 - 35

I'll let you finish drawing the logic diagram

### Programmable State Machines

- What if we want to change the pattern of the sign?
  - An alternative state machine implementation
  - Use a memory indexed by state number

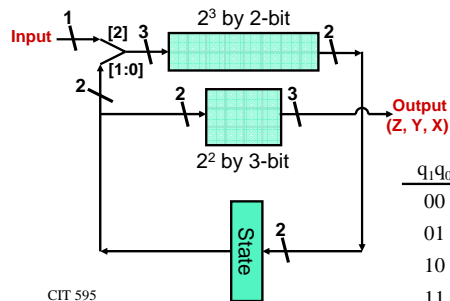


CIT 595

6 - 36

## Programmable State Machines

- Change to a two-state pattern:



S/Q <sub>1</sub> /Q <sub>0</sub>	q <sub>1</sub> q <sub>0</sub>
000	00
001	--
010	00
011	--
100	10
101	--
110	00
111	--

q <sub>1</sub> q <sub>0</sub>	X/Y/Z
00	000
01	--
10	111
11	--

CIT 595

6 - 37

## Designing Digital Circuits

- We have seen digital circuits from two points of view:
  - *Digital analysis* explores the relationship between a circuit's inputs and its outputs
  - *Digital synthesis* creates logic diagrams using the values specified in a truth/stable table
- Digital systems designers must also be mindful of the physical behaviors of circuits to include minute propagation delays that occur between the time when a circuit's inputs are energized and when the output is accurate and stable

CIT 595

6 - 38

## Designing Digital Circuits (contd..)

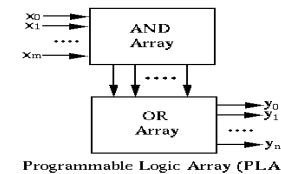
- Digital designers rely on specialized software to create efficient circuits
  - After few states Mealy and Moore Models become cumbersome
  - Thus, software is an enabler for the construction of better hardware
  - E.g. Verilog is Hardware Descriptive Language (HDL) where you can synthesize hardware components as if you writing a high-level language

CIT 595

6 - 39

## Designing Digital Circuits (contd..)

- Programmable Logic Devices (PLDs)
  - Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture
  - Example: PLA



- Other devices PAL, FPGAs are most commonly used for fast prototyping than actually working at the transistor level

CIT 595

6 - 40

## Designing Digital Circuits (contd..)

- When we need to implement a simple, specialized algorithm and its execution speed must be as fast as possible
  - Hardware solution is often preferred
- This is the idea behind *embedded systems*, which are small special-purpose computers that we find in many everyday things e.g. microwaves, cell phones etc..
- Embedded systems require special programming that demands an understanding of the operation of digital circuits, the basics of which you have learned chapter 3

CIT 595

6 - 41

## Aside: Potential Project Topic

- Embedded Systems
  - Software Tools for digital circuit designs e.g. Verilog, VHDL
  - Programmable Logic Devices and the companies that make them
  - Selling hardware designs as Open Source Hardware (HDL code sharing) – what about IP (intellectual property)?

CIT 595

6 - 42

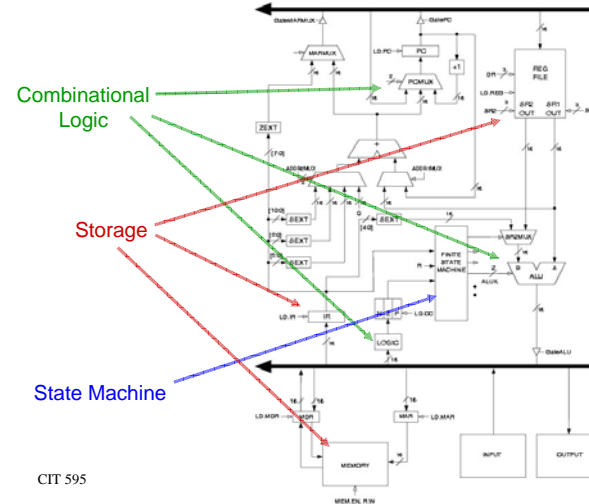
## From Logic to Data Path

- The data path of a computer is all the logic used to process information
  - See the data path of the LC-3 on next slide
- **Combinational Logic**
  - Decoders help convert instructions into control signals
  - Multiplexers help select inputs and outputs
  - ALU (Arithmetic and Logic Unit) perform operations on data
- **Sequential Logic**
  - Memory for storage
  - State machine (Control Unit)

CIT 595

6 - 43

## LC-3 Data Path



CIT 595

6 - 44

## Looking Forward...

- We've touched on basic digital logic
  - Transistors
  - Gates
  - Storage (latches, flip-flops, memory)
  - Sequential Circuits - State machines
- Seen some simple circuits
  - Shifter, mux, decoder, adder/subtractor, 2-bit ALU
  - Register, 4 x 3 Memory, Sequence Detector
  - Hard-coded traffic sign state machine
  - Programmable traffic sign state machine
- Up next: Putting it all together - a computer as a (simple?) state machine