# Logic Design ( Part 5)
## Sequential Logic Devices & Sequential Circuits



1

## Combinational vs. Sequential

- Combinational Circuit
  - always gives the same output for a given set of inputs
    - ex: adder always generates sum and carry, regardless of previous inputs
- Sequential Circuit
  - stores information
  - output depends on stored information (state) plus input
    - so a given input might produce different outputs, depending on the stored information

2

2

## Sequential Circuits – The agenda

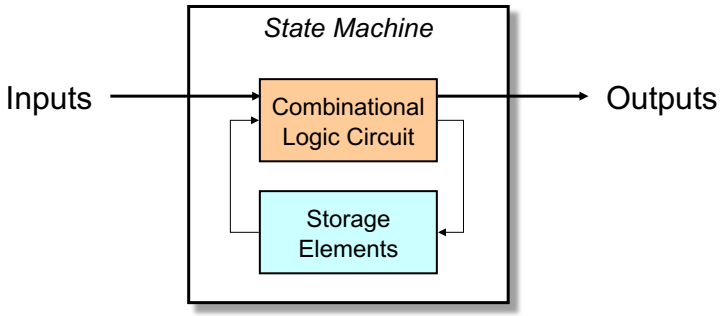
- Definition of sequential circuits
  - Components of a sequential circuit
- common storage 'devices' (built from latches)
  - Register
  - Memory
- synchronization using a CLOCK
- Modifying latches to work with a clock…Flip Flops
  - These become the basic unit of storage in sequential circuits
- Designing sequential circuits – methodology
  - Finite state machine diagrams
  - Mapping to truth table…..build circuit

3

3

## Sequential Circuit Schematic & State Machine

- sequential circuit
  - Combines combinational logic with storage
  - "Remembers" state, and changes output (and state) based on inputs and current state




**1.combinational circuit to compute output and next state**
**2.storage elements to store state**

4

4

## Sequential Circuits: Finite State Machines

- The behavior of sequential circuits can be expressed using characteristic tables or finite state machines (FSMs).
  - FSMs consist of a set of nodes that hold the states of the machine and a set of arcs that connect the states.
  - Directed graph to represent a FSM
- Moore and Mealy machines are two types of FSMs that are equivalent.
  - They differ only in how they express the outputs of the machine.
  - Moore machines place *outputs on each node/state*
    - Associate an output with each state
  - Mealy machines present their *outputs on the transitions.*

5

5

## FSM Design Process

- The first step is to model the behavior of the machine
  - Based on problem statement
  - Identify what the inputs are
  - Identify the outputs
  - Determine what needs to be stored to capture the "state" of the machine
- Represented as a graph – finite state diagram
  - Nodes: States – a state stores summary of events (until current time)
  - Edges: Transition from current state to next state
    - Based on input and current state
    - Computed by combinational logic
  - Outputs: Using Moore machine, determine value of outputs at each state

6

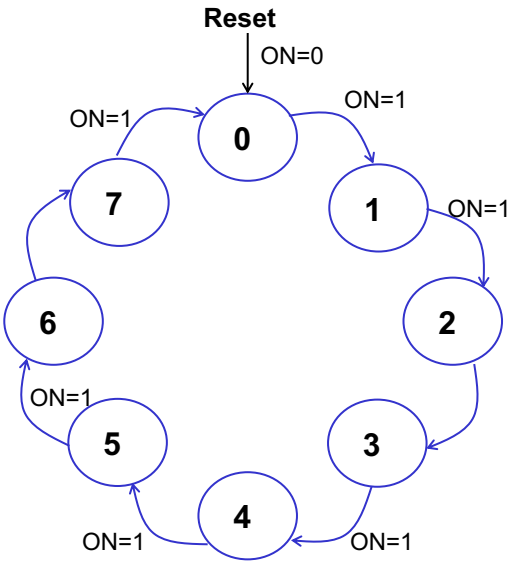
6

## Design a Counter: counts from 0 to 7

7

7

## Finite State Machine Representation of Counter: Counter to count from 0 to 7 (while switch is ON)




Bubbles represent all possible "states" for the machine

Arrows show movement from one state to the next while switch ON=1.

If ON=0, FSM goes to state 0

*Transitions occur at pulse of the clock*

8

8

## Example: A Vending Machine

- Accept user input (coins), when total is at least 50 cents dispense output (candy)
  - We will not model the change to be returned, and only care if the input is at least 50 cents
- Input valid coins:
  - Q (25cents) D (10) or N (5)
- What should it keep track of ?
  - current total
  - Is it 50 cents or more ?
- When it reaches 50 or more:
  - Generate output
- States of the machine ?
  - What should each state capture ?
  - How many states ?

9

9

## Sequential Logic

- Where do we start:
- Build a device, using combinational logic devices, to store a value…*done!!*
  - D Latch ( and RS Latch) – stores 1 bit
  - concept of memory
- Build other storage devices using the D Latch and logic devices we have at our disposal (i.e., in our *library*)
- What is the methodology behind design of sequential logic circuits
  - Finite State Machines to Truth Tables to Circuit
- Combine sequential and combinational logic devices to "assemble" a simple processor!

10

10

## Latches and Flip-Flops and Clock

- Latch: basic circuit for storage
  - Operate on changes in Level (i.e., 1 or 0)
  - D-Latch can store 1 bit
- Flip-flop:
  - Sequential circuits take input from output of storage
  - Latches that work on change of level can lead to unstable sequential circuits
    - As level changes the outputs change --- inputs change!
  - Flip-Flop circuits designed to operate properly when they are part of a sequential circuit
    - Modify D Latch to get a D Flip Flop (DFF)
    - Flip Flop changes state at the 'instant' that the level changes
- Clock:
  - Need to coordinate and synchronize when states change…
  - Use Clock to enable or disable the devices in a timed manner

11

11

## Our Latches

- We have designed two devices capable of storage

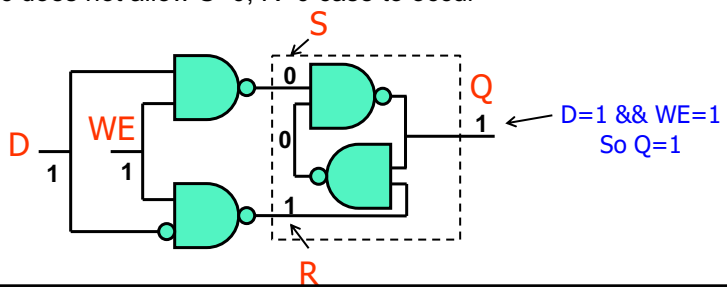
- RS Latch
  - Cross coupled NAND gates
  - Two inputs S (set) and R (reset) and Two outputs Q and Q'
- D Latch
  - Built from RS Latch
  - Two inputs D (value to be set), WE (write enable) and two outputs Q and Q'

12

12

## Recall the Gated D-Latch: Our 1-bit storage element

- Add logic to an R-S latch
  - Create a more convenient interface, prevent S=0 && R=0
- Two inputs: D (data) and WE (write enable)
  - When WE = 1, latch is set to value of D
    - S = NOT(D), R = D
  - When WE = 0, latch continues to hold previous value
    - S = R = 1 (hold condition for SR latch)
  - Extra logic does not allow S=0, R=0 case to occur

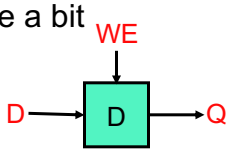



13

13

## Next… Storage Devices

- we now have a device ( D-Latch) that can store a bit
  - Abstract the device: input D, WE; output/storage Q



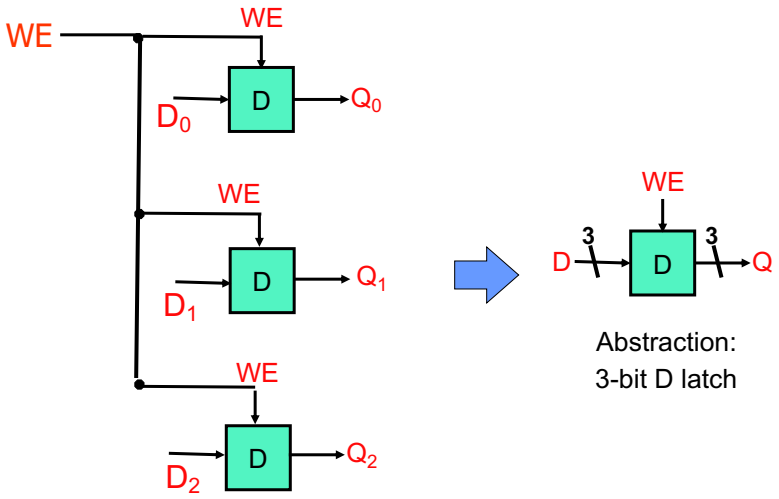

- Use this to build 'real' storage devices….
- Temporary storage in a computer…Register
  - Where are variables stored before being sent to the arithmetic unit for operations on them?
  - Can we build an n-bit register using latches?
- What about "main" memory
  - Memory hierarchy ?

14

14

## Multi–Bit D-Latch: Register ?

- A collection of D-latches, controlled by a common WE
- When WE=1, 3-bit value D is written to the outputs

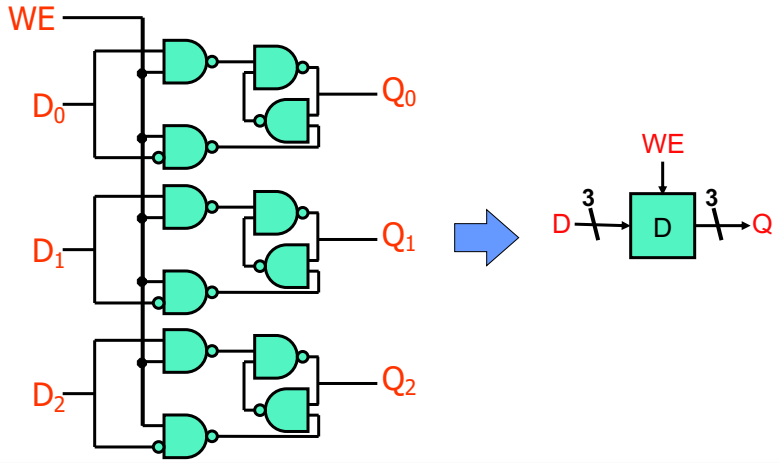



15

15

## Multi–Bit D-Latch – Register: Inside the latches

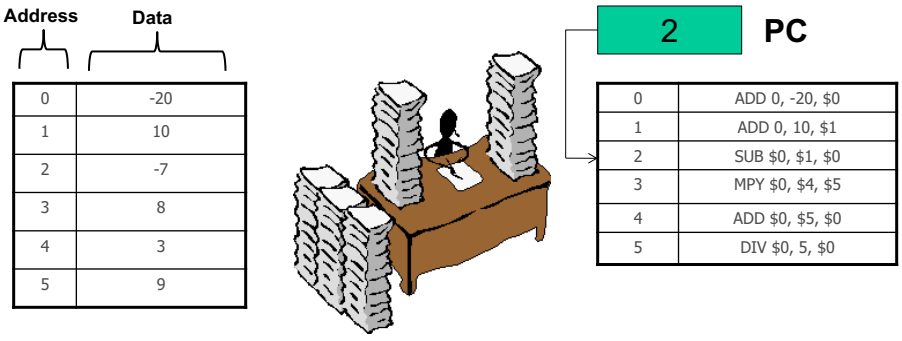
- A collection of D-latches, controlled by a common WE
- When WE=1, n-bit value D is written to the outputs




16

16

## Recall: A Basic Model of a Computer




**Memory** **CPU** **Instructions**

**Essential Part of Computer!**

**Basic Components: Address: Looks up data**
**Note: both are in binary**

17

17

## Memory

- We know how to store m-bit number in a register
- How about many m-bit numbers ?
  - Bank of registers?
- How to fetch a specific m-bit number?
  - addressing

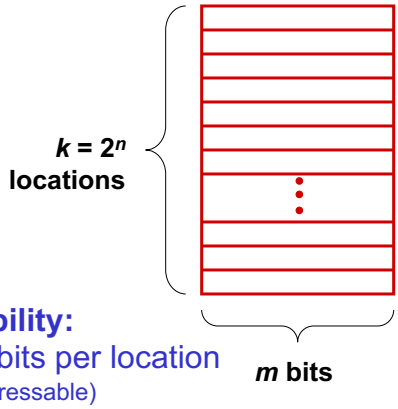
18

18

## Memory

•Now that we know how to store bits, we can build a memory – a logical *k by m* array of stored bits

**Address Space:**
number of locations
(usually a power of 2)




**Addressability:**
number of bits per location
(e.g., byte-addressable)

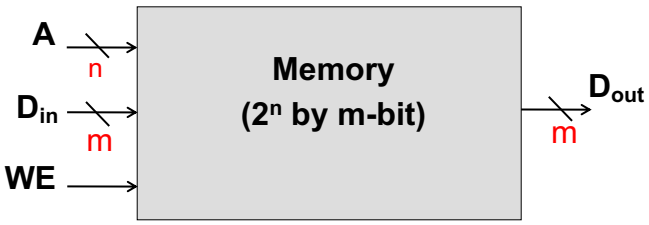
19

19

## Memory Interface

❖There are two basic operations on a memory
- Selecting one of the memory locations to read from
- Selecting one of the memory locations to write to

❖Interface signals
- A: n-bit address lines to select/specify a location
- $D_{out}$ : Contents of selected location during read (m bits)
- $D_{in}$ : Value to be stored during write (m bits)
- WE : If WE = 1 then write operation, WE = 0, read operation

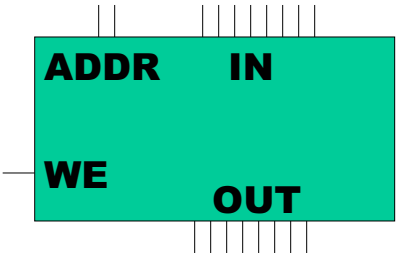



20

20

## Memory

- Looking from the outside, what do we need?




21

21

## Memory

A large number of addressable fixed size locations

- Address Space

n bits allow the addressing of $2^n$ memory locations.

- Example: 24 bits can address $2^{24}$ = 16,777,216 locations (i.e. 16M locations).
- If each location holds 1 byte (= 8 bits) then the memory is 16MB.
- If each location holds one word (32 bits = 4 bytes) then it is 64 MB.

22

22

## Memory

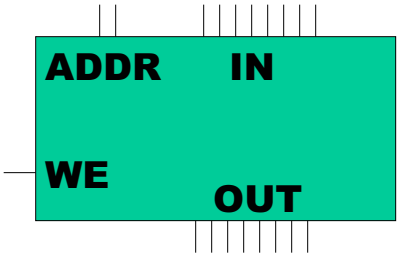
- Addressability
  - Computers are either byte or word addressable - i.e. each memory location holds either 8 bits (1 byte), or a full standard word for that computer (16 bits for the LC-3, more typically 32 bits, though now many machines use 64 bit words).
- Normally, a whole word is written and read at a time:
  - If the computer is word addressable, this is simply a single address location.
  - If the computer is byte addressable, and uses a multi-byte word, then the word address is conventionally either that of its most significant byte (big endian machines) or of its least significant byte (little endian machines).

23

23

## Memory

- READ operation: Given address A of N bits, fetch contents at that address
  - From $2^N$ locations we *select* one of them to be sent to the output
- WRITE: Given address A of N bits, write into exactly one of the $2^N$ locations.




24

24

## Devices to construct Memory ?

- To store a m-bit number use a ”register” (m-bit D-latch)
  - To store $2^N$ of these m-bit numbers, use $2^N$ m-bit latches

- For READ: what is the device that can send one out of $2^N$ inputs (inputs are in the $2^N$ latches) ?

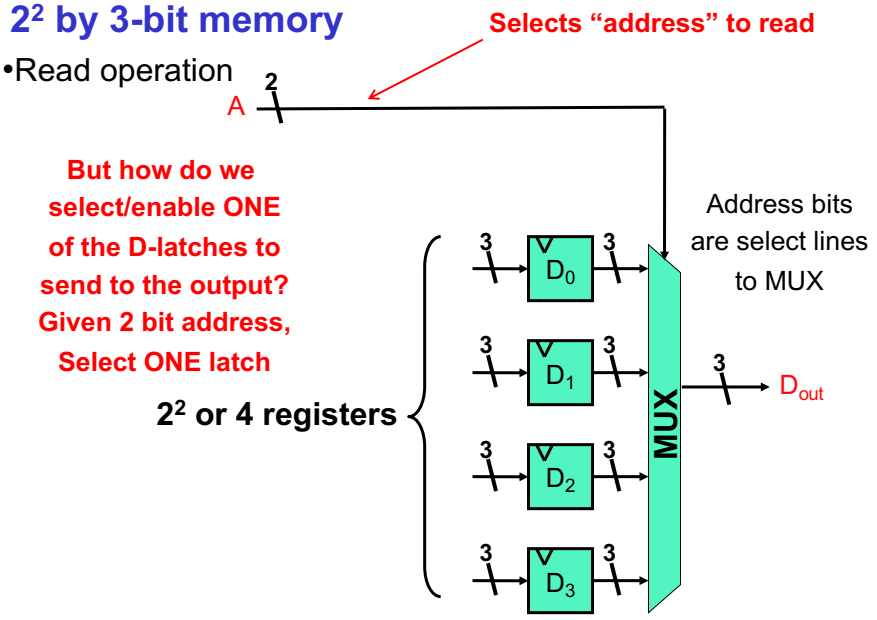
- For WRITE: what is the device that can enable exactly one Write Enable (WE) from the $2^N$ D-latches ?

25

25

## $2^2$ by 3-bit memory

- Read operation



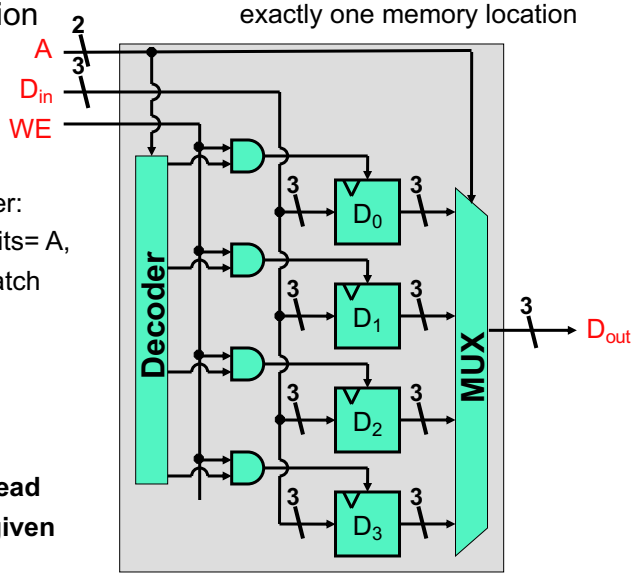

26

26

## $2^2$ by 3-bit memory

•Write operation

How do we enable write into exactly one memory location




Use 2-4 Decoder:
Input address bits= A,
exactly one D-latch
has WE=1

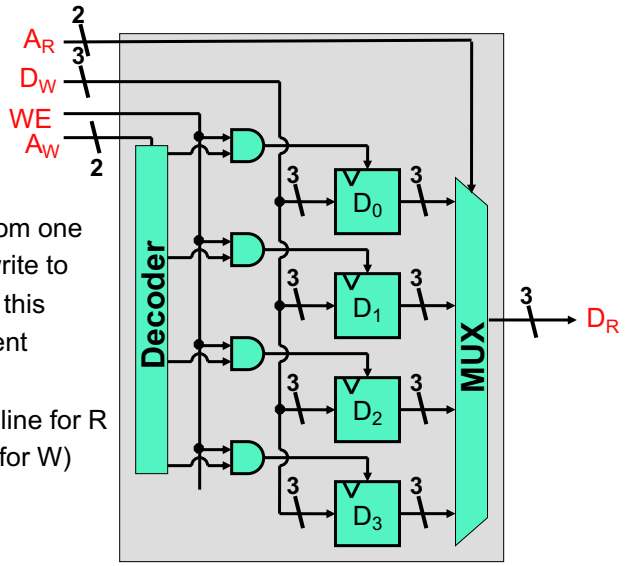
***Limitation:***
**You can only read or write at any given time**

27

27

## $2^2$ by 3-bit memory - Multiple "Ports"

•Independent Read/Write



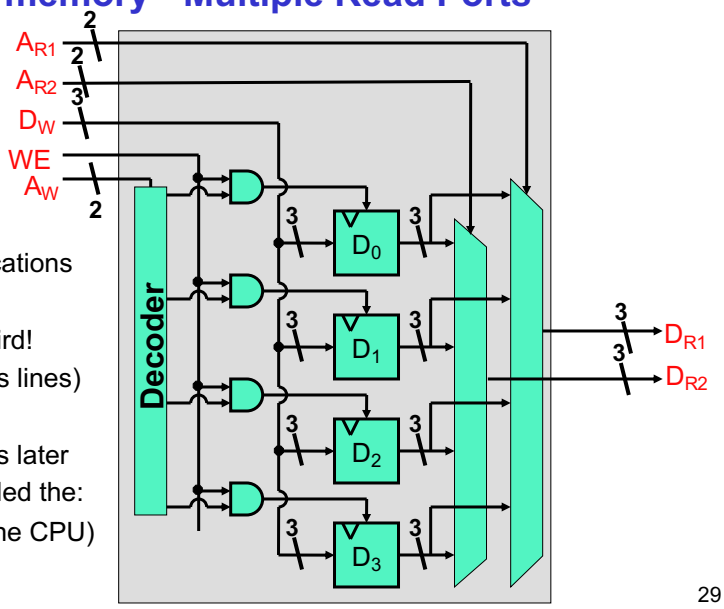

You can read from one address and write to another with this arrangement

(notice 1 address line for R
1 address line for W)

28

28

## $2^2$ by 3-bit memory - Multiple Read Ports



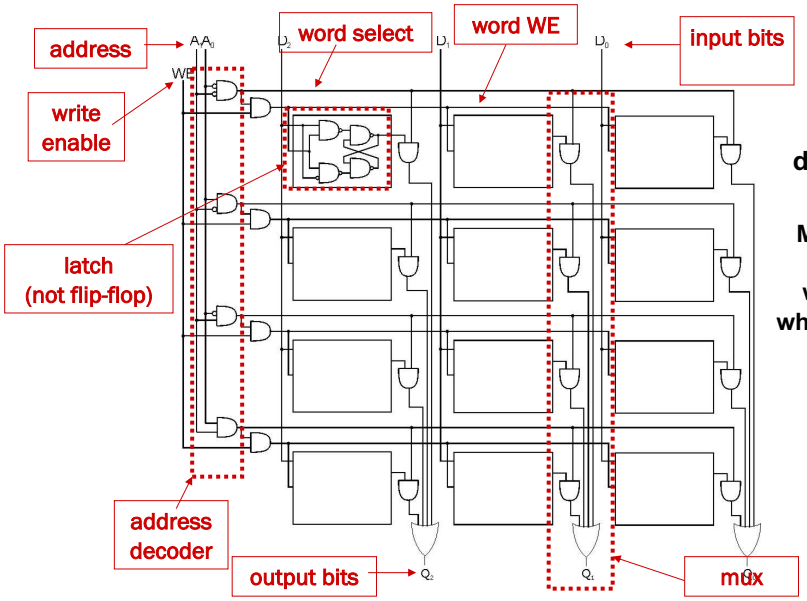

Read from 2 locations
At once,
Write to a third!
(notice 3 address lines)

(We will use this later
In something called the:
"register file" for the CPU)

29

29

## Inside an Efficient $2^2$ by 3-bit Memory - Single Port




**What is different?**

***D-latch***

**Makes this memory writeable when clock is HIGH,**

30

## More Memory Details

- This is still not the way actual memory is implemented
  - Real memory: fewer transistors, denser, relies on analog properties
- But the logical structure is similar
  - Address decoder
  - Word select line, word write enable
  - Bit line
- Two basic kinds of RAM (Random Access Memory)
- Static RAM (SRAM) - 6 transistors per bit
  - Fast, maintains data as long as power applied
- Dynamic RAM (DRAM) - 1 transistor per bit
  - Denser but slower, relies on "capacitance" to store data, needs constant "refreshing" of data to hold charge on capacitor

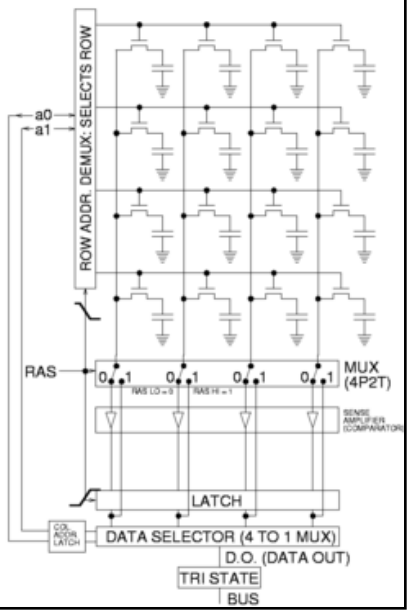
*Also, non-volatile memories: ROM, PROM, flash, …*

31

31

## Dynamic RAM

- Information stored as charge on capacitors.
- Capacitors leak so values have to be 'refreshed' continually
- As memory chips get larger, access times tend to increase. The processor spends more time waiting for data.
  - ➢ This is a major issue limiting computer systems performance




32

## Speed mismatch: Example

- Intel Core i5 – Processor
  - Clock rates approx 2.5GHz, Clock period approx 0.4 ns
- DDR2-667 PC2-5300 SO-DIMM – 2 GB Memory
  - Can deliver at most 1 64-bit word every 1.5 ns
- Mismatch between processor speed and memory speed

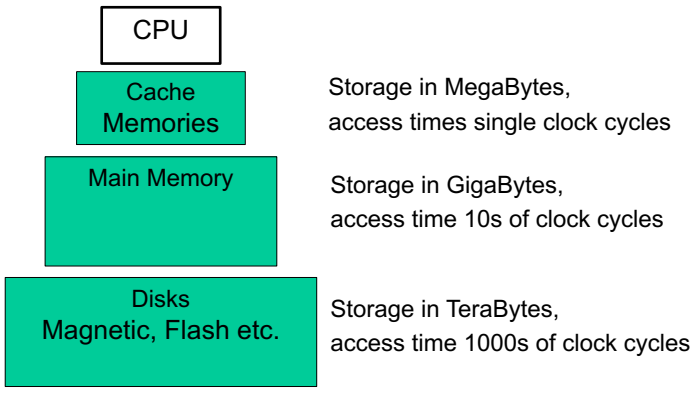


33

33

## Memory Hierarchy

- Modern computers try to mitigate memory delays by exploiting locality of reference through caches.
- Smaller, faster memory stores are placed closer to the CPU and bulk transfers from slower memory are used




34

34

## Memory Hierarchy

•Will return to this at the end of the course….!

35

35

## Are we ready to design sequential circuits and finite state machines ?

•Is something missing ?

•When do states change in a machine ?

•Do we let states change at arbitrary times ?

•What do you think happens in a computer ?

36

36

## Clocked Flip-Flops/Circuits

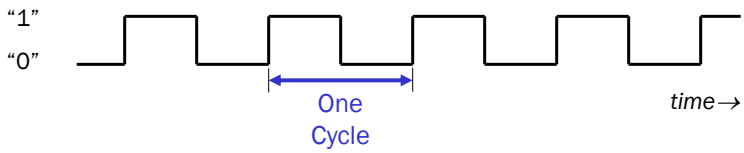
- Subsystem in a computer consists of a large number of combinational and sequential devices
  - Each sequential device is like latch which is in one of two states
  - As machine executes its cycle, the states of all sequential devices change with time
- To control large collection of devices in an orderly (synchronized) fashion, machine maintains a clock
  - Requires all devices to change their states at the same time
  - Clock generates sequence of pulses
  - Much easier to design, debug, implement, and test
- How do we change latches so that they allow change in state synchronized with the clock ?
- Sequential logic circuits require a means by which events can be sequenced…..clock!

37

37

## Introducing - The Clock!

- A clock controls when stored values are "updated"
  - Electrical waveform – sends pulses through a circuit
  - Changes values at a periodic rate

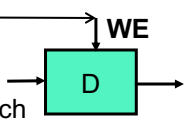



- The clock will act as the 'heartbeat' of our system
  - The number of cycles per second is the clock frequency measured in cycles per second or Hertz (Hz)
  - The clock period refers to the duration of one clock cycle. The period and frequency are inversely related.
    - Typical clock frequency: 2.5GHz = 2.5 x$10^9$ Hz
    - So corresponding clock period = 1/(2.5 x $10^9$) = .4x$10^{-9}$ sec
      - That would be: 0.4 nanoseconds

38

38

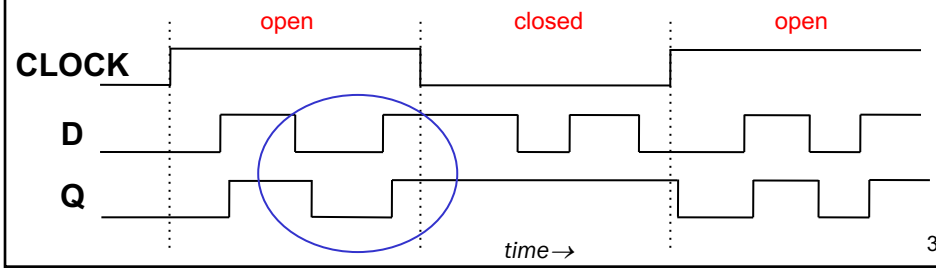
## Attaching Clock to D-Latch




- Attach CLOCK to the WE on D-Latch
- We create "windows" of time that we can store data into latch
  - When the CLOCK is "HIGH" – D-latch is open
  - When the CLOCK is "LOW" – D-latch is closed
- D=Q while CLOCK is High
- We have to prepare what we wish to store, right before latch closes

*Oops…changes in input while clock=High cascade to output Q*

*- Q can change multiple times during clock cycle: not synchronized with clock*

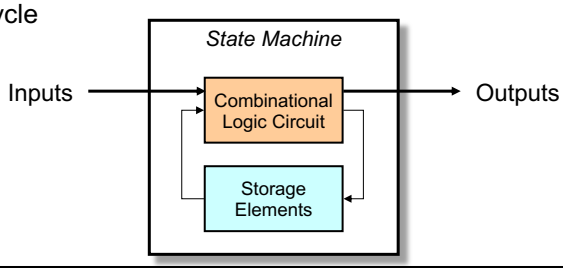



39

39

## Why Clocked D-latches may not work

- Sequential circuit: the next state (and output) depend on values of current state and input
- If input to D-latch changes while clock is High, then next state (and output) can change during this single clock cycle
  - state we want to store as next state could be overwritten
- we want to force changes (in state and input) to be synchronized to the clock
  - allow input/states to be read and output/next state to change ONCE per clock cycle




40

40

## Enter the Flip Flop….

- Flip flops are edge triggered devices
  - They capture the input and change state when the clock changes level
    - Positive edge triggered: when clock goes from 0 to 1
    - Negative edge triggered: when clock goes from 1 to 0

- We refer to the D flip flop as an edge-triggered device.
  - D=Q ONLY when WE changes from 0 to 1

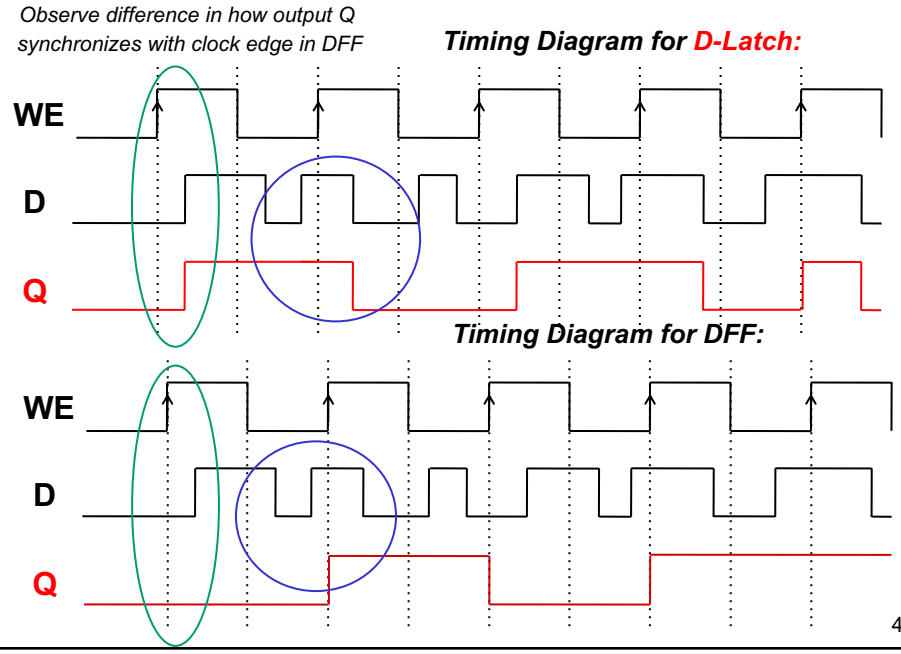
- This differs from D latch, which is: level-triggered
  - D=Q anytime WE equals 1

41

41

## D Flip-Flop vs. D-Latch – Timing Diagrams

*Observe difference in how output Q synchronizes with clock edge in DFF*

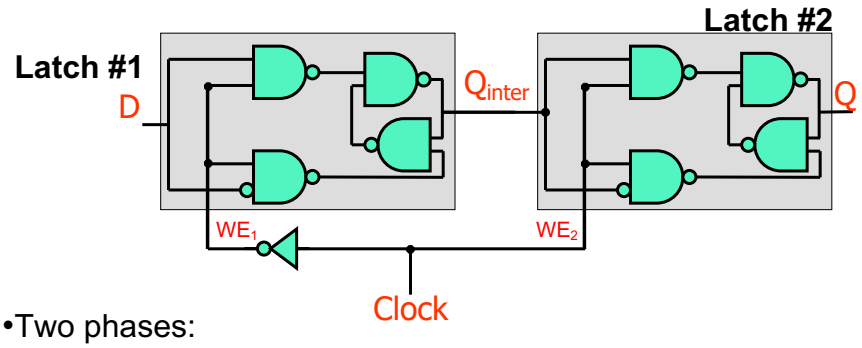



42

42

## D Flip-Flop

- D Flip-Flop is a pair of D latches
  - Isolates *next* state from *current* state
  - *Output of DFF read when clock is high, Next state stored end of cycle*

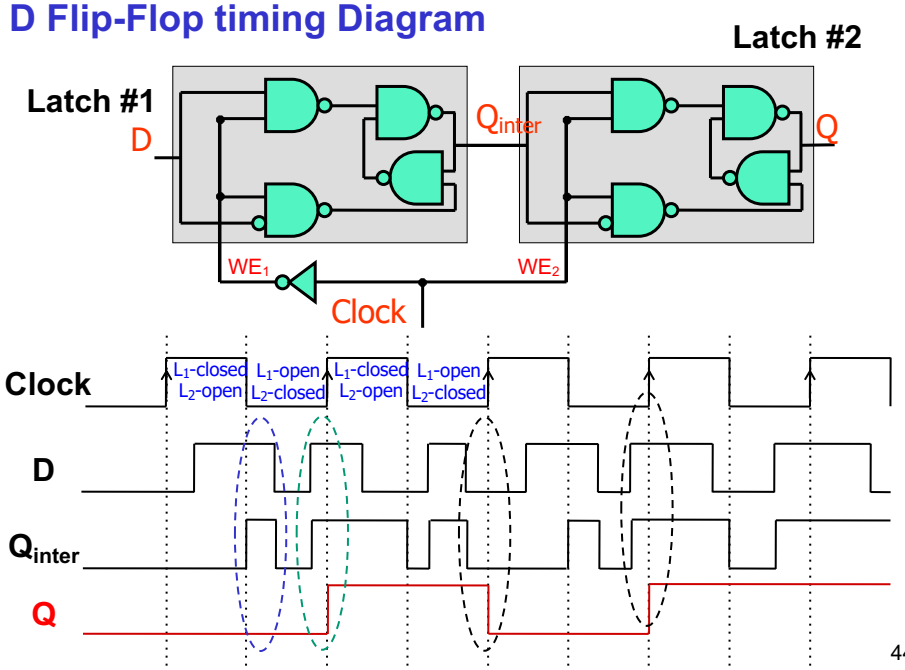



- Two phases:
  - Clock = 1: $WE_1$ =0: Latch #1 closed, $WE_2$ =1: Latch #2 open
  - Clock = 0: $WE_1$ =1: Latch #1 open, $WE_2$ =0: Latch #2 closed

43

43

## D Flip-Flop timing Diagram




44

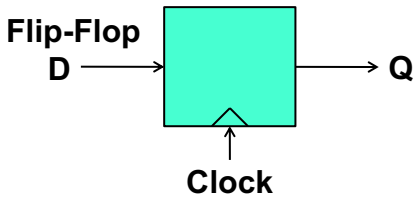
44

## D Flip-Flop

- We can think of the D Flip-Flop as a 1 bit storage container with an input, D, and an output, Q and connected to a clock input
- A set of D flip-flops can be grouped together with common Clock and WE inputs to form a register -- replace D-latch with D flip flop

Truth table for DFF:
X = don't care (i.e, 0 or 1)

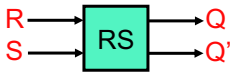
| D | Clk | Q |
|---|---|---|
| 0 | 0 → 1 | 0 |
| 1 | 0 → 1 | 1 |
| X | 0 | Last Q |
| X | 1 | Last Q |



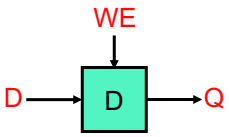

45

45

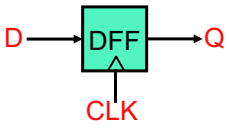
## Our basic "Storage" Devices




RS Latch – Stores 1 Bit, Level-Triggered
-1 "forbidden" input: S=0, R=0
-Holds Data when RS=11




D-Latch – Stores 1 Bit, Level-Triggered
-No "forbidden" inputs (fixes RS Latch)
-D=Q when WE=1
-Holds Data when WE=0



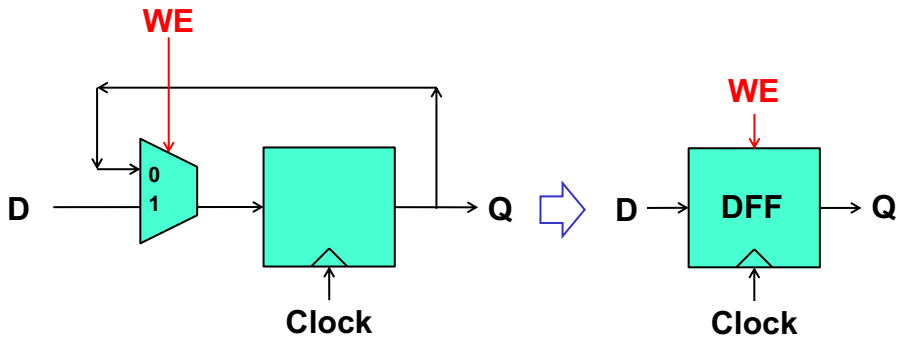

D-Flip-Flop – Stores 1 Bit, Edge-Triggered
-No "forbidden" inputs
-D=Q when WE (CLK) transitions from 0 to 1
-Holds Data for WE=1 or WE=0
-Except when WE transitions from 0 to 1

46

46

## One more tweak…
## D Flip-Flop with Additional Write Enable

- From previous slides, we attached clock to WE of the D-flip-flop
- Now, we add another WE line to the flip flop
  - Just holds onto data already stored in DFF
- Give it the ability to "ignore" the clock!

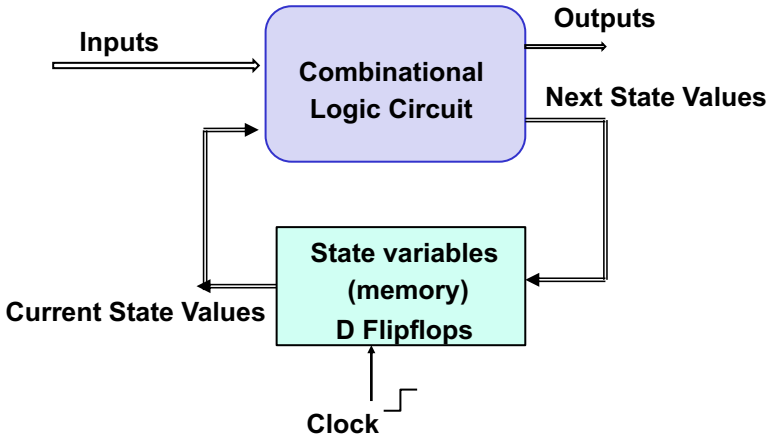



Flip-Flop w/WE

47

47

## Schematic of a Sequential Circuit




48

48

## Next: Design methodology for sequential logic circuits .. Finite State Machines

- We have storage devices and method for synchronization
  - Flip flops and Clock
- How do we design a clocked circuit to solve our problems ?
  - Example: Is there a methodology behind designing a circuit to implement a Counter ?
- Provide procedure for designing Sequential circuits (to implement Finite State Machine)
  - Storage Device to store state: D flip flop
  - Logic to implement next state: combinational gates/devices
  - How to derive the logic: truth tables

49

49

## States in a FSM

- The concept of state
  - the state of a system is a "snapshot" of all relevant elements at a moment in time.
  - a given system will often have only a finite number of possible states.
  - For many systems, we can define the rule which determines under what conditions a system can move from one state to another.
    - So when do they change states ?
      - Synchronized to clock (edge triggered)
- Determining the 'states'
  - Problem statement determines what information needs to be stored (state is a summary)
  - How many states does the machine need ?

50

50

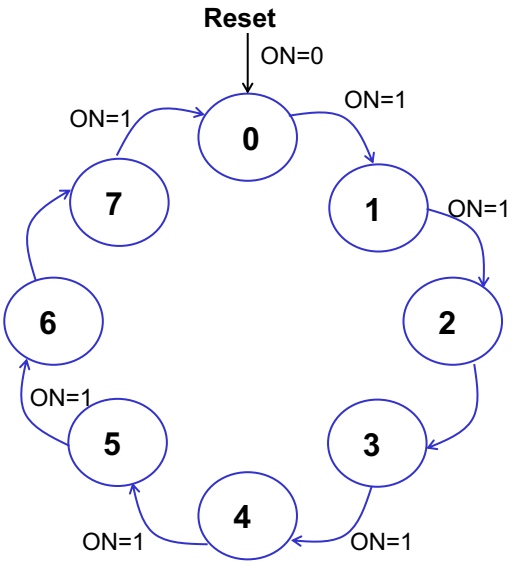
## Designing and implementing a FSM

1. First draw the state diagram
   - Encode each state in binary using N bits
   - These N bits correspond to N "state variables" that need to be stored. Call them $S_{N-1}\, S_{N-2} \ldots S_1 S_0$
   - State diagram will show transitions from state to state based on value of inputs
2. Next, derive the truth table (from state diagram)
   - "inputs" in truth table are N current state variables and the inputs
   - "outputs" are the values of the state variables in the next state and the output at each state -- common notation is S' but confusion with complement operator, so let's use S*
3. From truth table, derive combinational circuit (boolean function) for each of the next state values
   - State variables are stored in your N storage elements

51

51

## Finite State Machine Representation of Counter: Counter to count from 0 to 7 (while switch is ON)




Bubbles represent all possible "states" for the machine

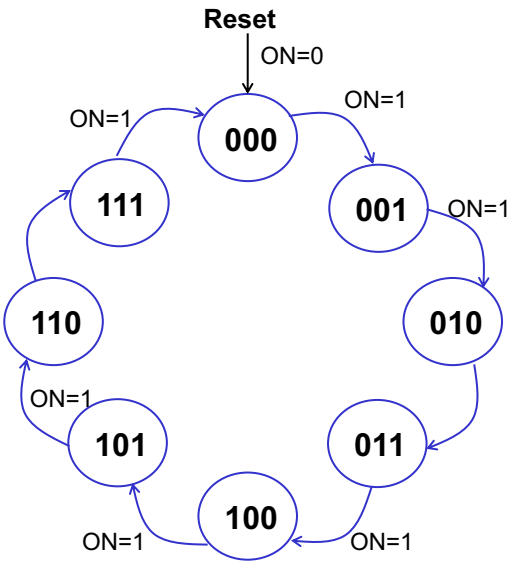
Arrows show movement from one state to the next while switch ON=1.
If ON=0, FSM goes to state 0

*Transitions occur at pulse of the clock*

52

52

## Labelling States in Binary




We have 8 states, therefore we need $\log_2 8 = 3$ bits to encode states in binary:
from 000 to 111

Each bit is called a state variable & Stored in one D flip flop
we need 3 Flip flops to store 3 bits $S_2\, S_1\, S_0$

53

53

## Storage

- Each D flip flop stores one state bit.
- The number of storage elements (flip-flops) needed is determined by the number of states (and the representation of each state).
  - Each bit can be 0 or 1 = 2 states
  - N bits can represent $2^N$ states

- Example: If FSM has 12 states, then circuit needs $\log_2 12 = 4$ storage elements (i.e., flip flops).
  - Fewer the states, less hardware needed
    - *Concept of Minimization of States for a given FSM…*
      - *…in Foundations (where else!!)*

54

54

## Designing and implementing a FSM

1. First draw the state diagram
   - Encode each state in binary using N bits
   - These N bits correspond to N "state variables" that need to be stored. Call them $S_{N-1}\, S_{N-2} \ldots S_1 S_0$
   - State diagram will show transitions from state to state based on value of inputs
2. Next, derive the truth table (from state diagram)
   - "inputs" in truth table are N current state variables and the inputs
   - "outputs" are the values of the state variables in the next state S* and the output at each state (common notation is S' but confusion with complement operator, so let's use S*)
3. From truth table, derive combinational circuit (boolean function) for each of the next state values
   - State variables are stored in your N storage elements

55

55

## Truth Table Representation of Counter

| Input | Present State | | | | Next State | | |
|---|---|---|---|---|---|---|---|
| On | $S_2$ (t) | $S_1$ (t) | $S_0$ (t) | | $S_2$* (t+1) | $S_1$* (t+1) | $S_0$* (t+1) |
| | | | | | | | |
| 1 | 0 | 0 | 0 | | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | | 0 | 0 | 0 |
| 0 | X | X | X | | 0 | 0 | 0 |

56

56

## Designing and implementing a FSM

1. First draw the state diagram
   - Encode each state in binary using N bits
   - These N bits correspond to N "state variables" that need to be stored. Call them $S_{N-1}\, S_{N-2} \ldots S_1 S_0$
   - State diagram will show transitions from state to state based on value of inputs
2. Next, derive the truth table (from state diagram)
   - "inputs" in truth table are N current state variables and the inputs
   - "outputs" are the values of the state variables in the next state and the output at each state -- common notation is S' but confusion with complement operator, so let's use S*
3. From truth table, derive combinational circuit (boolean function) for each of the next state values
   - State variables are stored in your N storage elements

*This part is no different from combinational circuit design!*

57

57

## Boolean functions for values of next state

- For each state variable, derive function that determines next state for that variable:
- Note: $S_2$ (t+1) – value of state variable at time (t+1) is denoted as $S_2$*
- Current states: $S_2\ S_1\ S_0$
- Derivation here is only for case when On=1

- $S_2^* = On.(S_2'\, S_1 S_0 + S_2 S_1' S_0' + S_2 S_1' S_0 + S_2 S_1 S_0')$

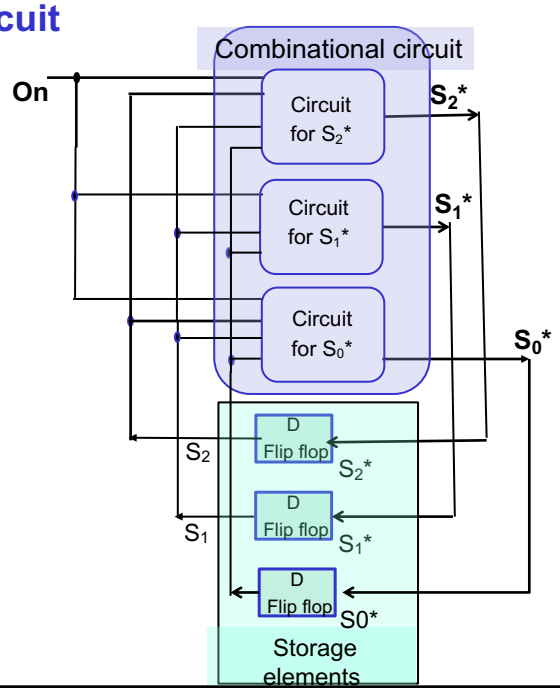
- $S_1^* = On.(S_2'\, S_1' S_0 + S_2' S_1 S_0' + S_2 S_1' S_0 + S_2 S_1 S_0')$

- $S_0^* = On.(S_2'\, S_1' S_0' + S_2' S_1 S_0' + S_2 S_1' S_0' + S_2 S_1 S_0')$

58

58

## Circuit




59

59

## Summary: Designing and implementing a FSM

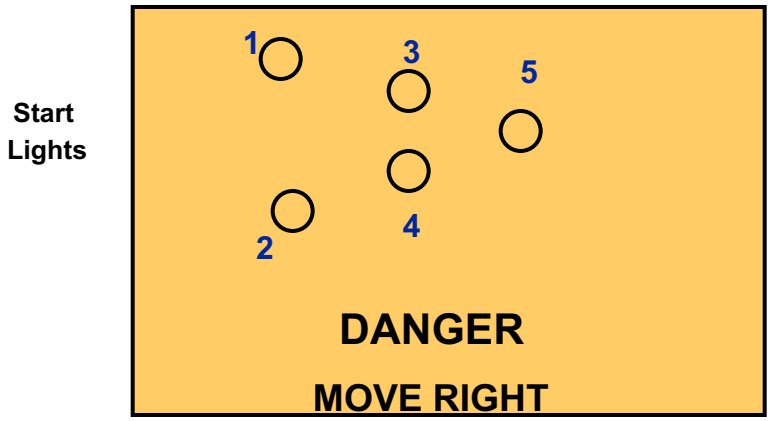
1. First draw the state diagram
   - Encode each state in binary using N bits
   - These N bits correspond to N "state variables" that need to be stored. Call them $S_{N-1}\, S_{N-2} \ldots S_1 S_0$
   - State diagram will show transitions from state to state based on value of inputs
2. Next, derive the truth table (from state diagram)
   - "inputs" in truth table are N current state variables and the inputs
   - "outputs" are the values of the state variables in the next state and the output at each state -- common notation is S' but confusion with complement operator, so let's use S*
3. From truth table, derive combinational circuit (boolean function) for each of the next state values
   - State variables are stored in your N storage elements

60

60

## Example 2: Blinking Traffic Sign: Start all lights off – lights bulbs 1,2,3,4,5 off

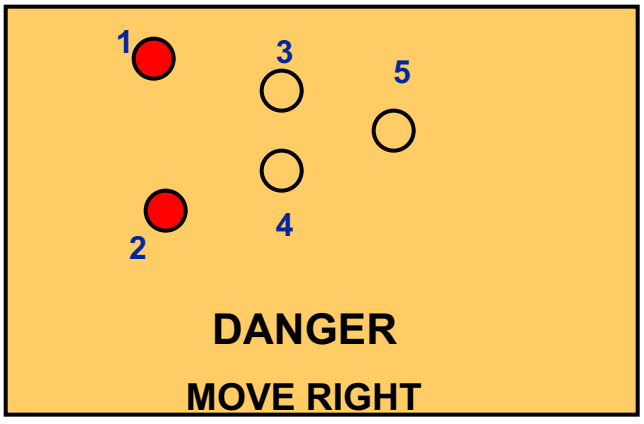
- (from textbook) Design control circuitry for a blinking traffic sign (to show "move right" message)




61

61

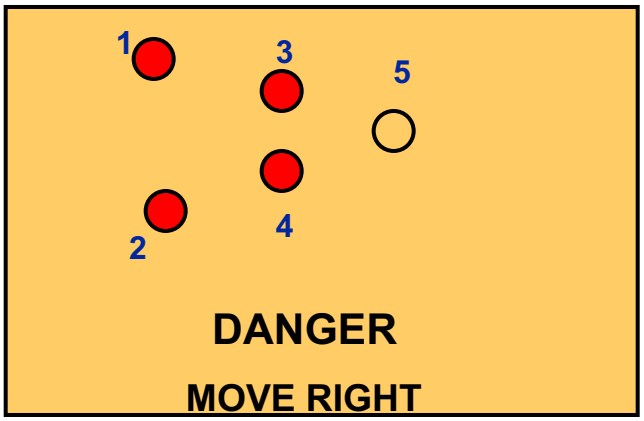
## Step2: switch 'on' 2 lights – bulbs 1,2




62

62

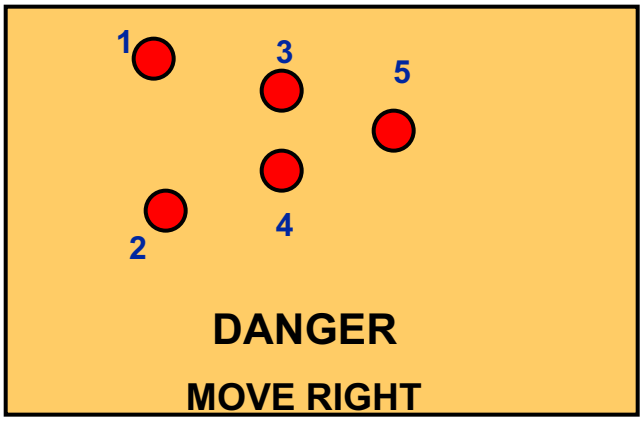
## Step 3: Switch ‘on’ 4 lights – bulbs 1,2,3,4




63

63

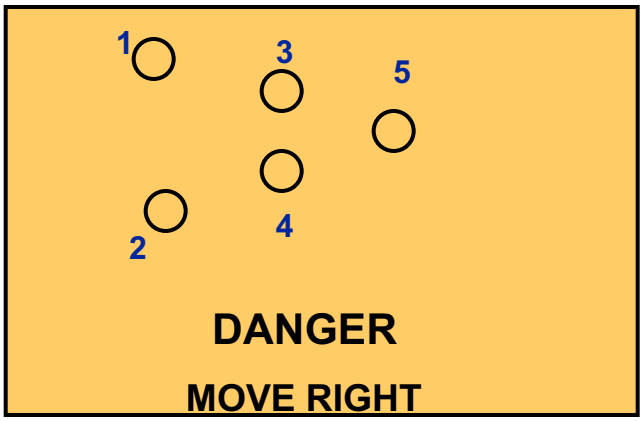
## Step 4: switch ‘on’ 5 lights – 1,2,3,4,5




64

64

## Step 5- switch all 'off' – lights bulbs 1,2,3,4,5 off And repeat the cycle




65

65

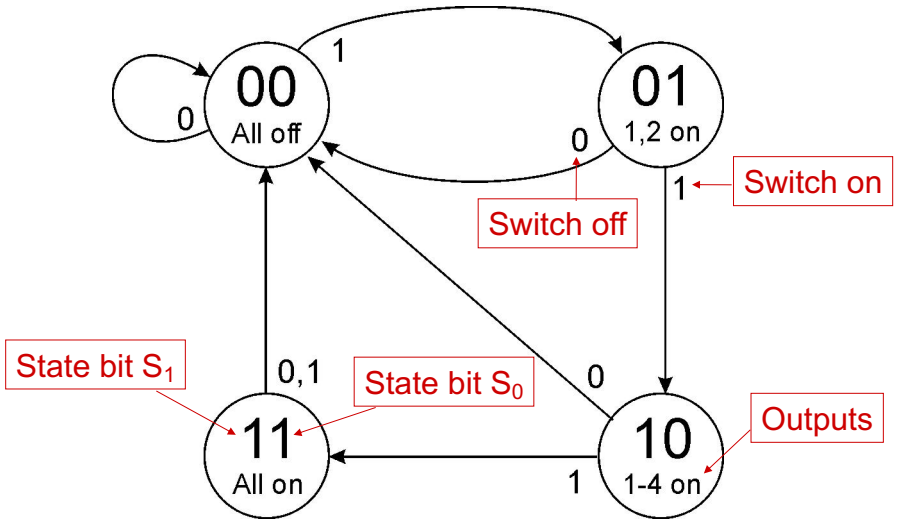
## Example: Traffic Sign

- A blinking traffic sign: How many lights/lightbulbs = 5
- How many states ?
- 4 states
  - No lights on
  - 1 & 2 on
  - 1, 2, 3, & 4 on
  - 1, 2, 3, 4, & 5 on
  - (repeat as long as switch is turned on)
- How many bits to represent the 4 states
- $S_1S_0$
  - With $S_1S_0$ values: 00, 01, 10, 11
- How many 'outputs' (to control the 5 lights) = 5 ?
- If the sign is switched off then all lights turn off

66

66

## Traffic Sign State Diagram




*Transition on each clock cycle.*

67

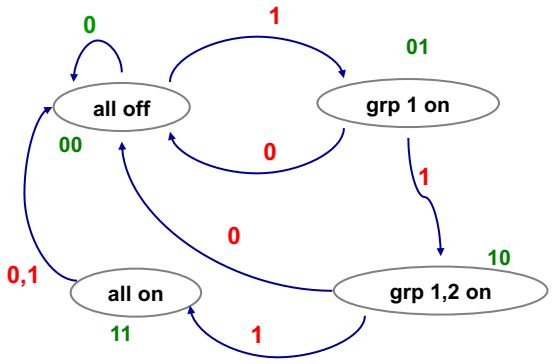
67

## Outputs

- Note we really have 3 groups of lights to be controlled = 3 control lines X,Y,Z
  - Group 1: Lights 1 and 2; controlled by Z
    - If Z=1 then Group 1 lights (1 and 2) are switched on
  - Group 2: lights 3 & 4; controlled by Y
  - Group 3: Light 5; controlled by X
- In this example, we associate each state with an output
  - Depending on the current state, we switch on specific groups of lights

68

68




69

69

- When is group 1 on?
  - in states 01, 10 and 11 - but only when the switch IN is on!
- Logic expressions for X,Y,Z
  - Depends on $S_0$ and $S_1$ and Input is on
    - If Input is off then X,Y,Z are al 0
- can you come up with a logic expression for next state values of $S_0$ and $S_1$?
  - Depends on current values of $S_0$ and $S_1$ and Input is on
    - Input off then both bits are set to 0 since next state is 00
  - Next state value of S0 denoted S'0 = 1 if current state is 00 or current state 10 and In=1
- When do we switch to the next state?
  - the two bits of S[1:0] are updated at every clock cycle
  - we have to make sure that the new state does not propagate to the combinational circuit input until the next clock cycle.

70

70

## Traffic Sign Truth Tables

Outputs
(depend only on state: $S_1S_0$)

Lights 1 and 2
Lights 3 and 4
Light 5

| $S_1$ | $S_0$ | 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: $S_1'S_0'$
(depend on state and input)

Switch

| In | $S_1$ | $S_0$ | $S_1'$ | $S_0'$ |
|---|---|---|---|---|
| 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 |

Whenever In=0, next state is 00.

71

71

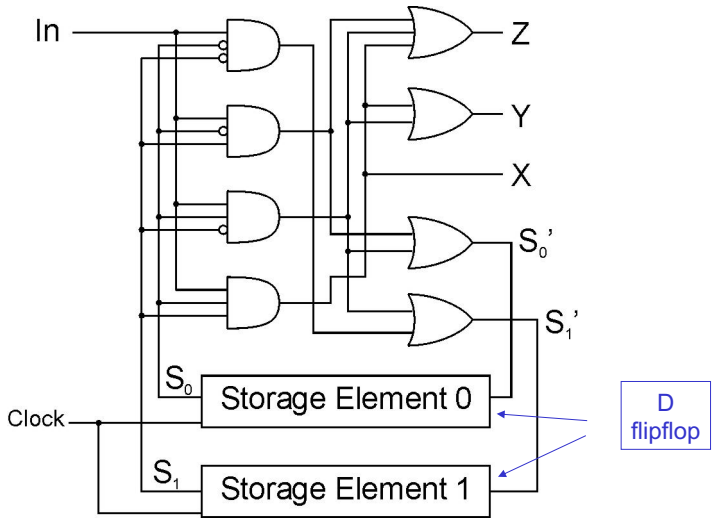
## Boolean functions for light control bits

- from truth table, consider all rows where outputs =1
- $Z = ((\text{NOT } S_1)S_0 + S_1(\text{NOT } S_0) + S_1S_0).\text{In}$
- $Y = (S_1S_0 + S_1(\text{NOT } S_0)).\text{In}$
- $X = (S_1S_0).\text{In}$

72

72

## Traffic Sign Logic




73

73

## Exercise: Design a sequential circuit

- Counter that counts from 0 to 6
  - Resets to 0 after 6
  - Keeps going as long as "input" = 1
  - Ignore this for now

- Finite state diagram
- Truth table
- Logic functions for next state

74

74

## Summary of Digital Logic

- Combinational logic
  - Basic gates
  - Combinational devices
- Sequential logic
  - Storage element…Flip flop
  - Theory behind design of finite state machines
    - They act like controllers of a circuit
- Sequential logic devices
  - Memory, ROM, RAM, Registers

So, what is the purpose of all this ?.....

75

75

## The Agenda

- Take the elements that we have encountered so far
  - Combinational Elements
    - Gates, Adders, Muxes, decoders
  - Storage Elements
    - Flip flops, registers, memories
- And use them to build a circuit that can perform a sequence of arithmetic operations.
- In essence, we will build a very simple CPU

76

76

## From Logic to Processor Design

- Combinational Logic
  - Decoders -- convert instructions into control signals
  - Multiplexers -- select inputs and outputs
  - ALU (Arithmetic and Logic Unit) -- operations on data
- Sequential Logic
  - State machine -- coordinate control signals and data movement
  - Registers and latches -- storage elements

77

77

## Next: Putting it all together

- The goal:
  - Turn a theoretical device - Turing's Universal Computational Machine - into an actual computer ...
  - … interacting with data and instructions from the outside world, and producing output data.
- Smart building blocks:
  - We have at our disposal a powerful collection of combinational and sequential logic devices.
- Now we need a master plan ...
  - We need to start with defining the model of a computer architecture
    - **Von Neuman model**

78

78