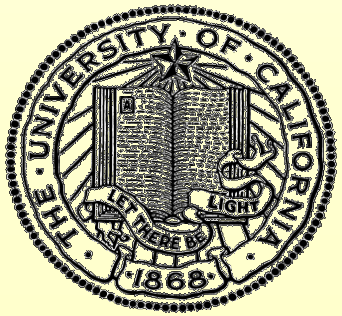
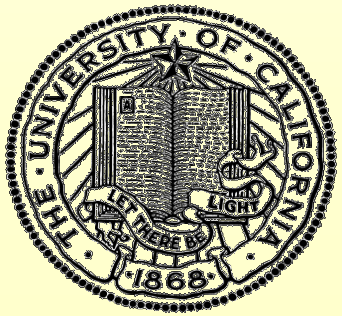
Digital Logic: Boolean Algebra and Gates (contd..)



HW: Please go through Participation activities on Zybooks before doing Challenge activities!



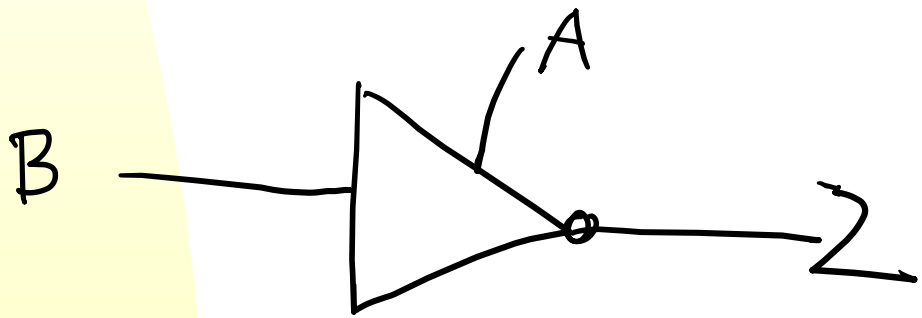
XOR gate as a "programmable "inverter (NOT gate)

A	B	Z (A ⊕ B)
0	0	0
0	1	1
1	0	1
1	1	0

When A=0, Z=B

When A=1, Z=$\overline{B}$

- Thus, we can "program" Z to either be inverse of B, or simply be equal to B, depending on the value of A

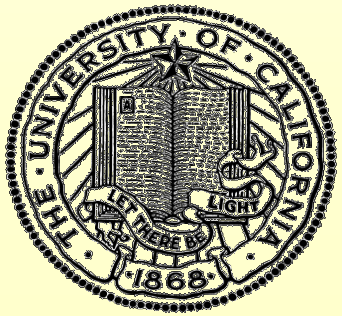


Above is an alternative symbol for XOR gate as a programmable inverter

But in general, we will stick to the normal XOR gate symbol

Sum of Products (SOP)

- How do you get from a truth table to a logic expression?
- Sum of products is standard way of synthesizing simple circuits
- Procedure:
  1. Find the rows with the '1' output
  2. Write the product-form expression for the inputs in that row (0=inverted, 1=normal)
  3. Combine the products in step 2 into a sum (OR the results of step 2)

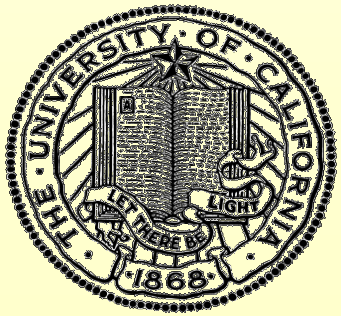


Sum of Products

- XOR Gate

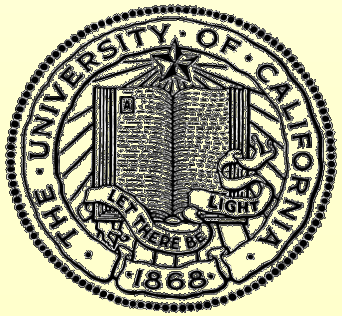
A	B	Y
0	0	
0	1	
1	0	
1	1	

1. Find the rows with the '1' output
2. Write the product-form expression for the inputs in that row (0=inverted, 1=normal)
3. Combine the products in step 2 into a sum (OR the results of step 2)



Product of Sums

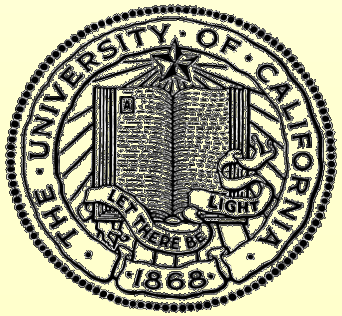
- Procedure:
  1. Find the rows with the '0' output
  2. Write the sum-form expression for the inputs in that row (0=normal, 1=inverted)
  3. Combine the sums in step 2 into a product (AND the results of step 2)
- Note: we treat 0 and 1 reverse than for SoP



Product of Sums

- XOR Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



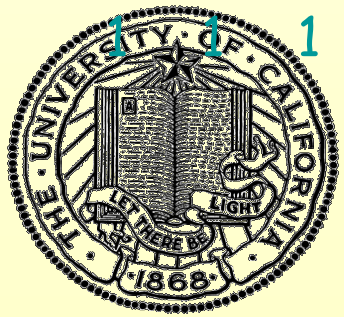
Examples of SoP and PoS

Find SoP

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Find Pos

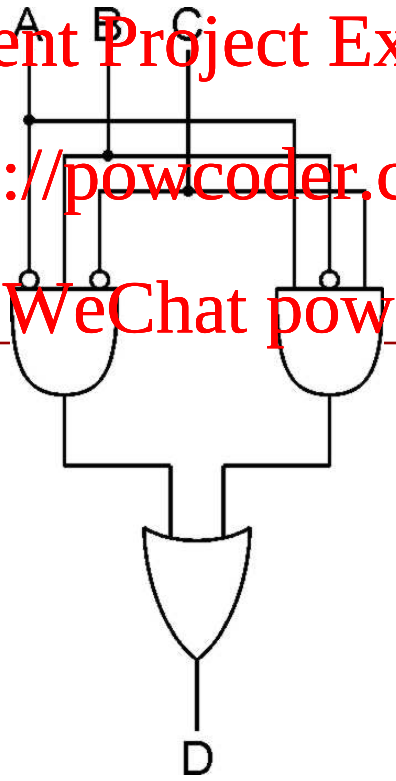
A	B	C	Z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



Logical Completeness

Can implement ANY truth table AND, OR, and NOT!

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

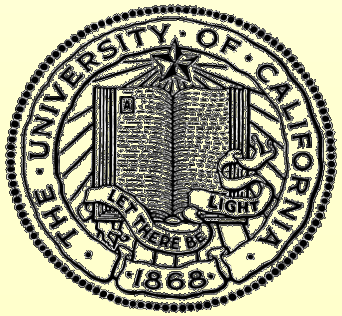


1. AND combinations that yield a "1" in the truth table.
2. OR the results of the AND gates.
3. Is not necessarily a minimal solution.

De Morgan's Laws

Two laws:

- A' + B' = (AB)'
- A' B' = (A+B)'

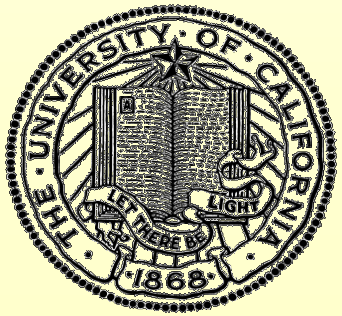


De Morgan's Laws

(A + B)' = A'B' conversely (AB)' = A' + B'

"Break the line, change the sign"

A	B	$\overline{A+B}$	$\overline{AB}$	$\bar{A}+\bar{B}$	$\bar{A}\cdot\bar{B}$
0	0	1	1	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	0	0	0	0

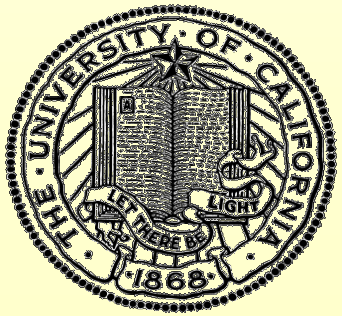


De Morgan's Laws

$(A + B)' = A'B'$ conversely $(AB)' = A' + B'$

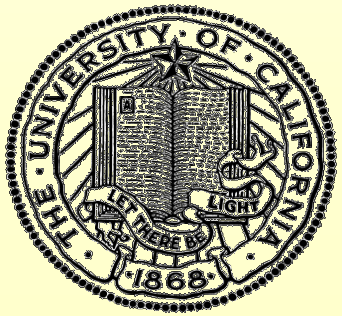
"Break the line, change the sign"

A	B	$\overline{A+B}$	$\overline{AB}$	$\overline{A}\,\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	0	1	1 0	1
1	0	0	1	0 1	1
1	1	1	0	0 0	0

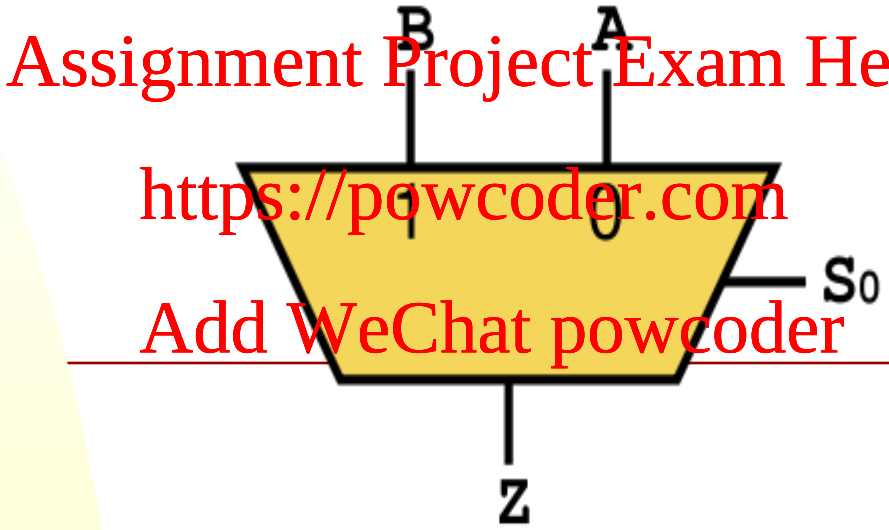


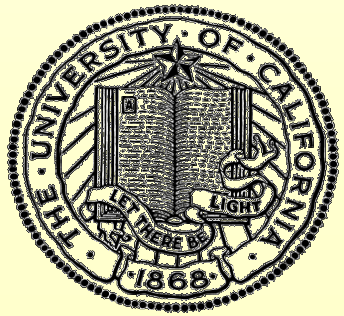
Even Simpler Logical Completeness

- Can implement ANY truth table NAND only!
  - Make NOT out of NAND?
  - Make OR out of NAND?
  - Make AND out of NAND?



Two-Way Multiplexer





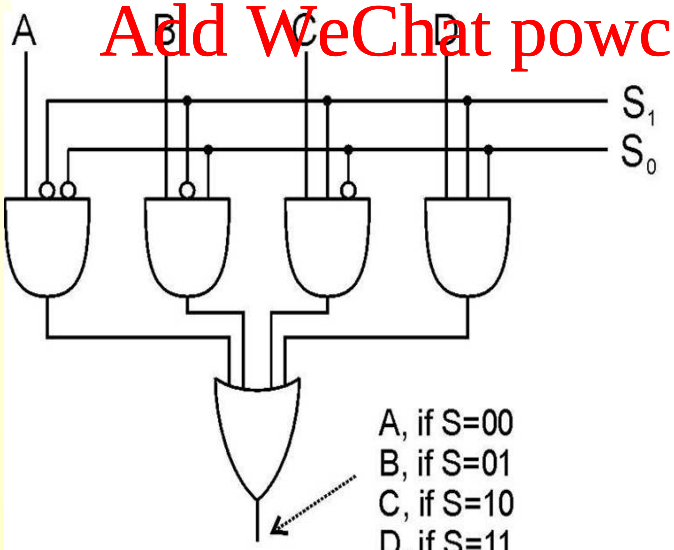
Two-Way Multiplexer

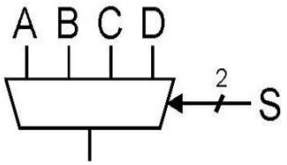
2-way multiplexer: the output is equal to one of the two inputs, based on a selector

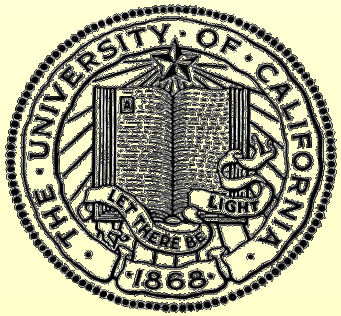
S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Four-Way Multiplexer

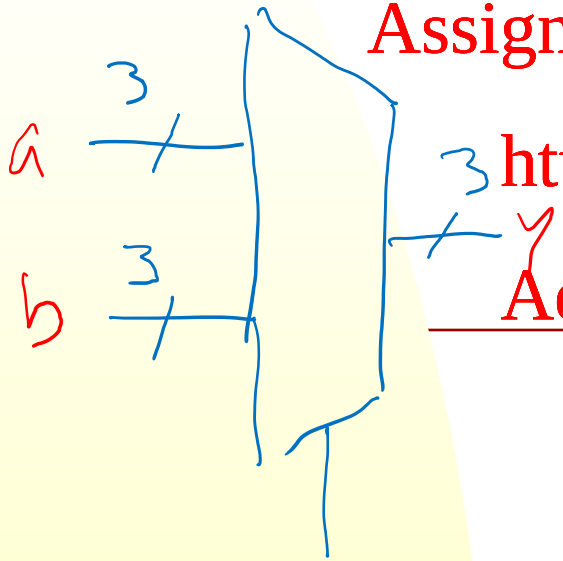
- *n*-bit selector and 2^n inputs, one output
  - output equals one of the inputs, depending on selector
- "Four-to-one mux"

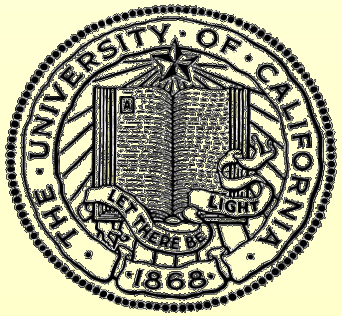






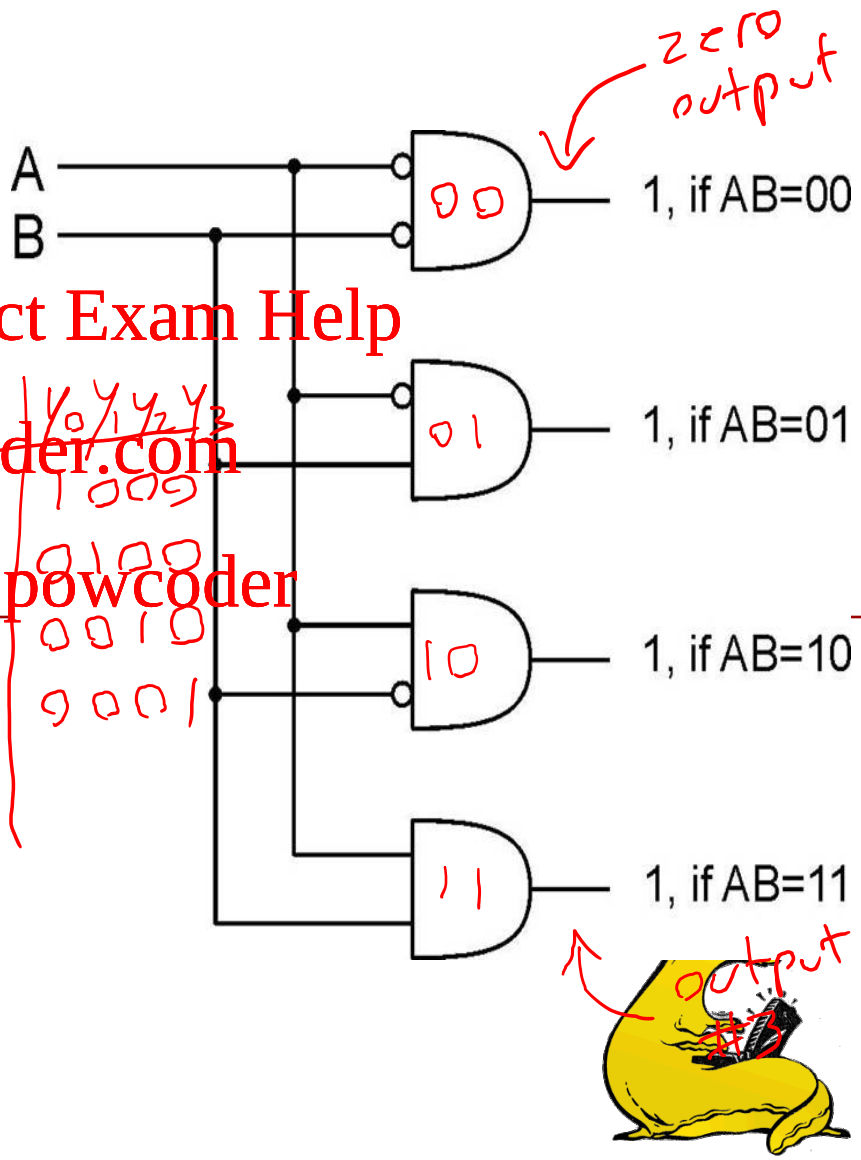
Multiple Bit Multiplexers (and Buses)





Two-to-Four Decoder

- *n* inputs, 2^n outputs
  - exactly one output is 1 for each possible input pattern
- Uses:
  - Convert memory or register address to a control line
  - Convert an opcode to one of n control lines
  - We will get to this in the MIPS material



Time for some...

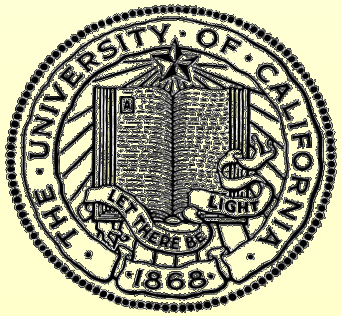
- We currently use decimal system in daily life (deci=10 digits,0-9)
- We know..

1+0=1

1+1=2;1+2=3;1+3=4…

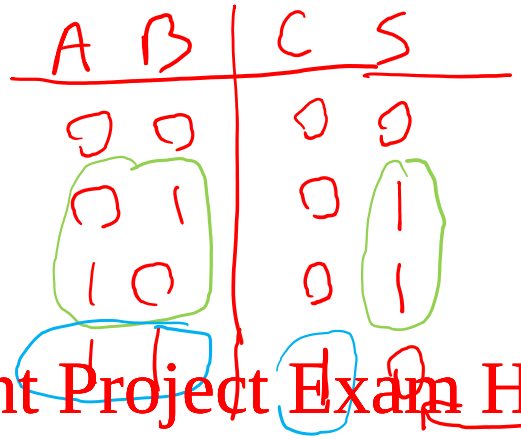
1+8=9;

- What is 1+9=??

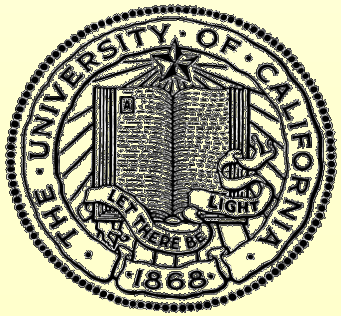


Binary Addition and Half-Adder

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 10

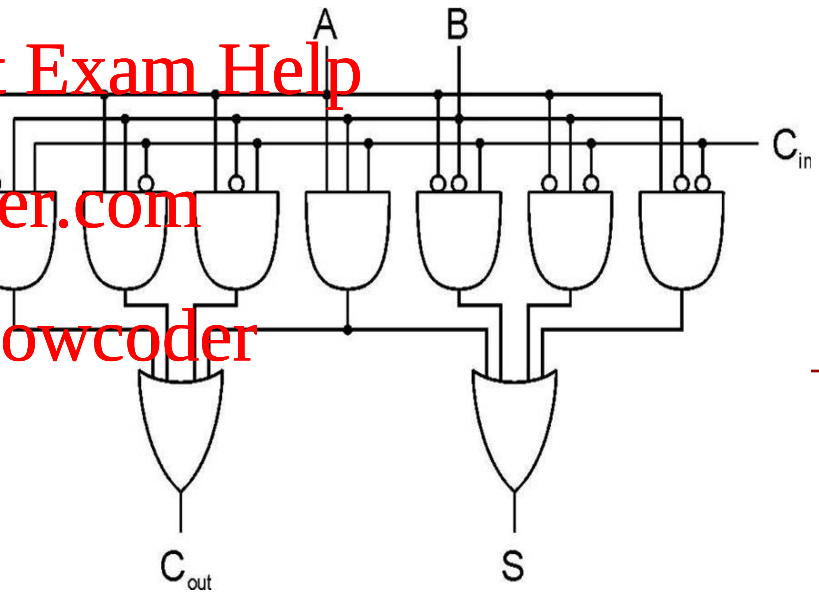


- A half-adder can add 2 bits and produces a sum and carry signal
  - Sum = A xor B
  - Carry = AB



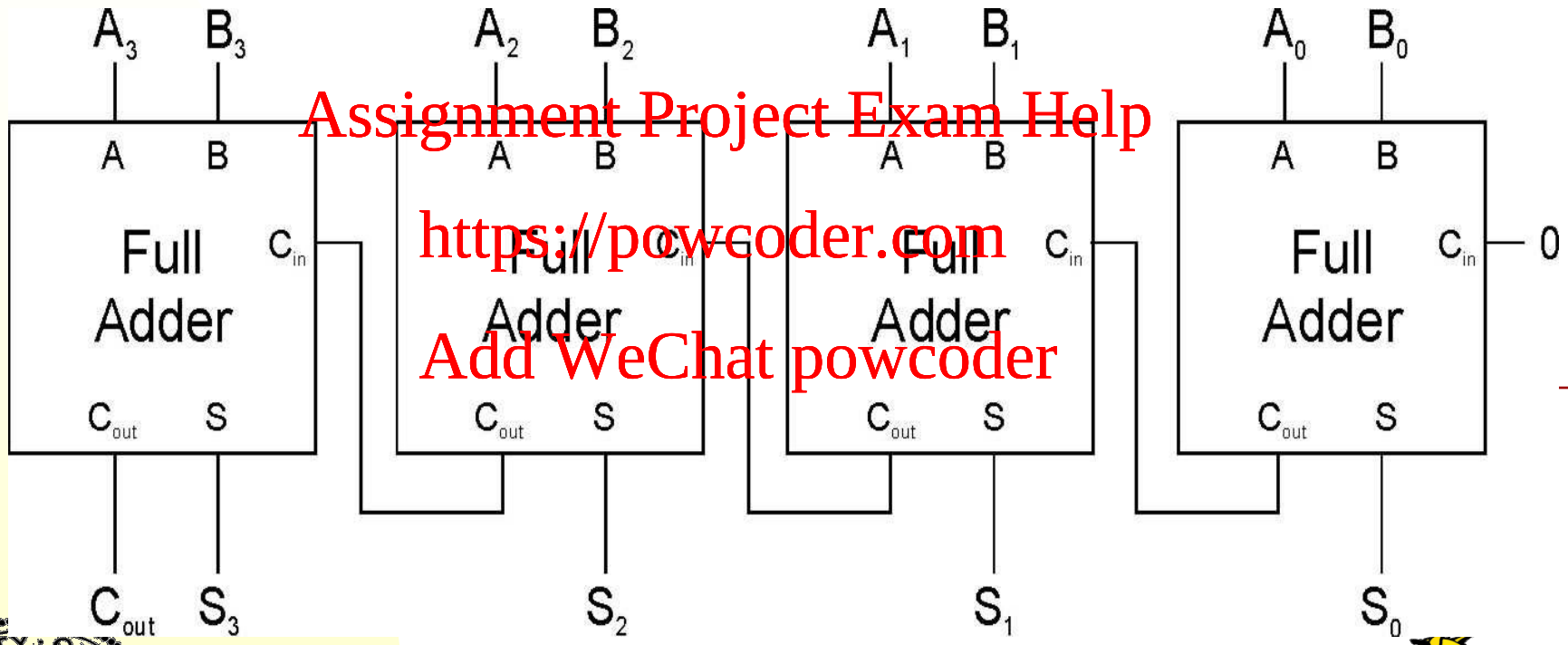
One-Bit Full Adder

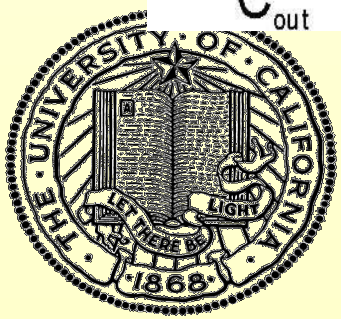
A	B	C_{in}	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



Four-Bit Full Adder

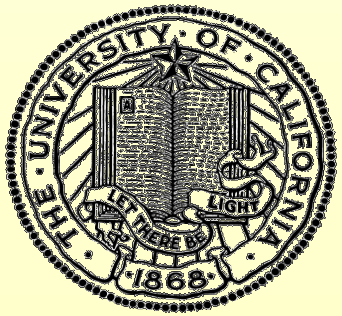
Ripple-carry adder





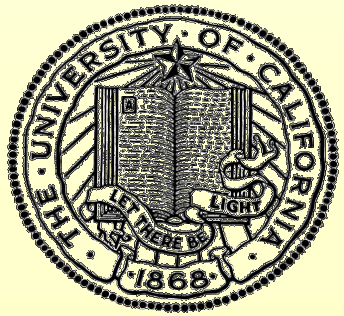
Masking

- Want to look only at certain bits of a binary word
- Use a mask to remove the uninteresting bits
- Example:
  - Two values: 01001101 and 01001001
  - If we want to see bit 3 from right, we AND it with 00000100 to get
    - 00000100 and 00000000, respectively.

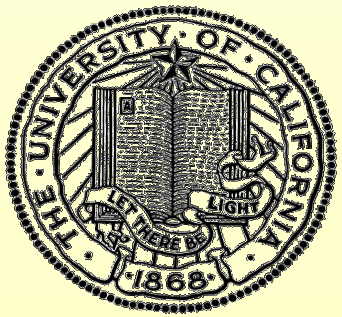


Building functions from logic gates

- Combinational Logic Circuit
  - Output depends only on the current inputs
  - Stateless (memoryless)
- Sequential Logic Circuit
  - Output depends on the sequence of inputs (past and present)
  - Stores information (state) from past inputs



Sequential Circuits and Memory



Combinational vs. Sequential

- Combinational circuit
  - Always gives the same output for a given set of inputs
  - Example: Adder always generates sum and carry, regardless of previous inputs
- Sequential circuit
  - **Remembers** previous input
  - Output depends on state and input

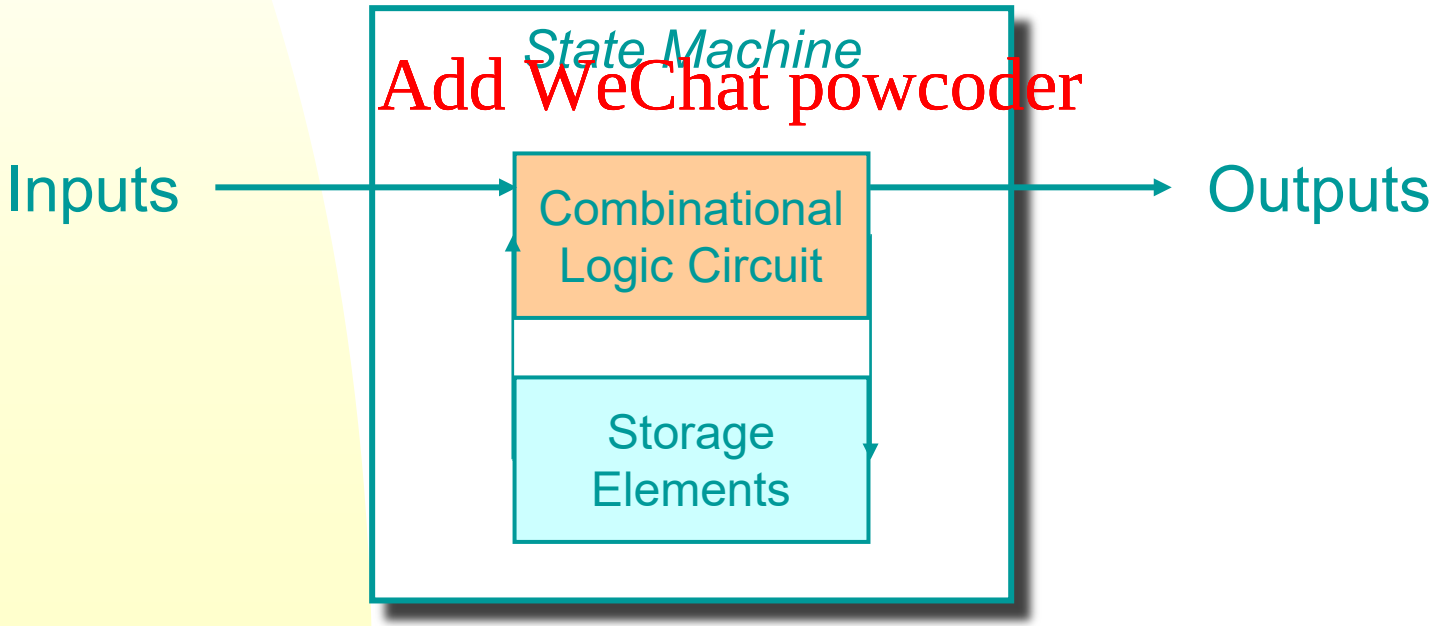
Sequential Circuits

- **Store** information
- Output depends on stored information (state) plus input
  - So a given input might produce different outputs, depending on the stored information
- Example: ticket counter
  - Advances when you push the button
  - Output depends on previous state

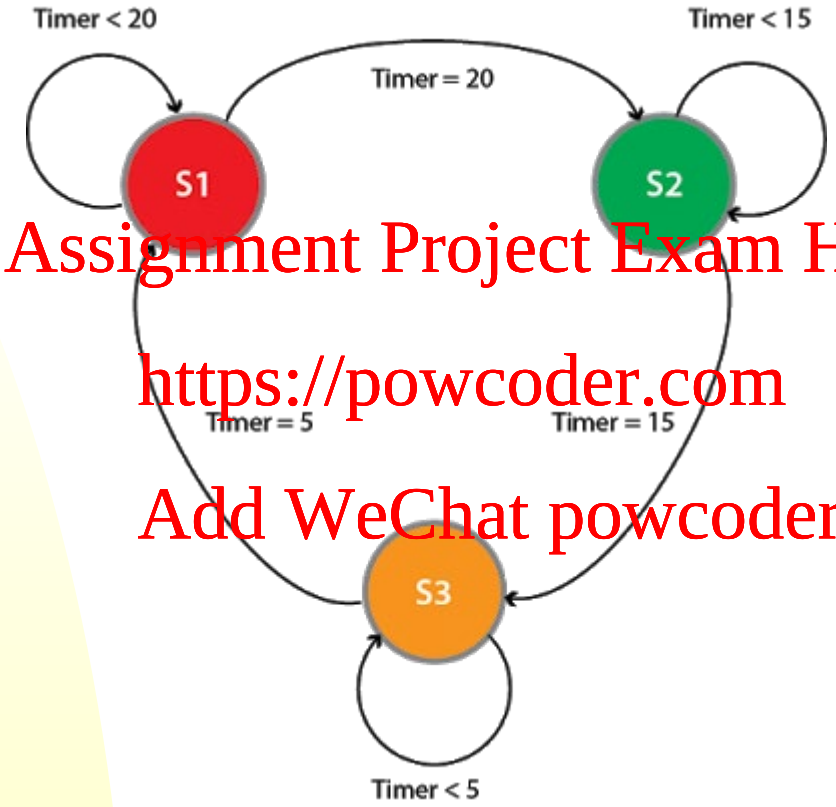
State Machine

The basic type of sequential circuit

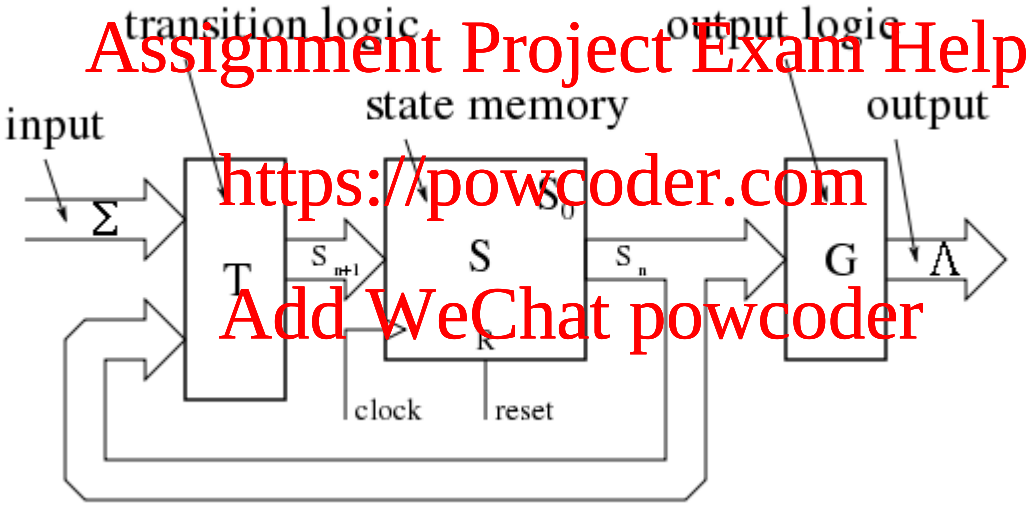
- Combines combinational logic with storage
- "Remembers" state, and changes output (and state) based on inputs and current state



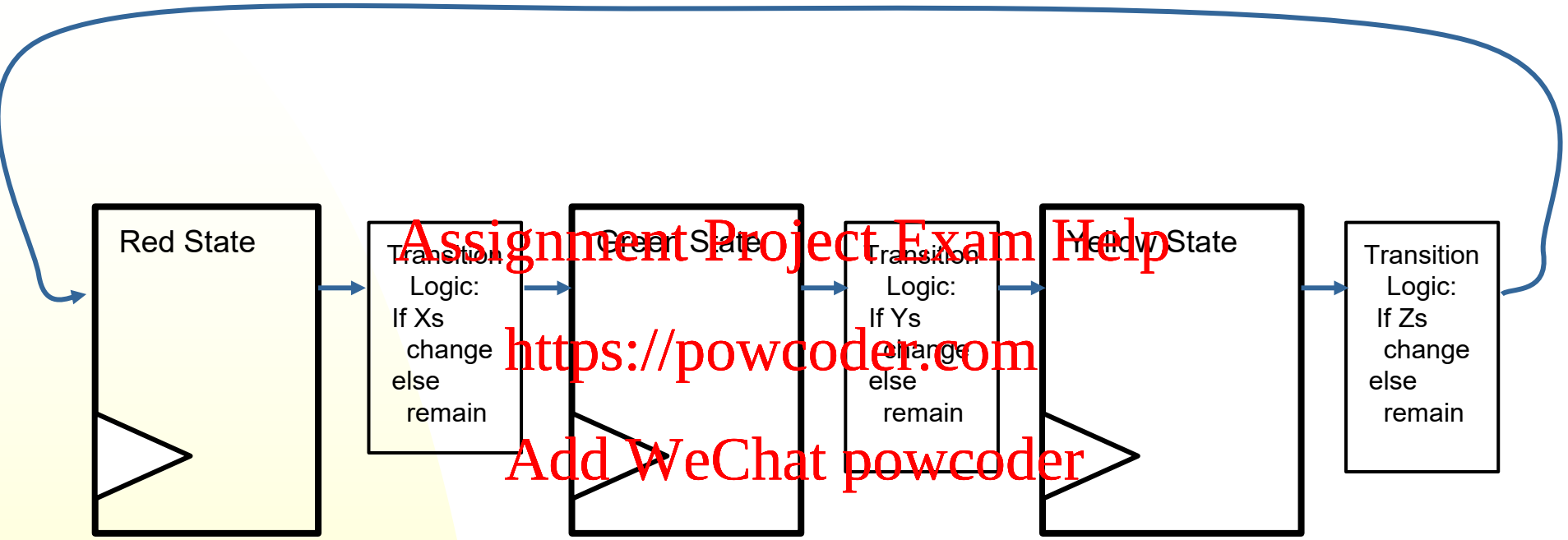
Example of State Machine: Traffic Light



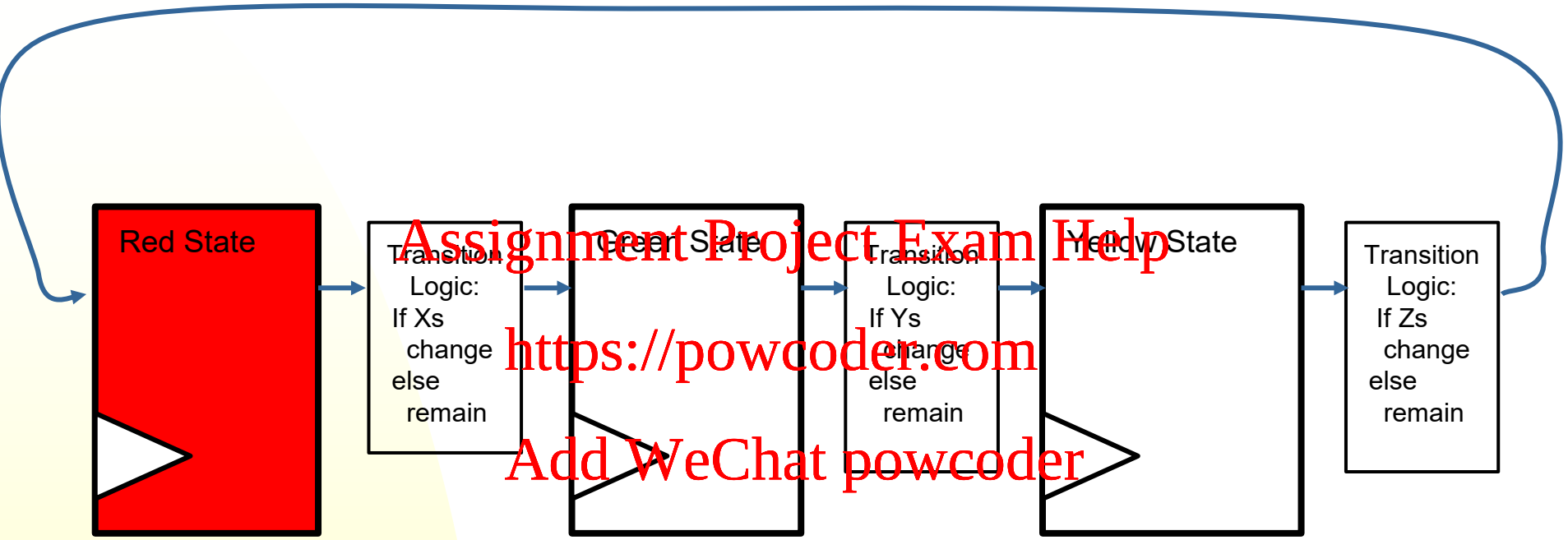
State Machine: Overview



State Machine: Traffic Light



State Machine: Traffic Light

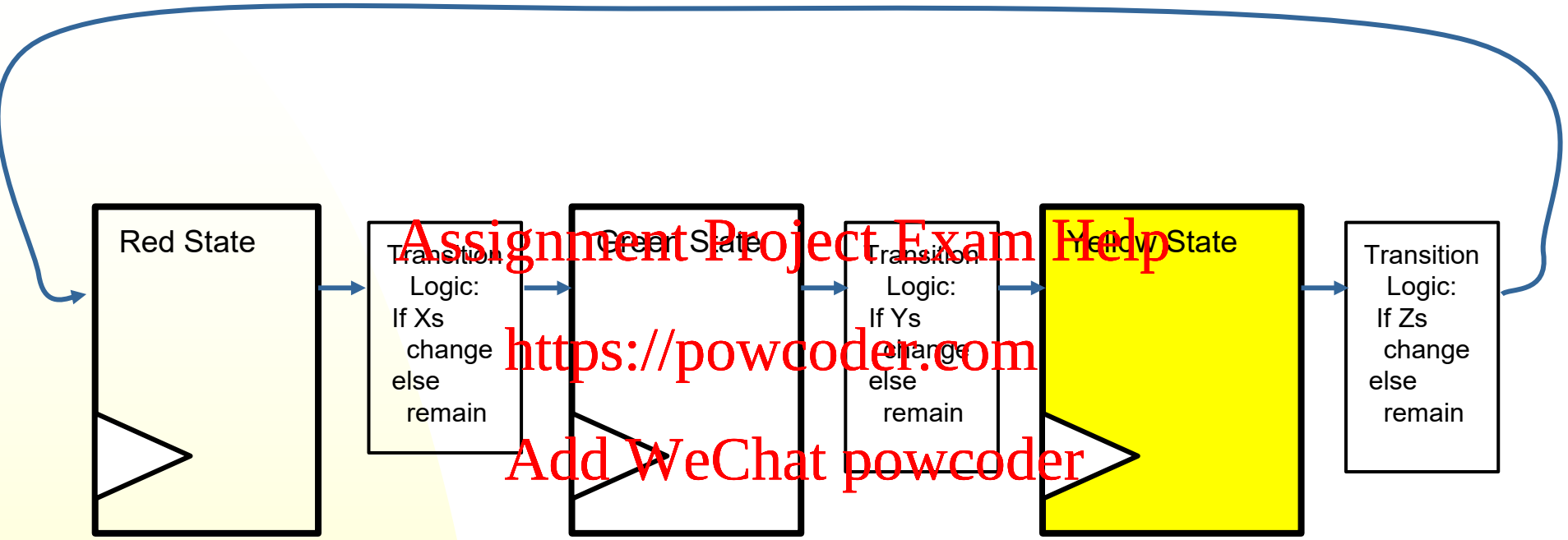


- Start off in default state Red

State Machine: Traffic Light

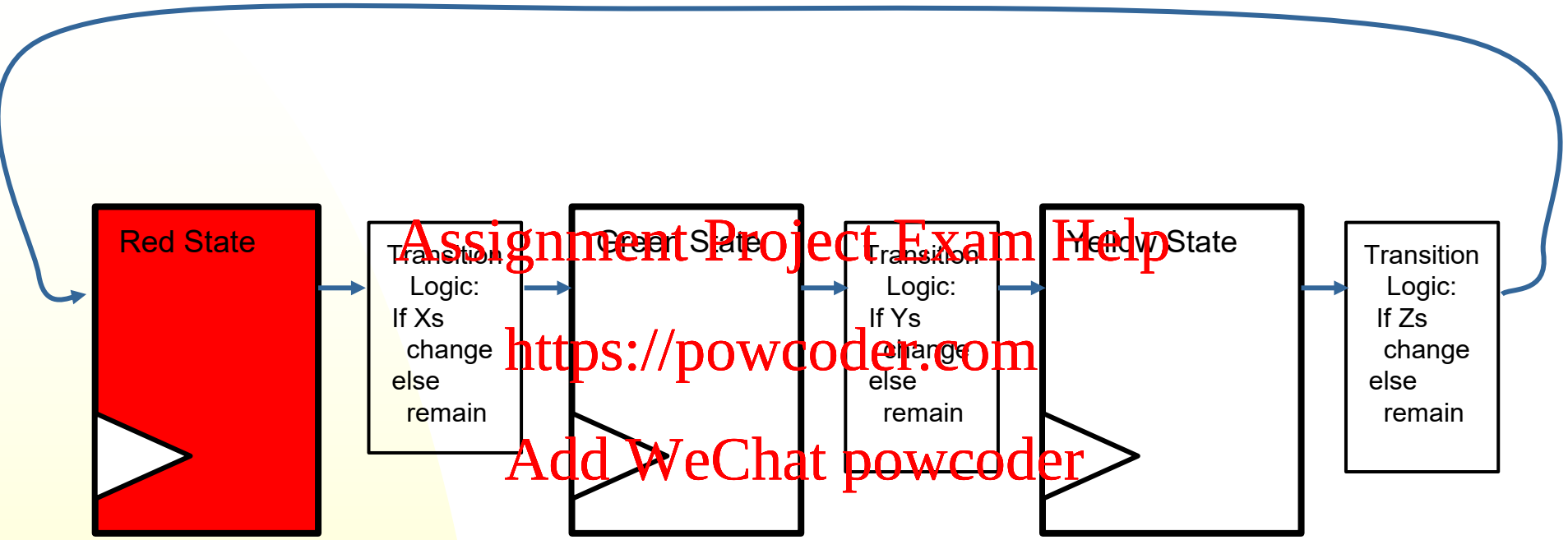
- After a specified time Xs switch to next state

State Machine: Traffic Light



- After a specified time Ys switch to next state

State Machine: Traffic Light

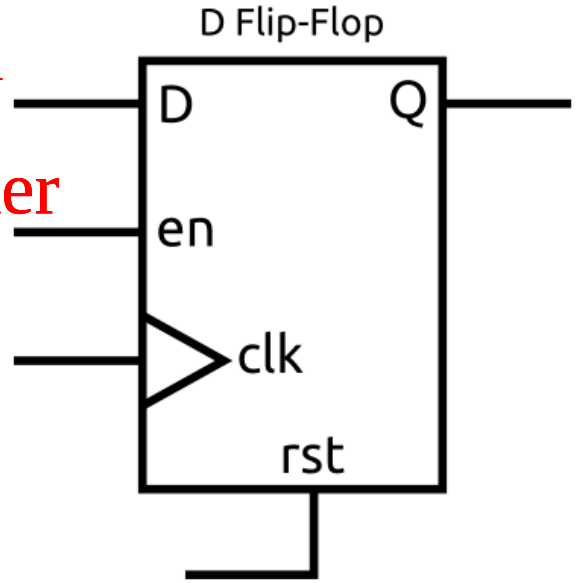


- After a specified time Zs switch to next state which is Red.

D Flip-Flops

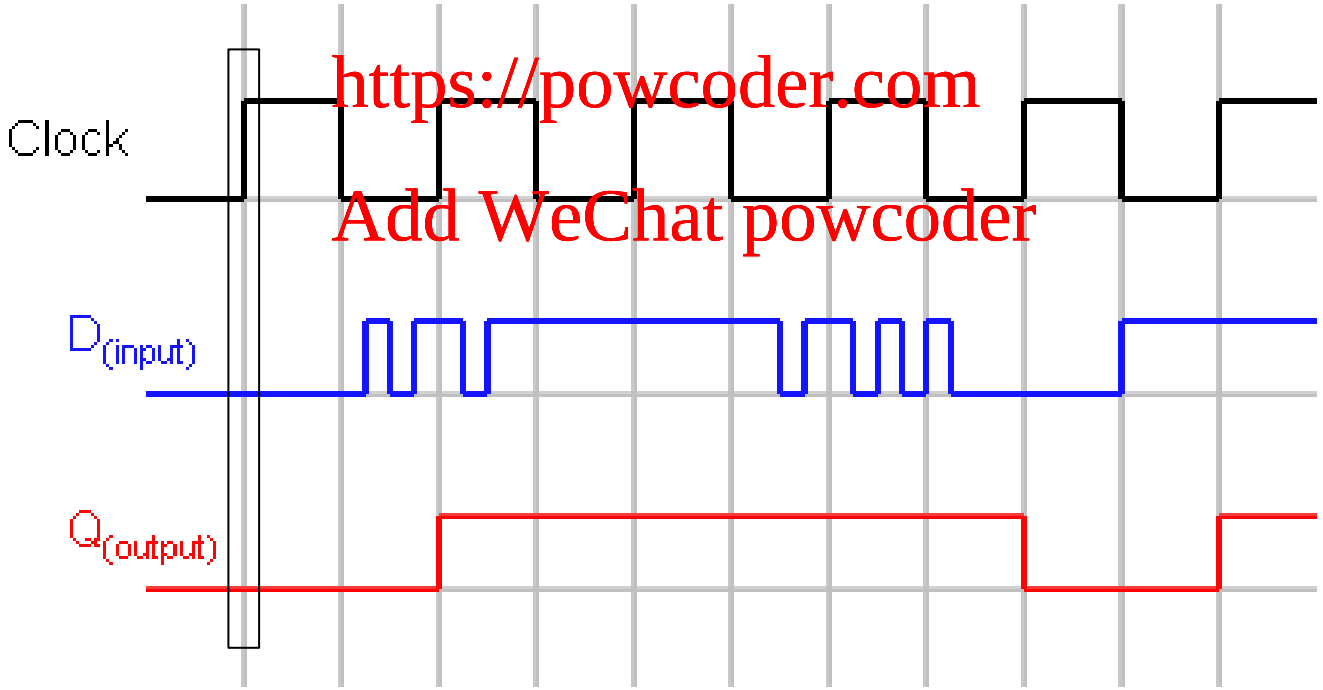
Memory device

- Can be positive triggered or negative triggered (by a clock usually abbreviated by clk)
- Different types e.g. RS, JK
- Inputs/Outputs:
  - D: input signal
  - clk: Clock signal
  - en: if 0 Q holds its value, if 1, Q becomes D at clk edge.
  - rst: if 1 then Q becomes 0
  - Q: output signal



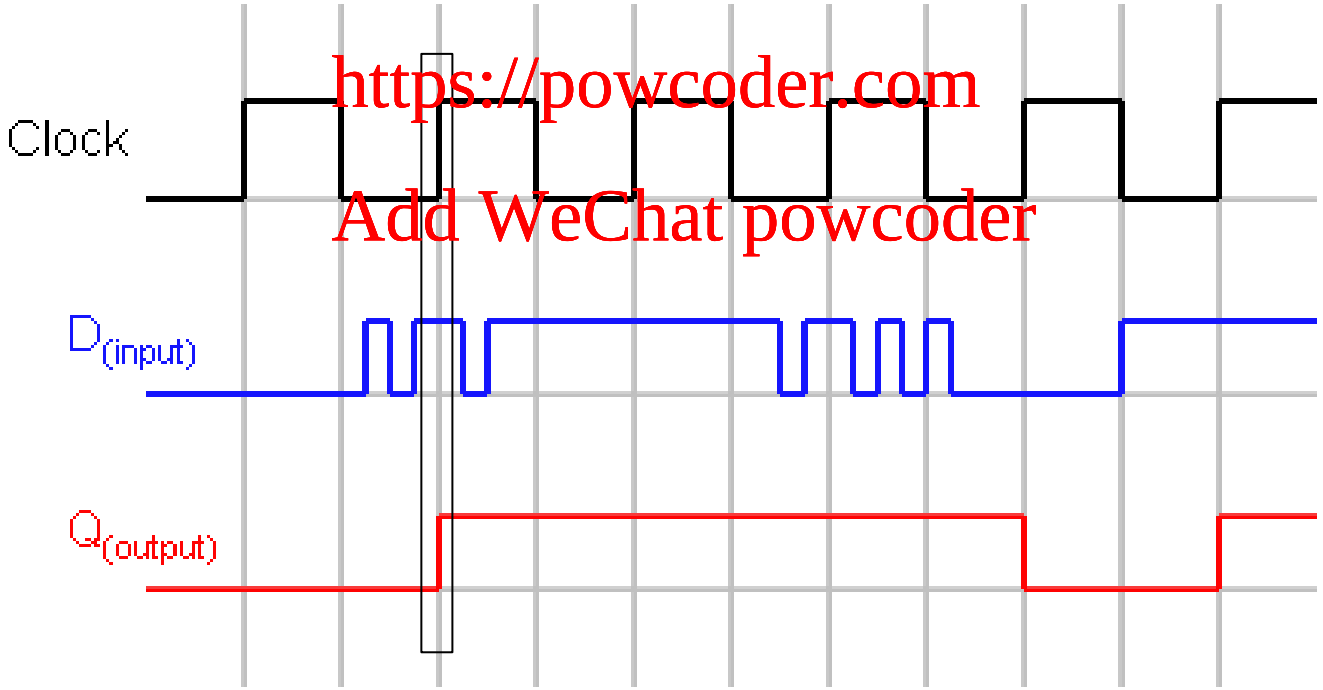
D flip-flop(positive-edge) timing diagram

- Q becomes D at positive clk edge (0 -> 1).
  - Stores value until next positive clk edge.
- clk oscillates between 0 and 1
  - frequency = 1/period



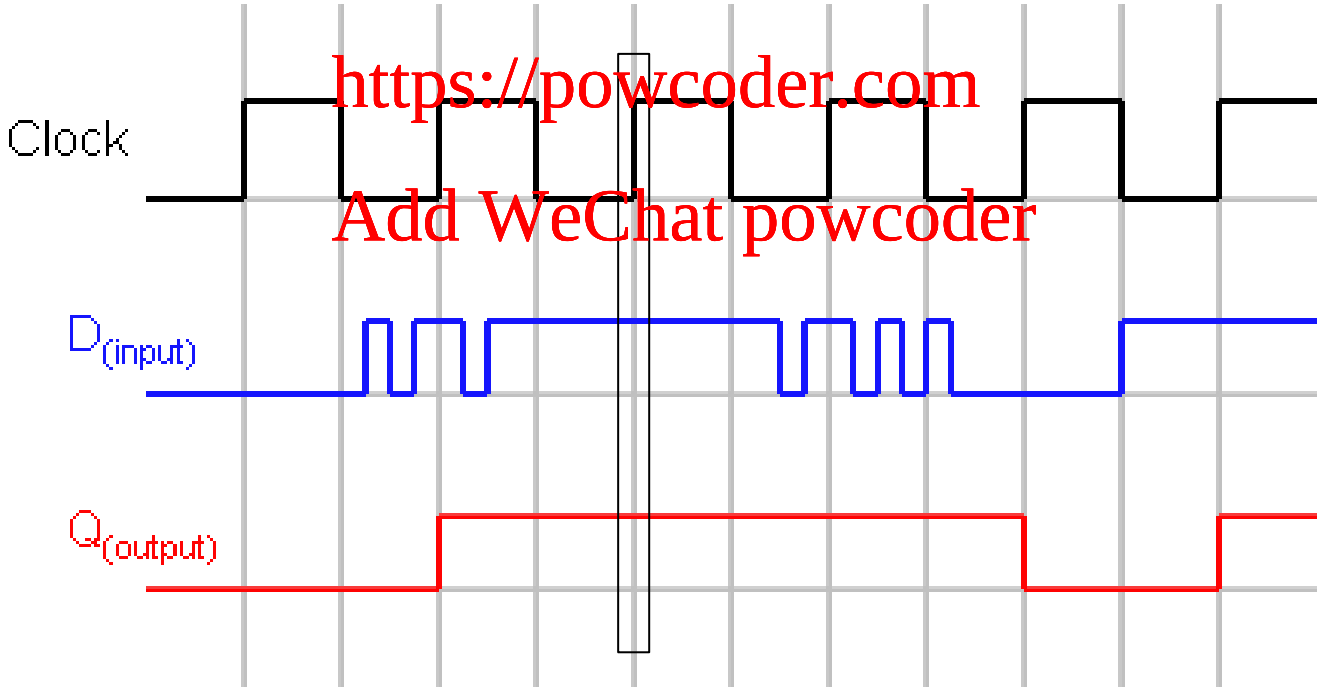
D flip-flop(positive-edge) timing diagram

- Q becomes D at positive clk edge (0 -> 1).
  - Stores value until next positive clk edge.
- clk oscillates between 0 and 1
  - frequency = 1/period



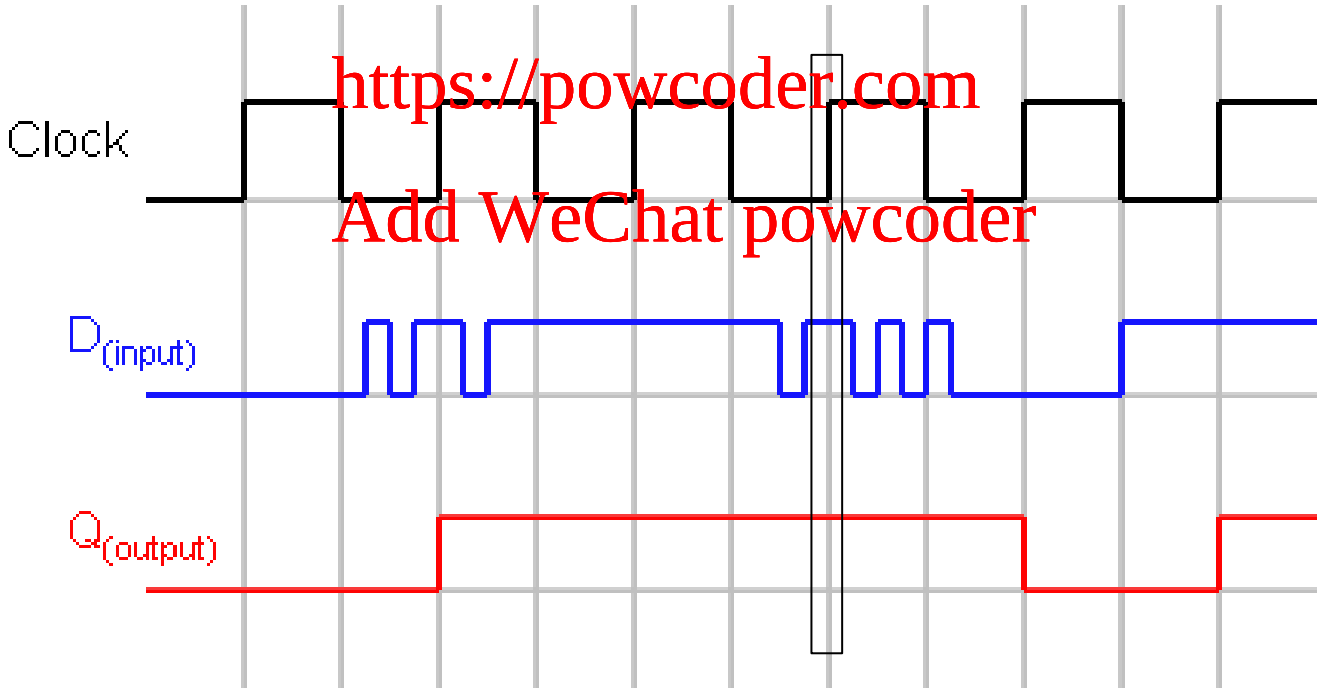
D flip-flop(positive-edge) timing diagram

- Q becomes D at positive clk edge.
  - Stores value until next positive clk edge.
- clk oscillates between 0 and 1
  - frequency = 1/period



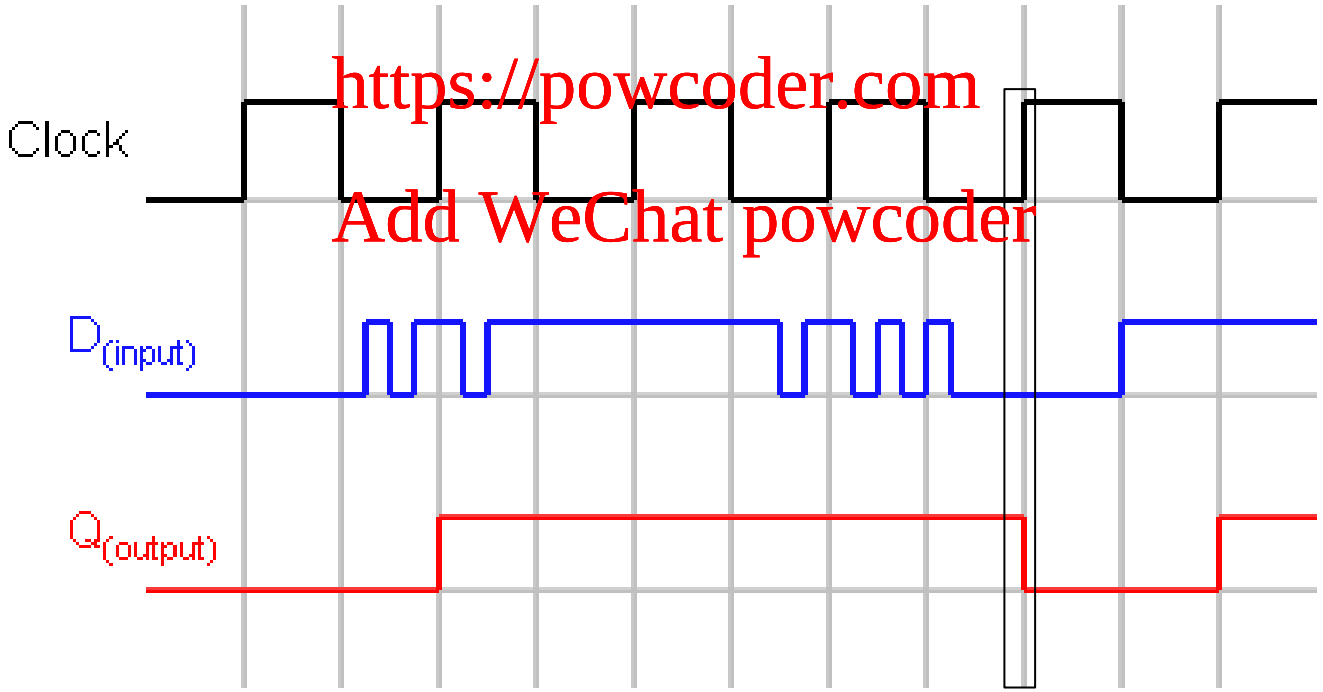
D flip-flop(positive-edge) timing diagram

- Q becomes D at positive clk edge.
  - Stores value until next positive clk edge.
- clk oscillates between 0 and 1
  - frequency = 1/period



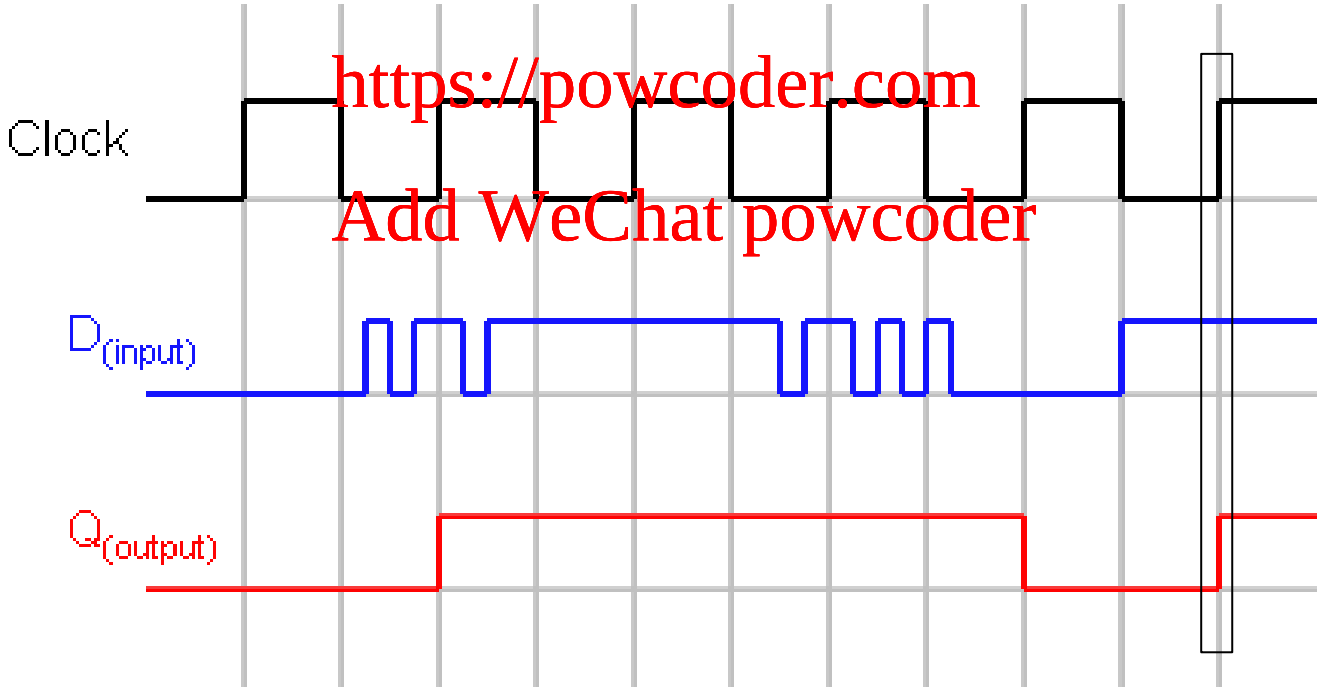
D flip-flop(positive-edge) timing diagram

- Q becomes D at positive clk edge.
  - Stores value until next positive clk edge.
- clk oscillates between 0 and 1
  - frequency = 1/period



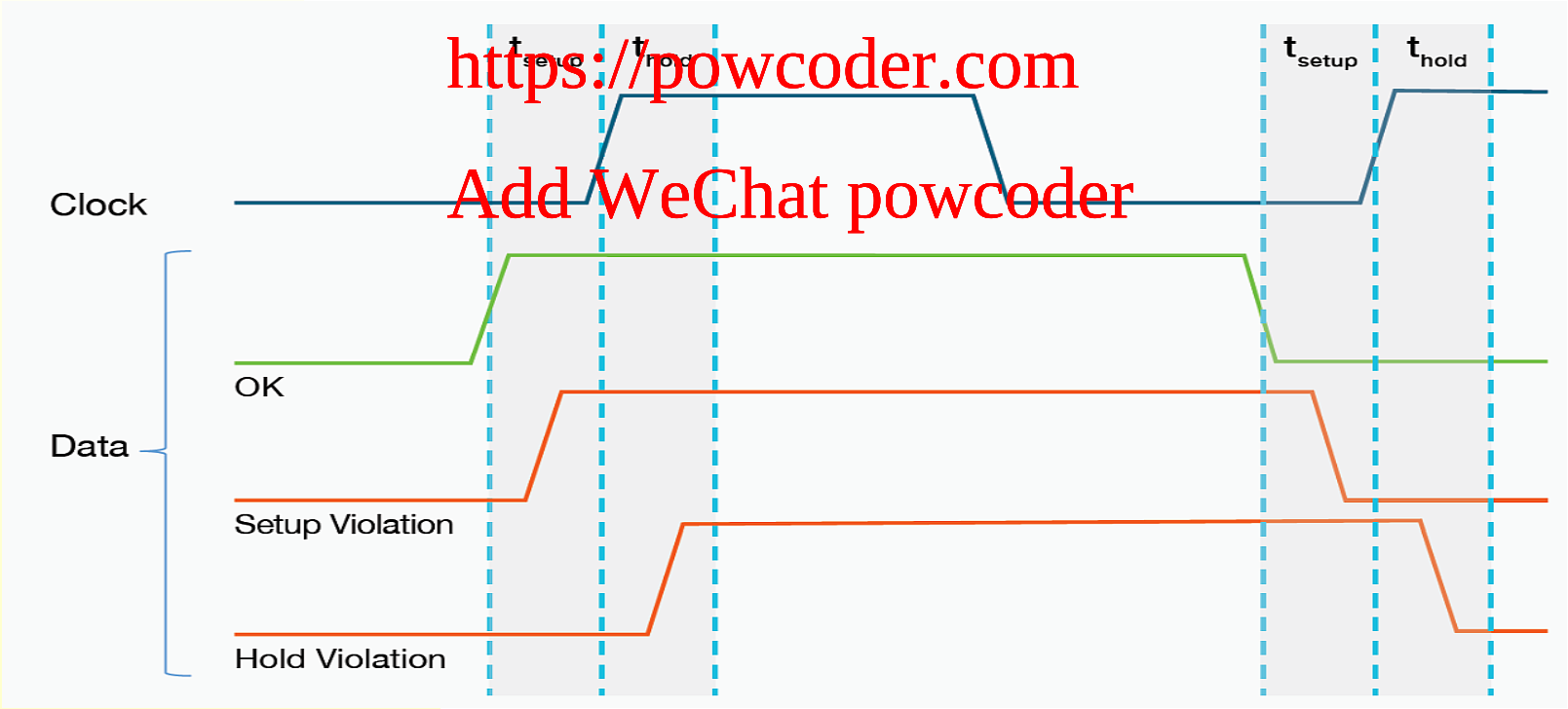
D flip-flop(positive-edge) timing diagram

- Q becomes D at positive clk edge.
  - Stores value until next positive clk edge.
- clk oscillates between 0 and 1
  - frequency = 1/period



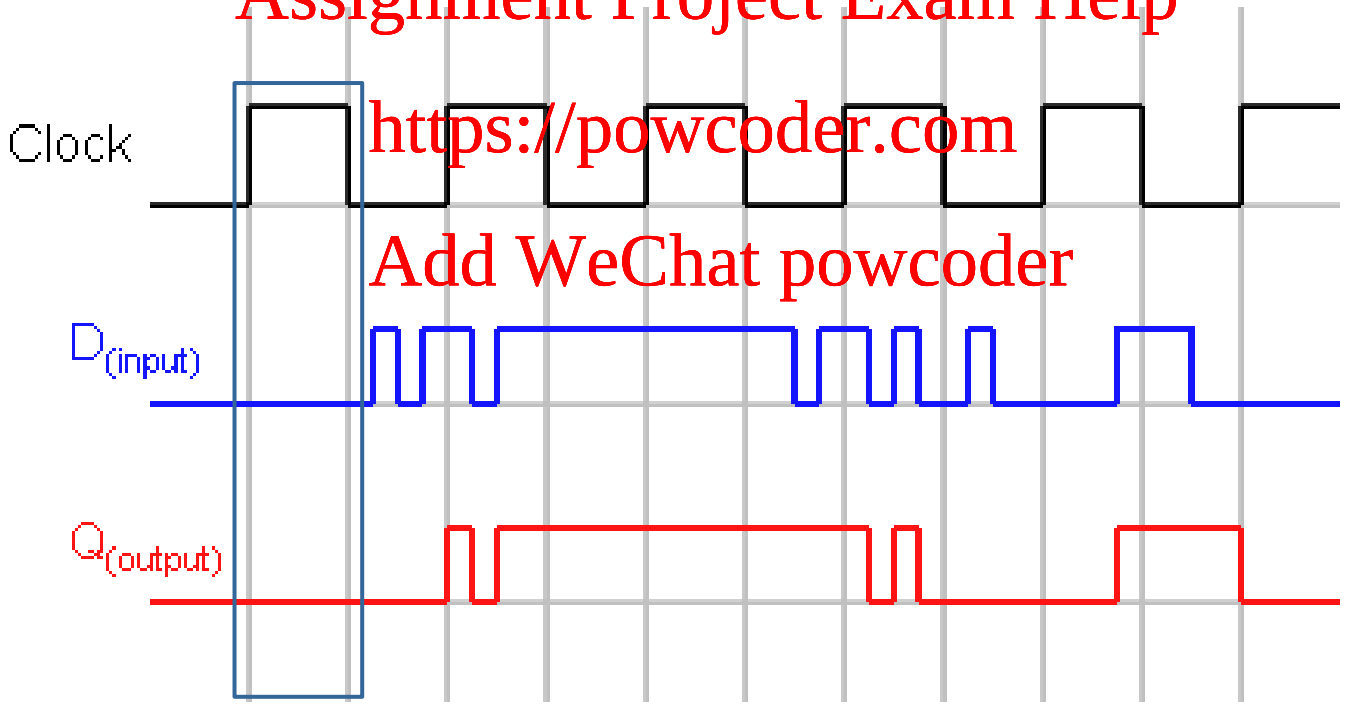
Setup and Hold Time

- Setup time: Time before clock edge where signal has to be stable
- Hold time: Time after clock edge where signal has to be stable



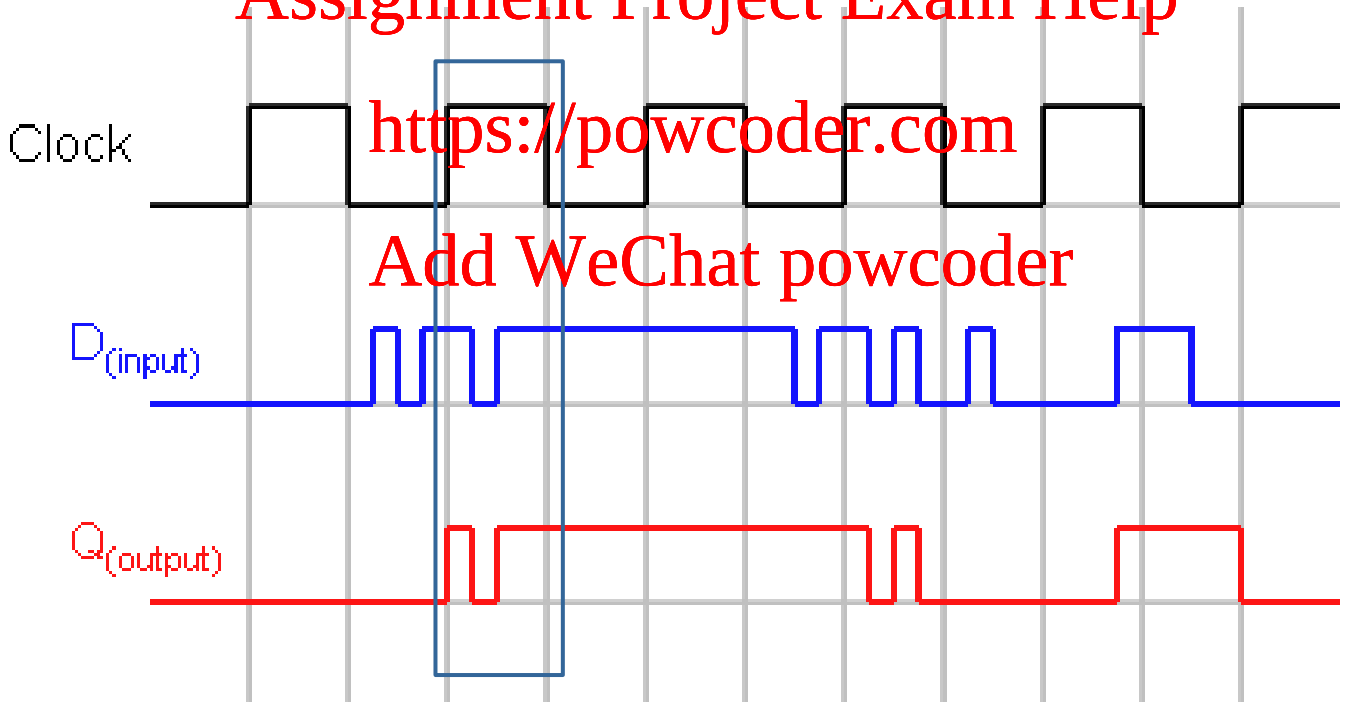
Latch

- Output is equal to Input when clk is high.
- Stores last value when clk is low.



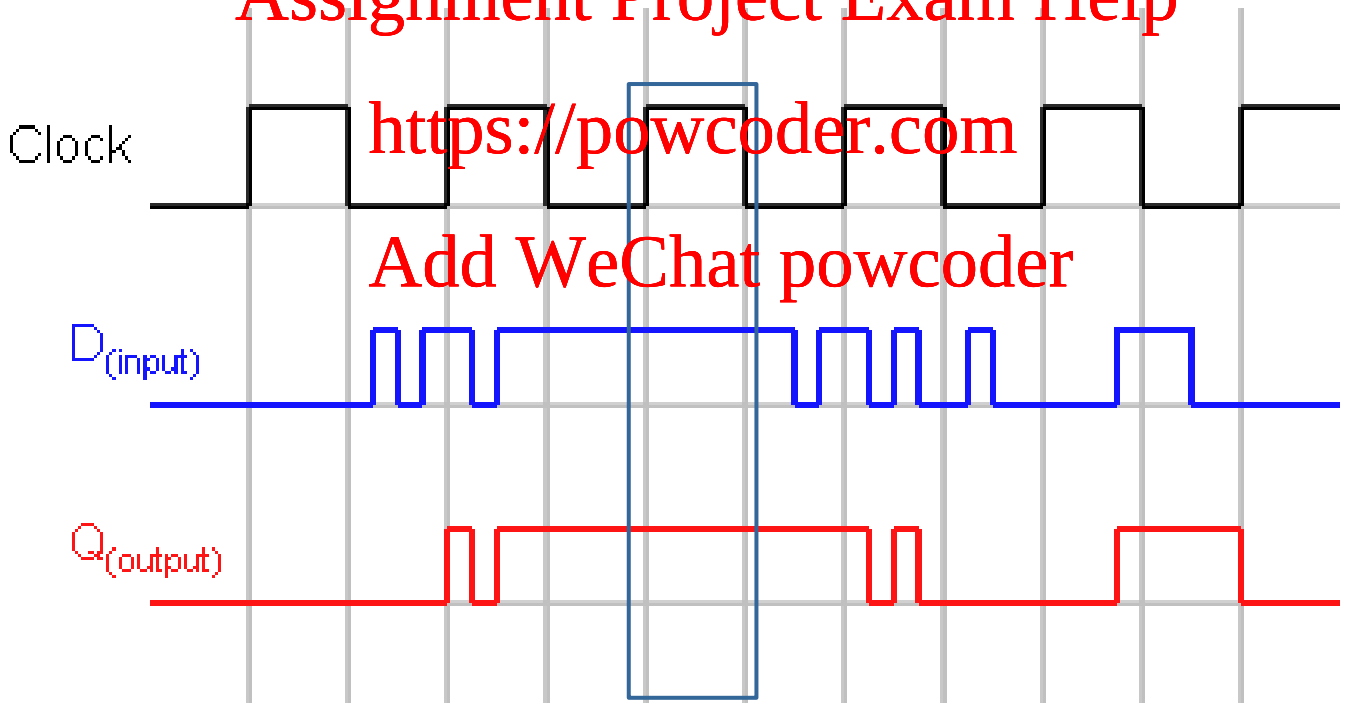
Latch

- Output is equal to Input when clk is high.
- Stores last value when clk is low.



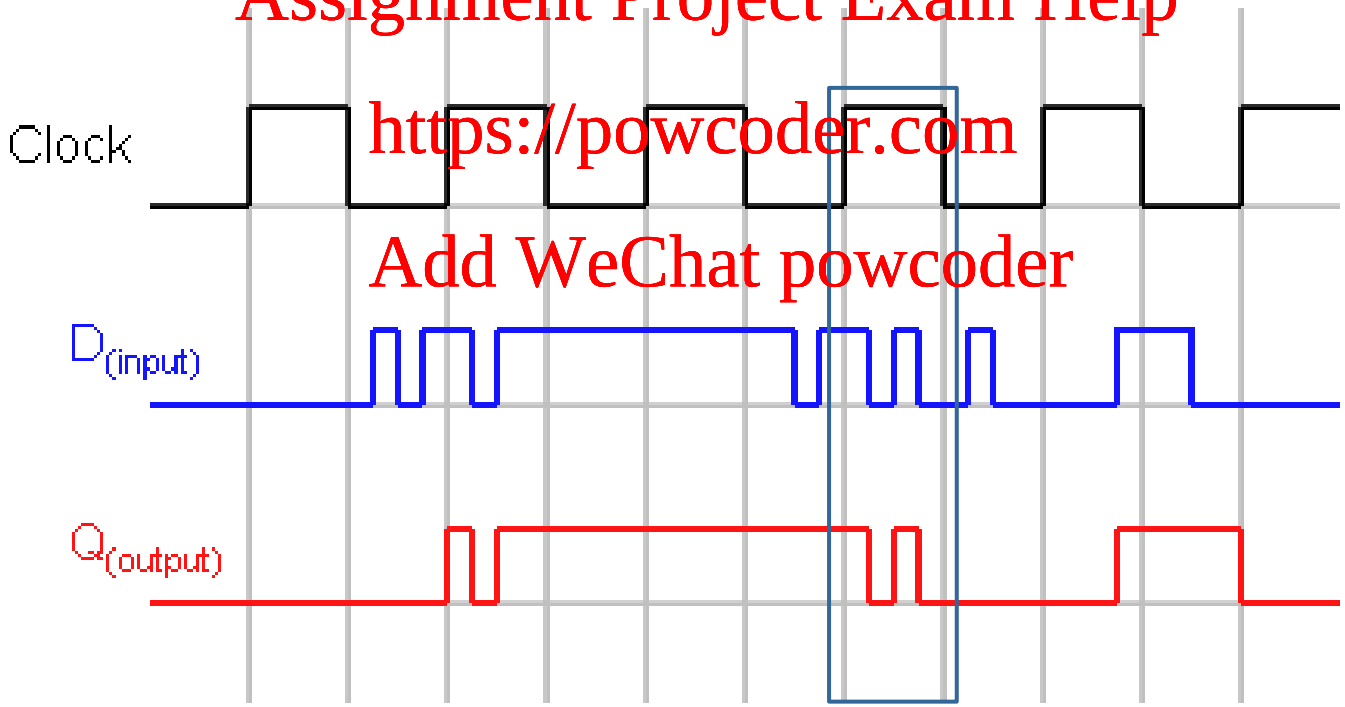
Latch

- Output is equal to Input when clk is high.
- Stores last value when clk is low.



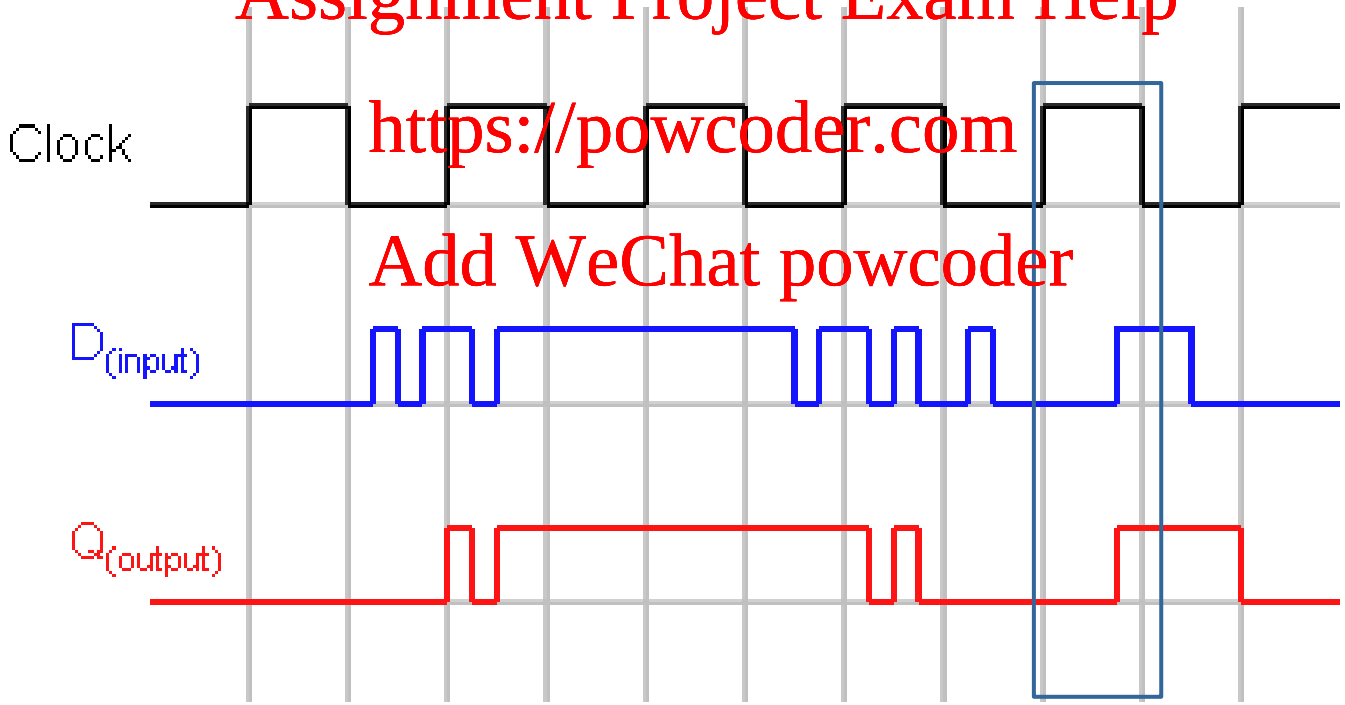
Latch

- Output is equal to Input when clk is high.
- Stores last value when clk is low.



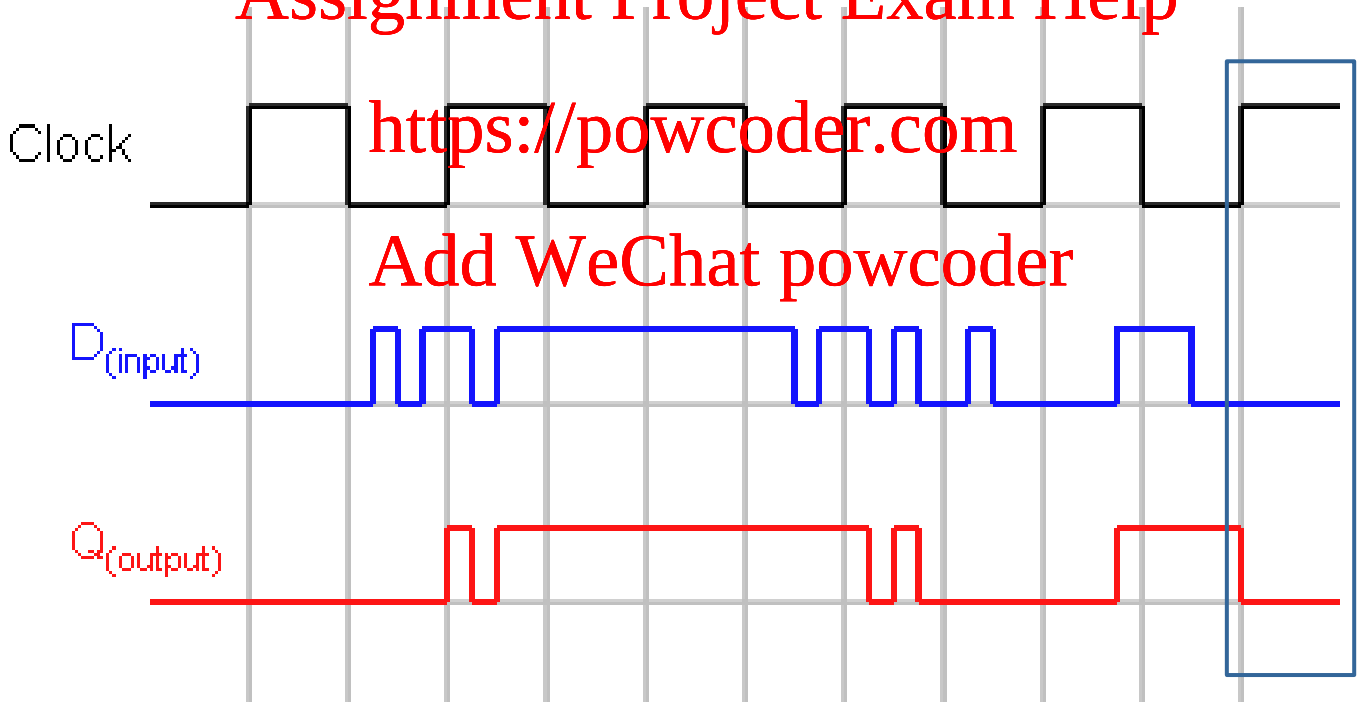
Latch

- Output is equal to Input when clk is high.
- Stores last value when clk is low.



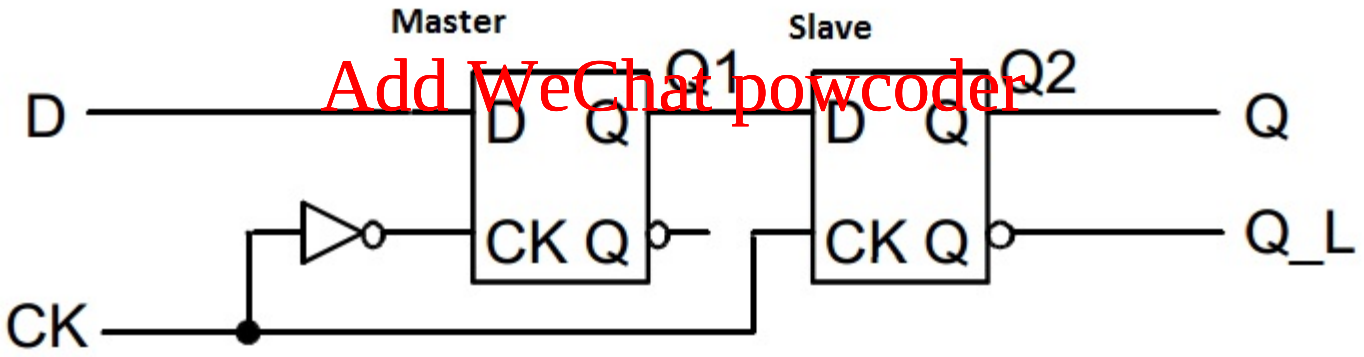
Latch

- Output is equal to Input when clk is high.
- Stores last value when clk is low.



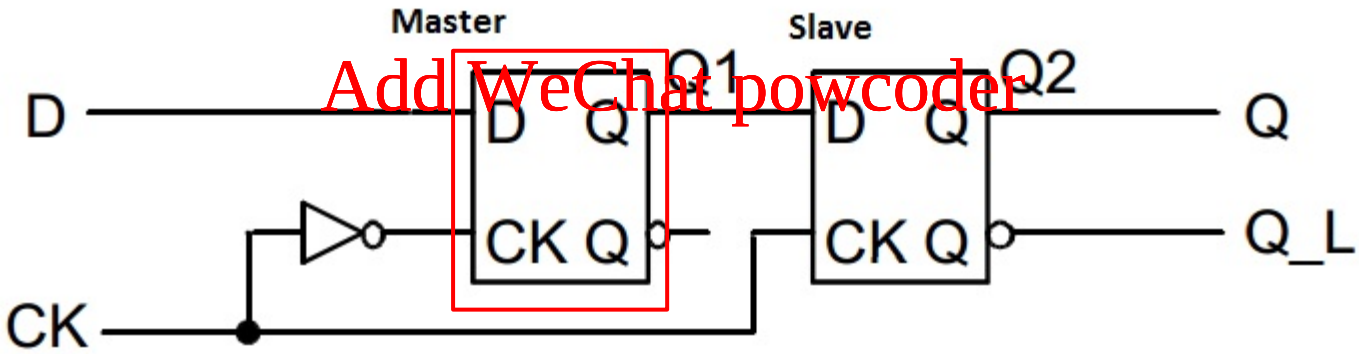
One way to make a Flip-Flop: Two Latches

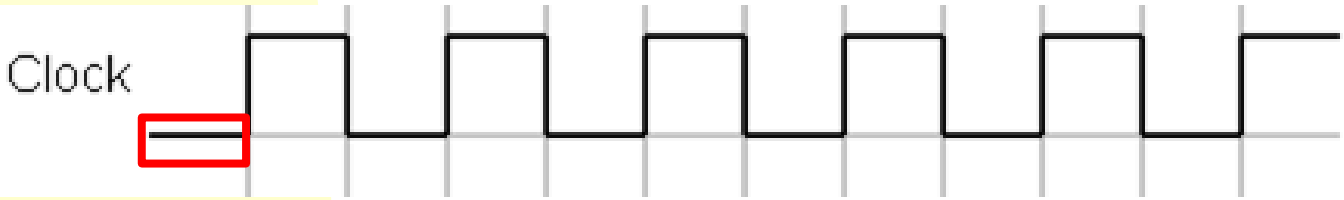
- One latch(master) is connected to clk' and the other(slave) to clk (positive triggered).
- When clk transitions to high, slave captures last value of master which is Q1 and holds since it's clk is low.



One way to make a Flip-Flop: Two Latches

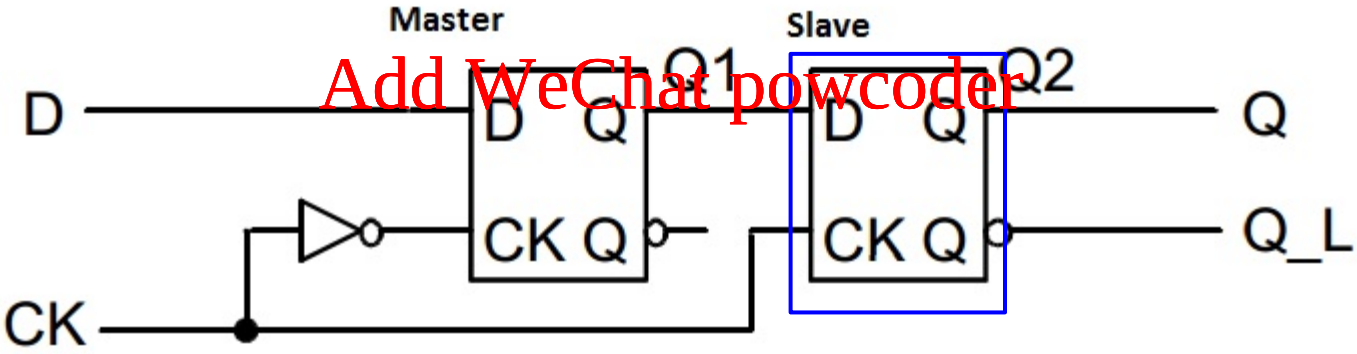
- One latch(master) is connected to clk' and the other(slave) to clk (positive triggered).
- When clk transitions to high, slave captures last value of master which is held since it's clk is low.

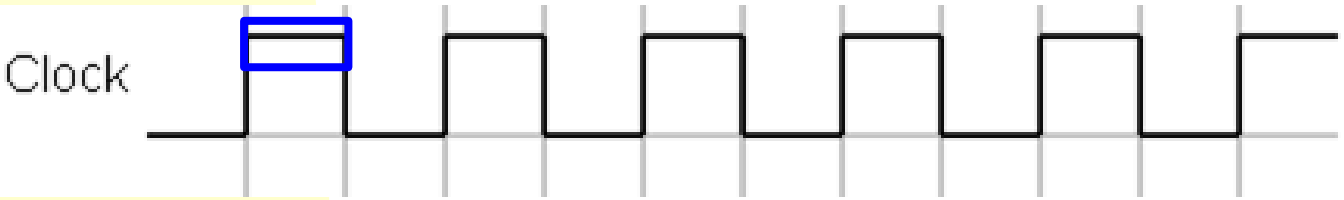




One way to make a Flip-Flop: Two Latches

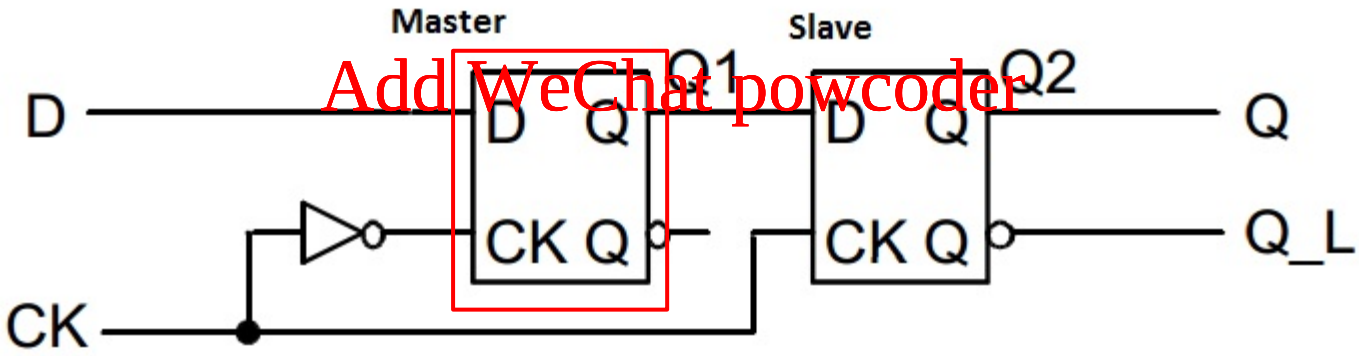
- One latch(master) is connected to clk' and the other(slave) to clk (positive triggered).
- When clk transitions to high, slave captures last value of master which is Q1 and holds since it's clk is low.

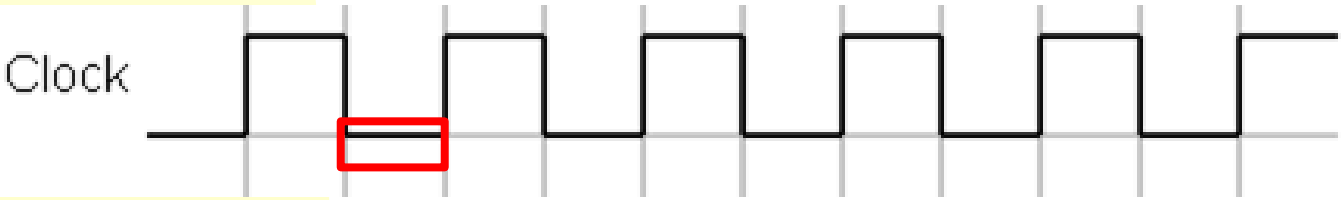




One way to make a Flip-Flop: Two Latches

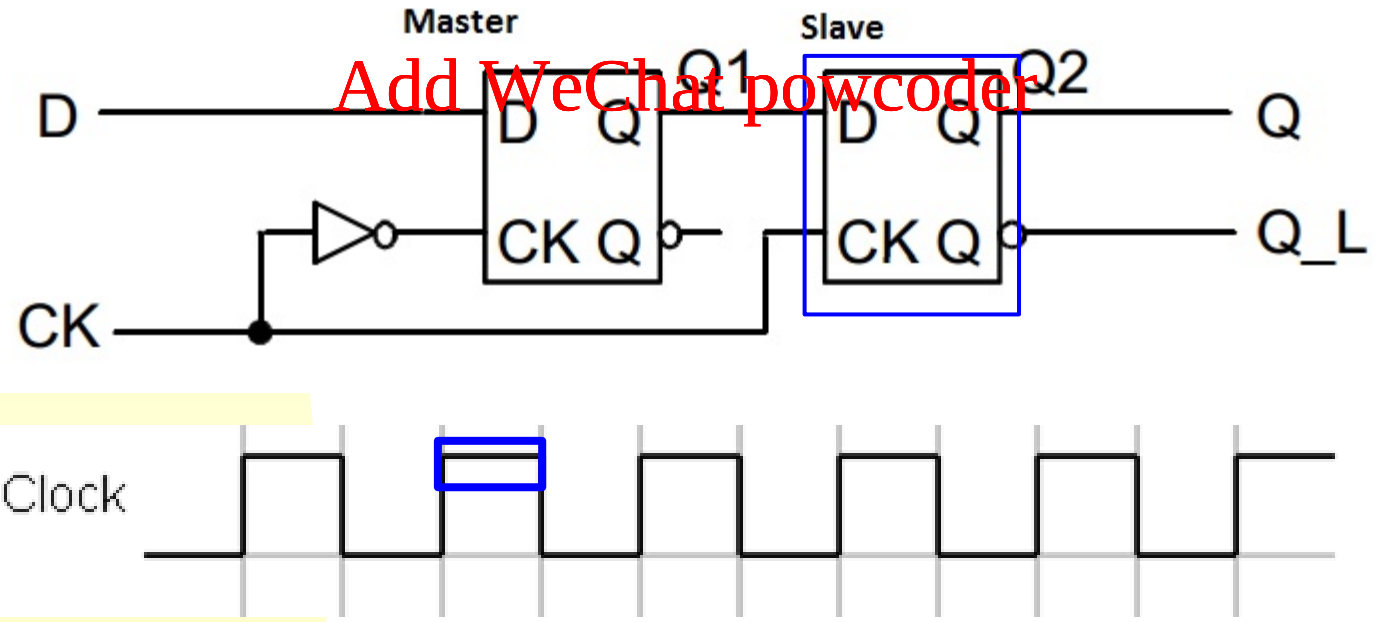
- When clk transitions to high, slave captures last value of master which is now stored since it's clk is low.
- When clk transitions to low, master is open again but slave is closed, retaining value





One way to make a Flip-Flop: Two Latches

- One latch(master) is connected to clk' and the other(slave) to clk (positive triggered).
- When clk transitions to high, slave captures last value of master which is Q1 and holds since it's clk is low.

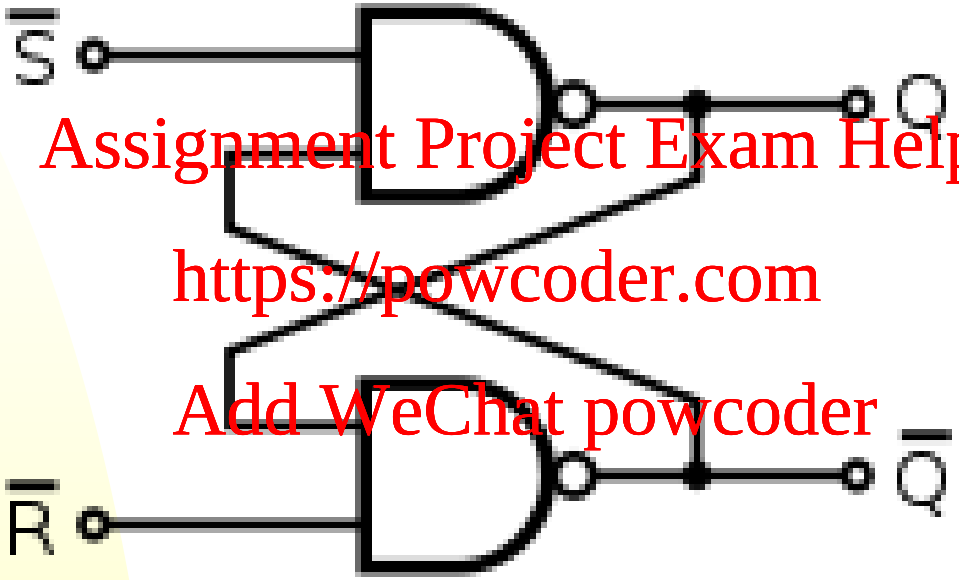


Reset-Set (RS) Latch – or $\overline{SR}$

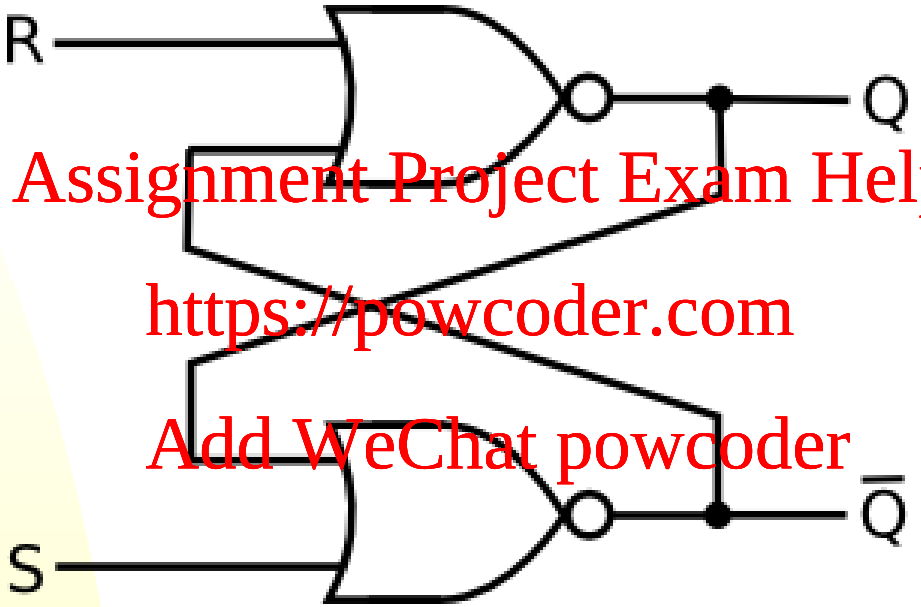
- Two inputs: Set and Reset
- Set to 0 one of the two inputs at a time to store a value, S sets, R clears
- The transition to 00 generates an undefined output

SR latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

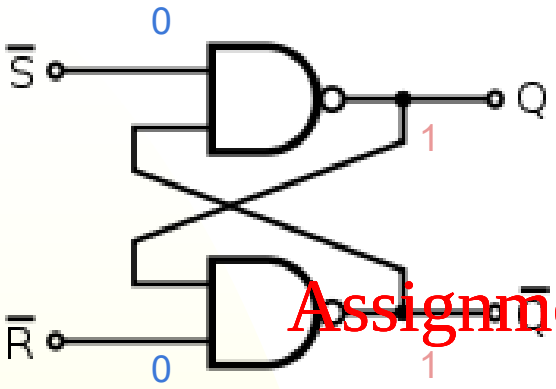
R-S Latch



R-S Latch Nor Gates

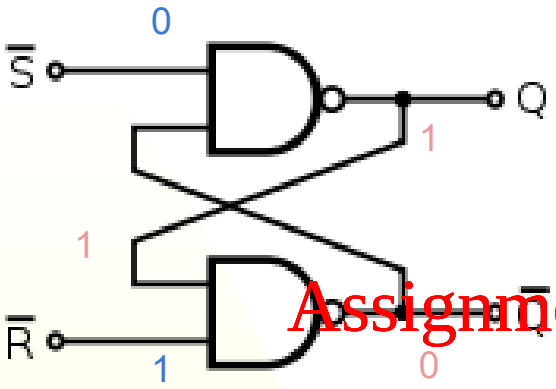


Four SR Latch States: S' 0, R' 0



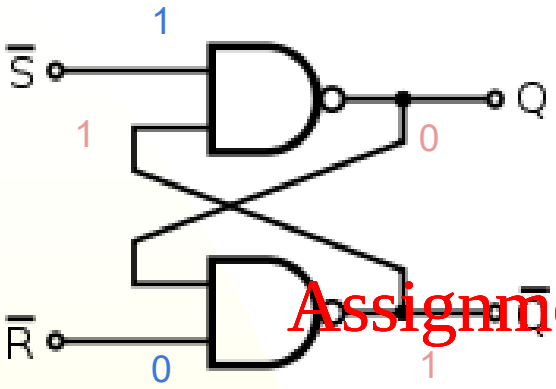
$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

Four SR Latch States: S' 0, R' 1



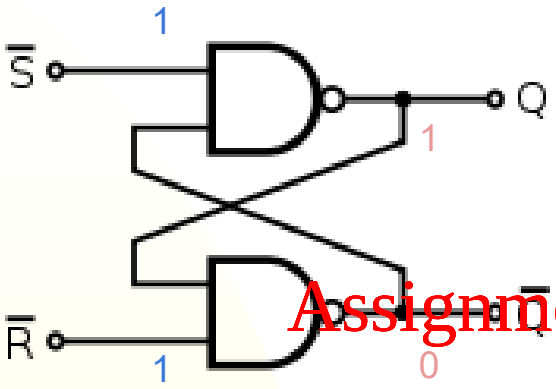
$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

Four SR Latch States: S' 1, R' 0



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

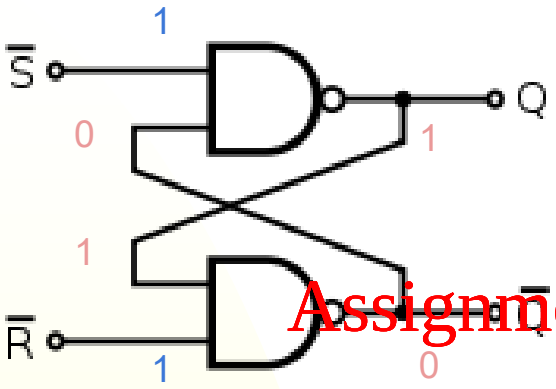
Four SR Latch States: S' 1, R' 1 Memory



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

Q = 1, Q' = 0

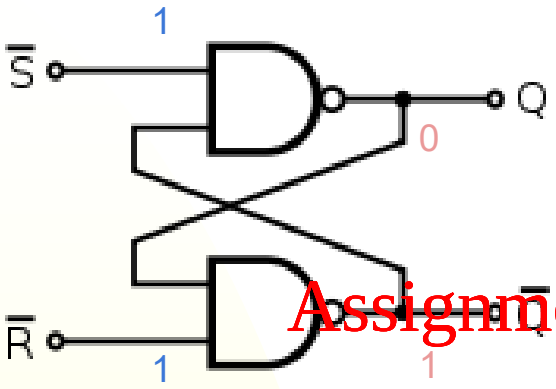
Four SR Latch States: S' 1, R' 1 Memory



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

Q = 1, Q' = 0

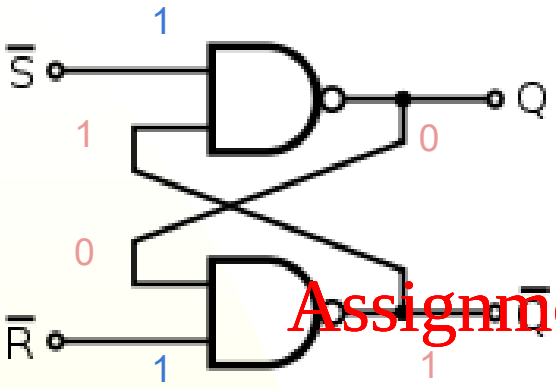
Four SR Latch States: S' 1, R' 1 Memory



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

Q = 0, Q' = 1

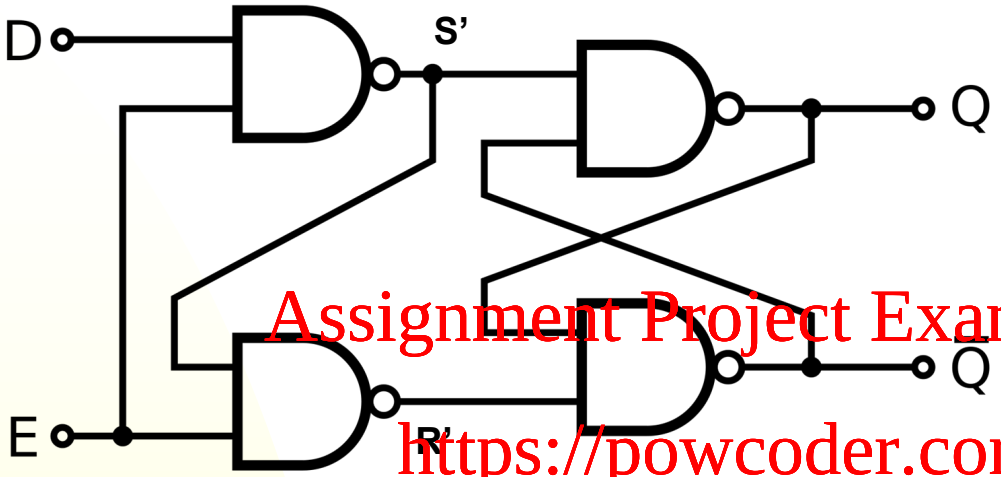
Four SR Latch States: S' 1, R' 1 Memory



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

Q = 0, Q' = 1

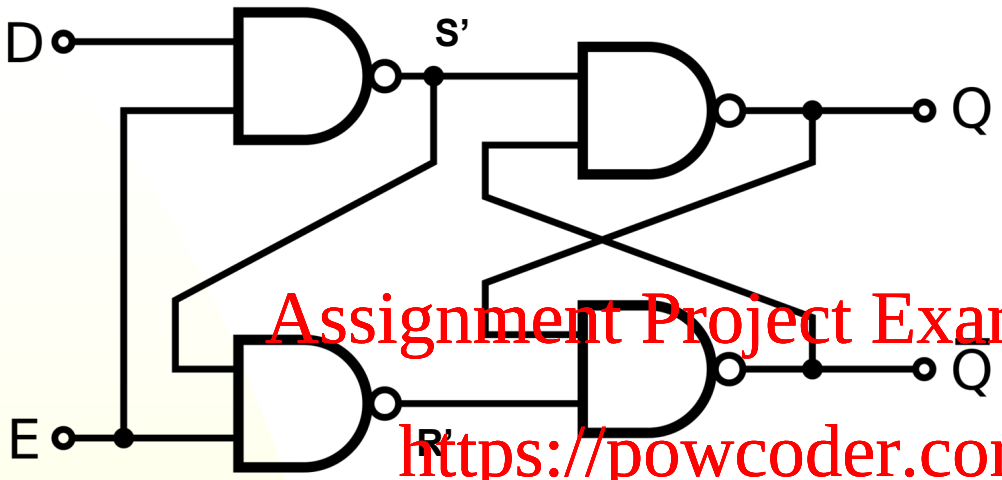
D Latch



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

E/clk	D	R'	S'	Q	Q'	Comment
0	0	1	1	Q	Q'	Keep state

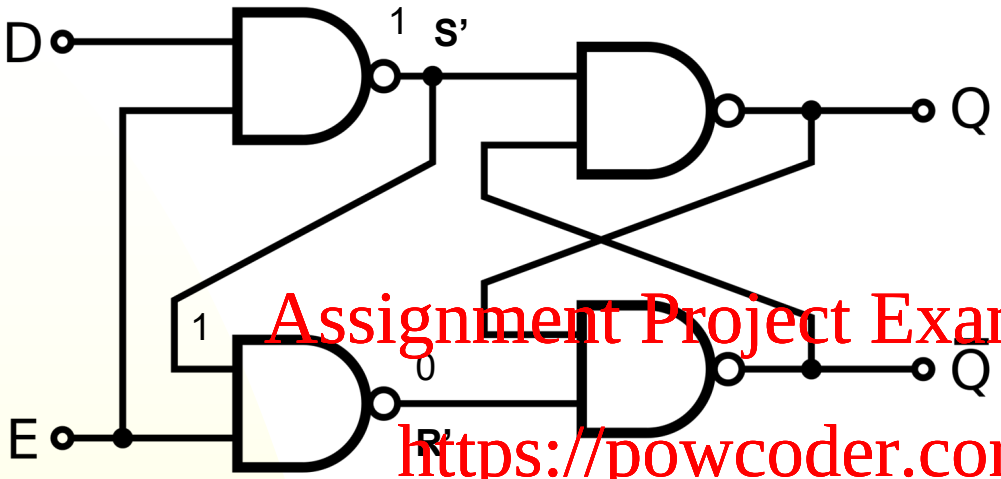
D Latch



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

E/clk	D	R'	S'	Q	Q'	Comment
0	0	1	1	Q	Q'	Keep state
0	1	1	1	Q	Q'	Keep state

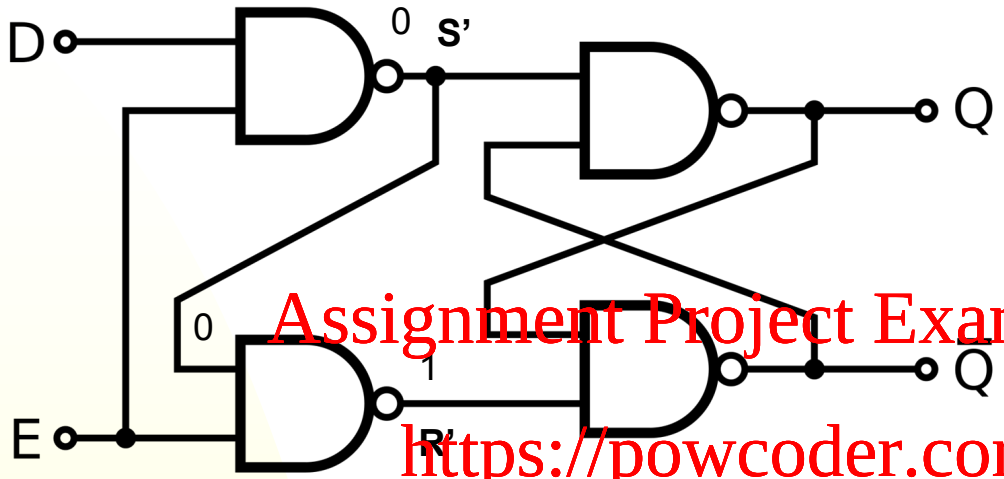
D Latch



$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

E/clk	D	R'	S'	Q	Q'	Comment
0	0	1	1	Q	Q'	Keep state
0	1	1	1	Q	Q'	Keep state
1	0	0	1	0	1	D = Q

D Latch

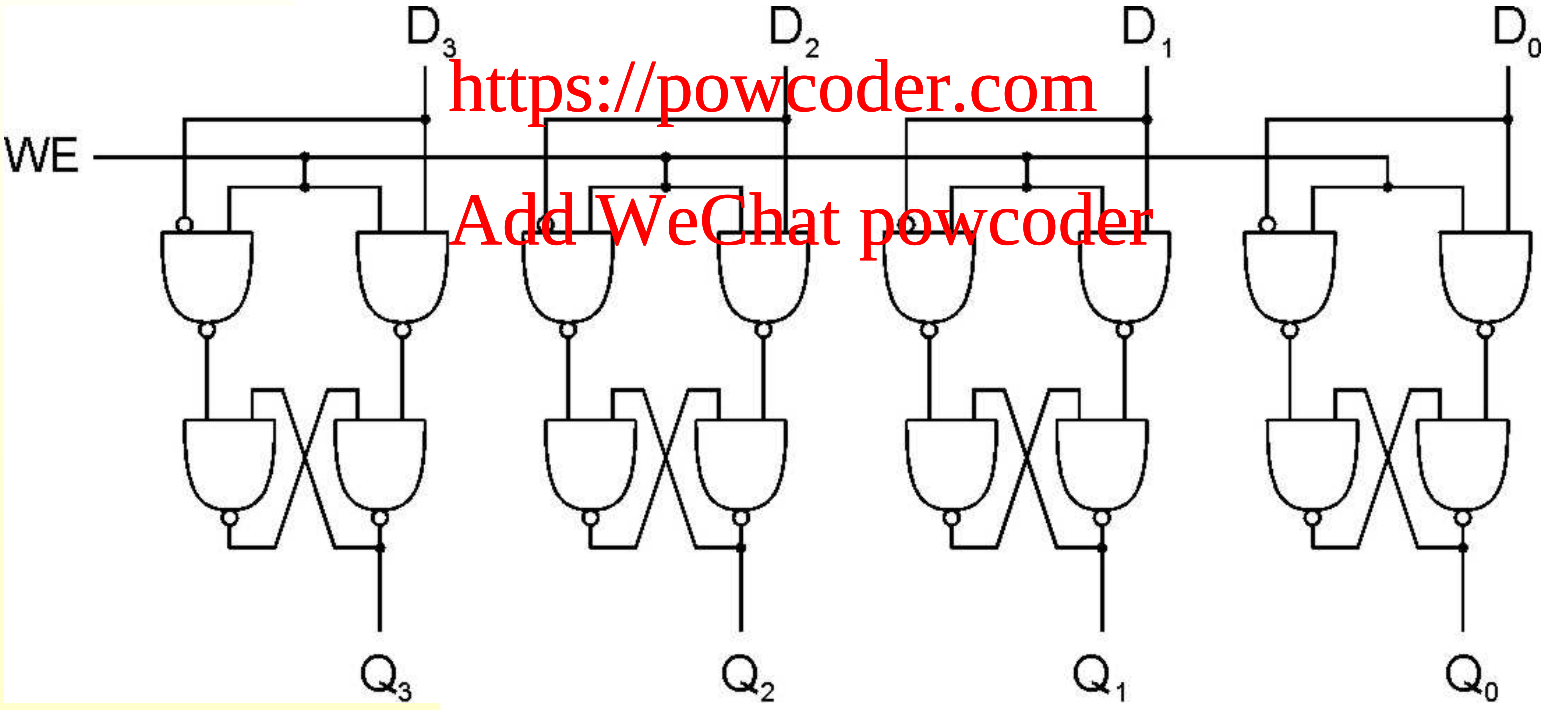


$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	Keep state

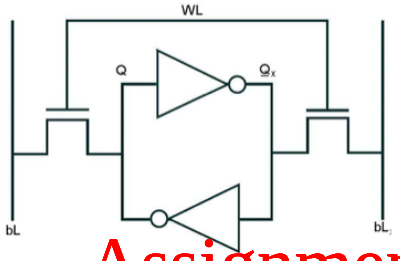
E/clk	D	R'	S'	Q	Q'	Comment
0	0	1	1	Q	Q'	Keep state
0	1	1	1	Q	Q'	Keep state
1	0	0	1	0	1	D = Q
1	1	1	0	1	0	D = Q

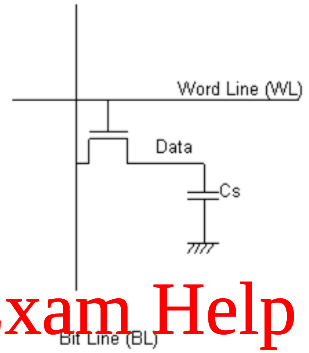
Register

- A register stores a multi-bit value
- Common WE which latches the n-bit value



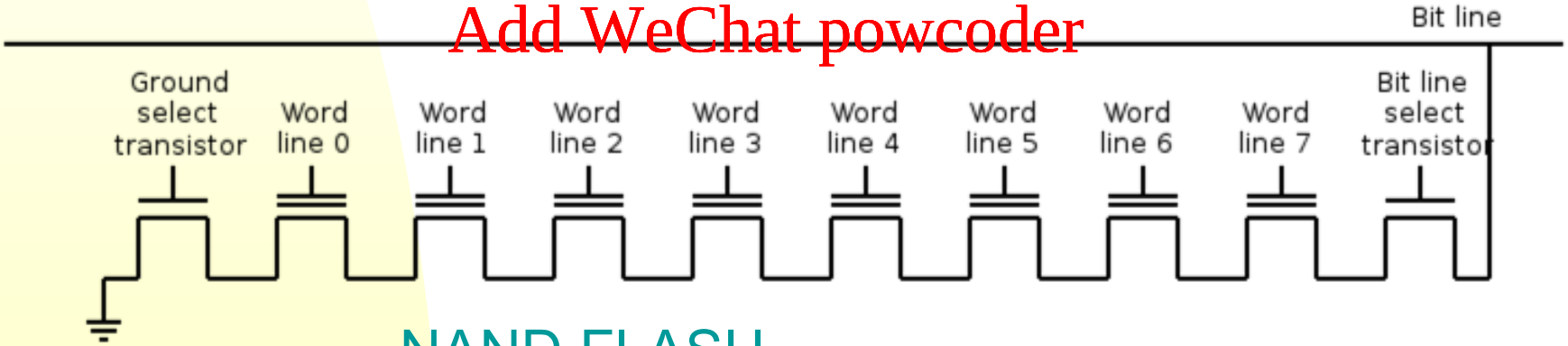
Other types of memory…





SRAM
(on-chip usually)

DRAM
(off-chip usually)



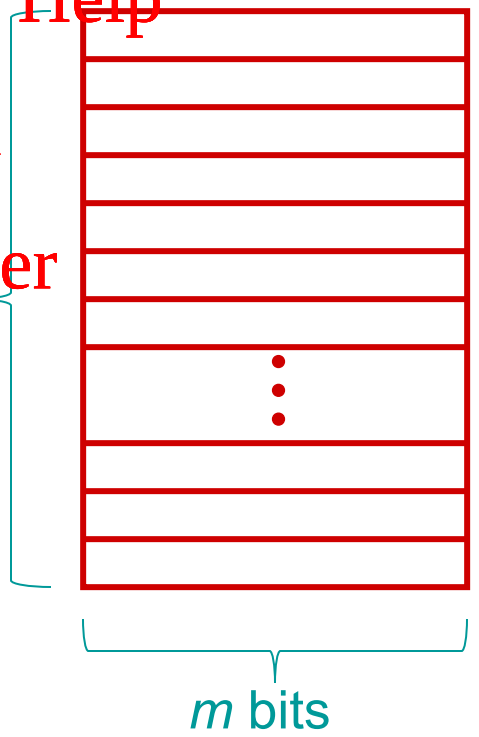
NAND FLASH
(on or off-chip, non-volatile)

Memory

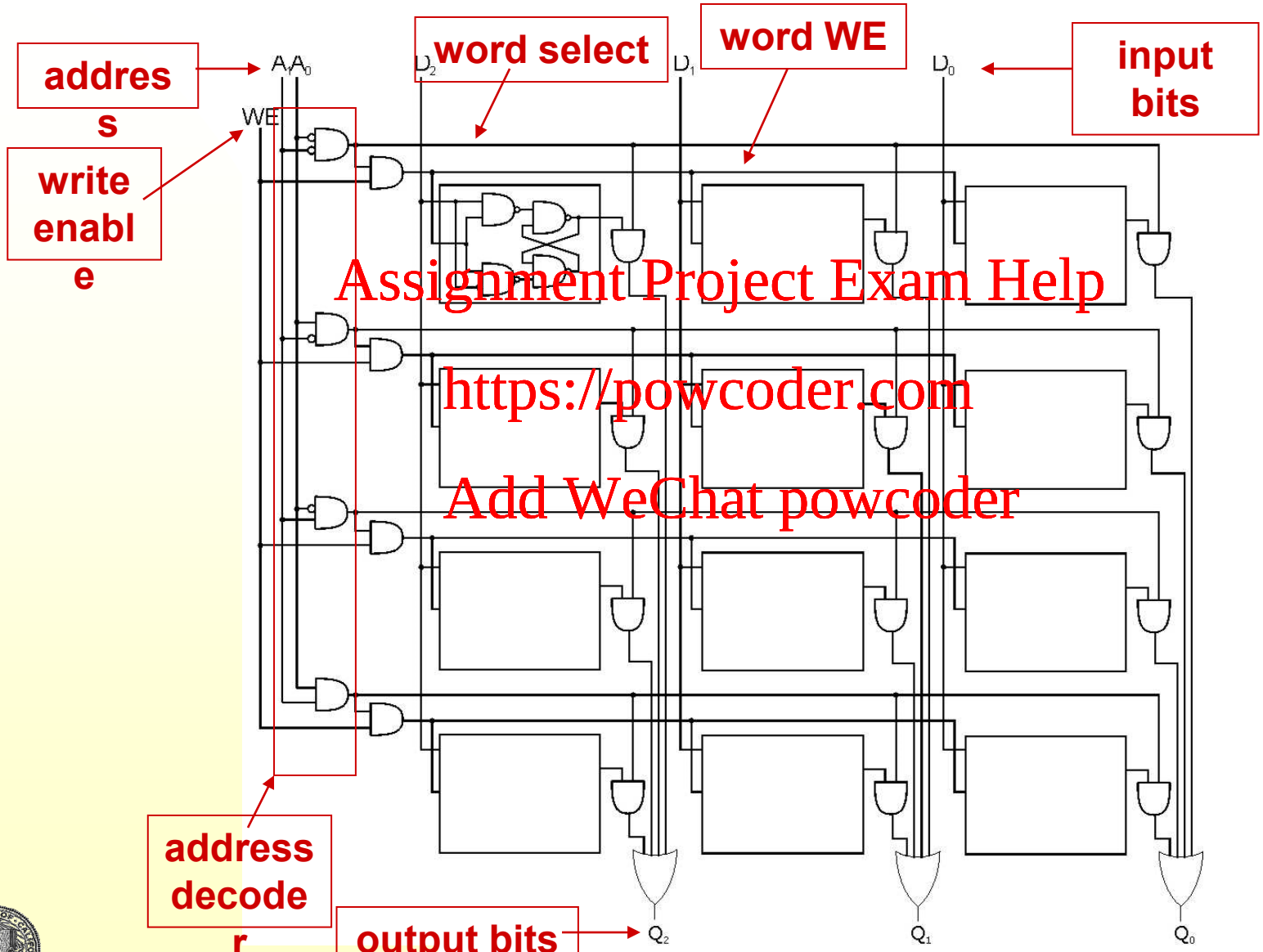
Now that we know how to store bits,
we can build a memory – a logical $k \times m$ array of stored bits.

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

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

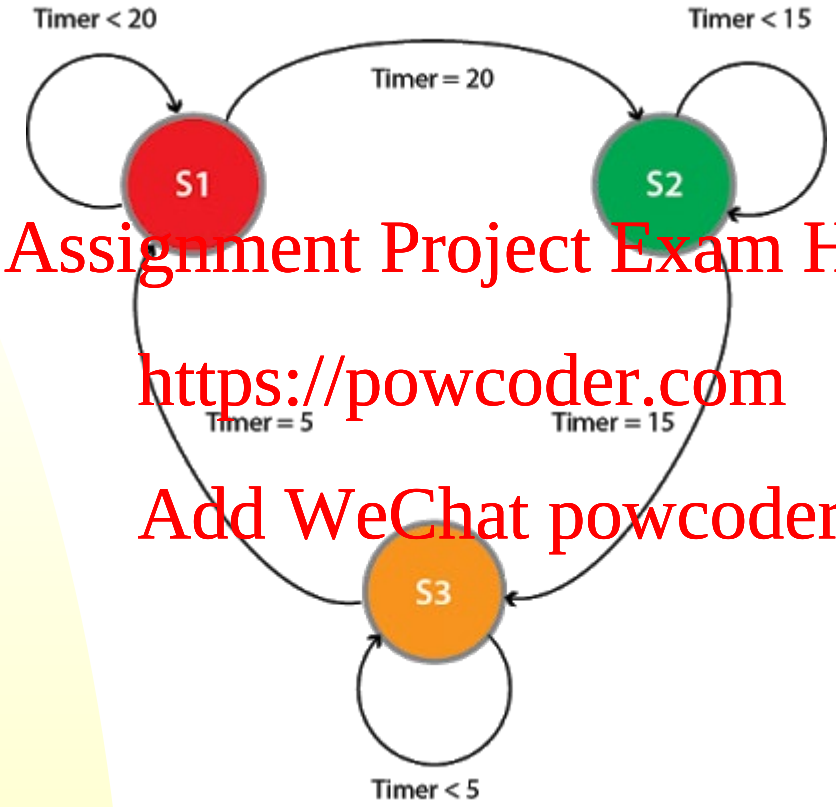


2^2 x 3 Memory

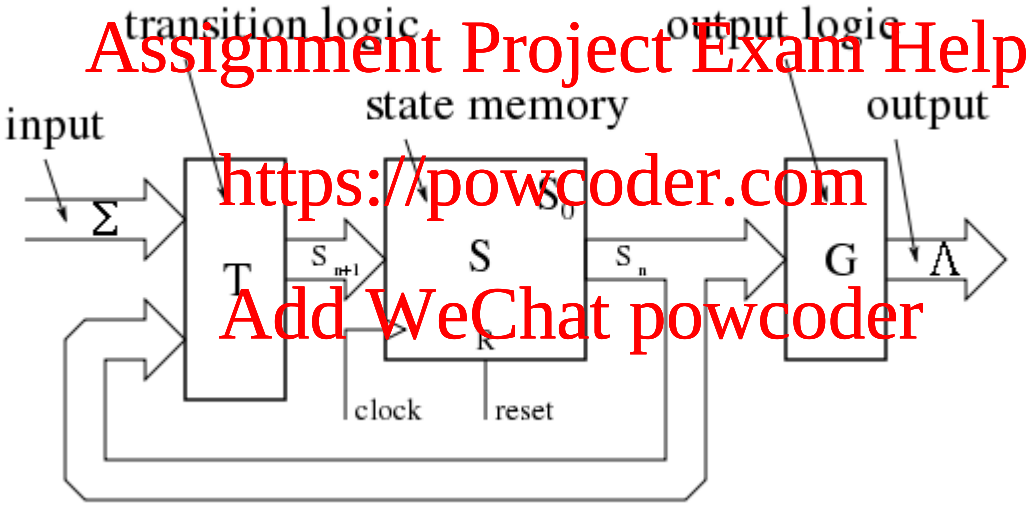


Let's Build a Computer

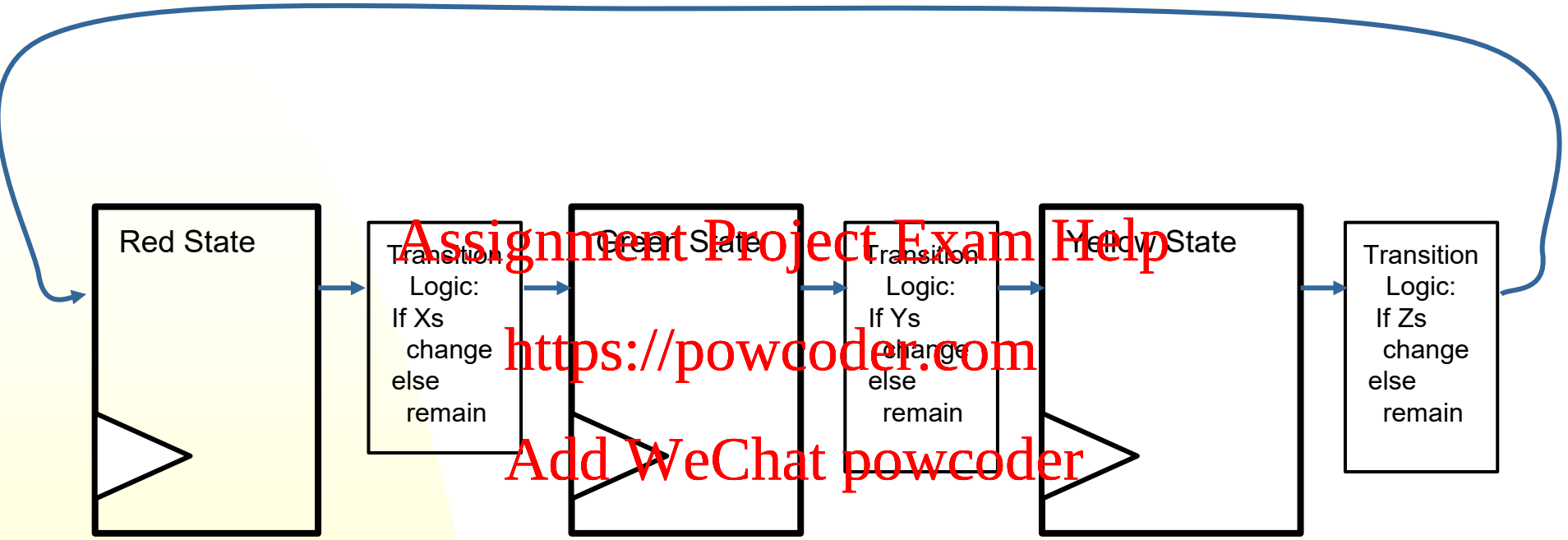
Example of State Machine: Traffic Light



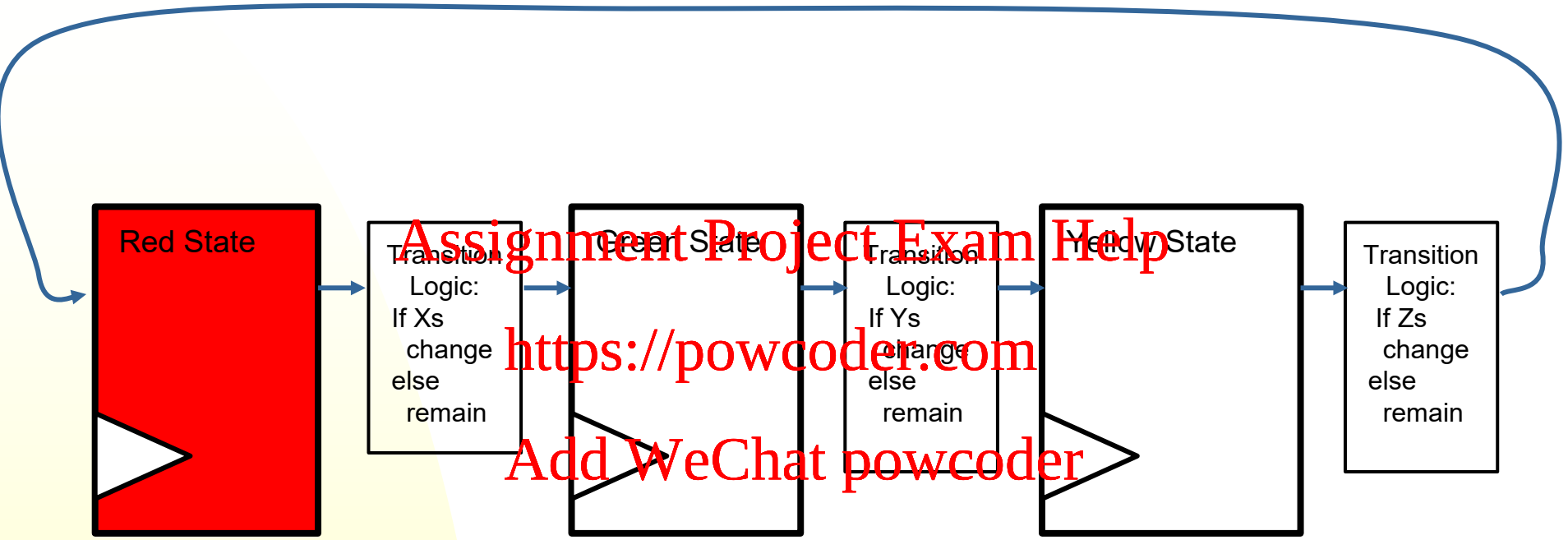
State Machine: Overview



State Machine: Traffic Light



State Machine: Traffic Light

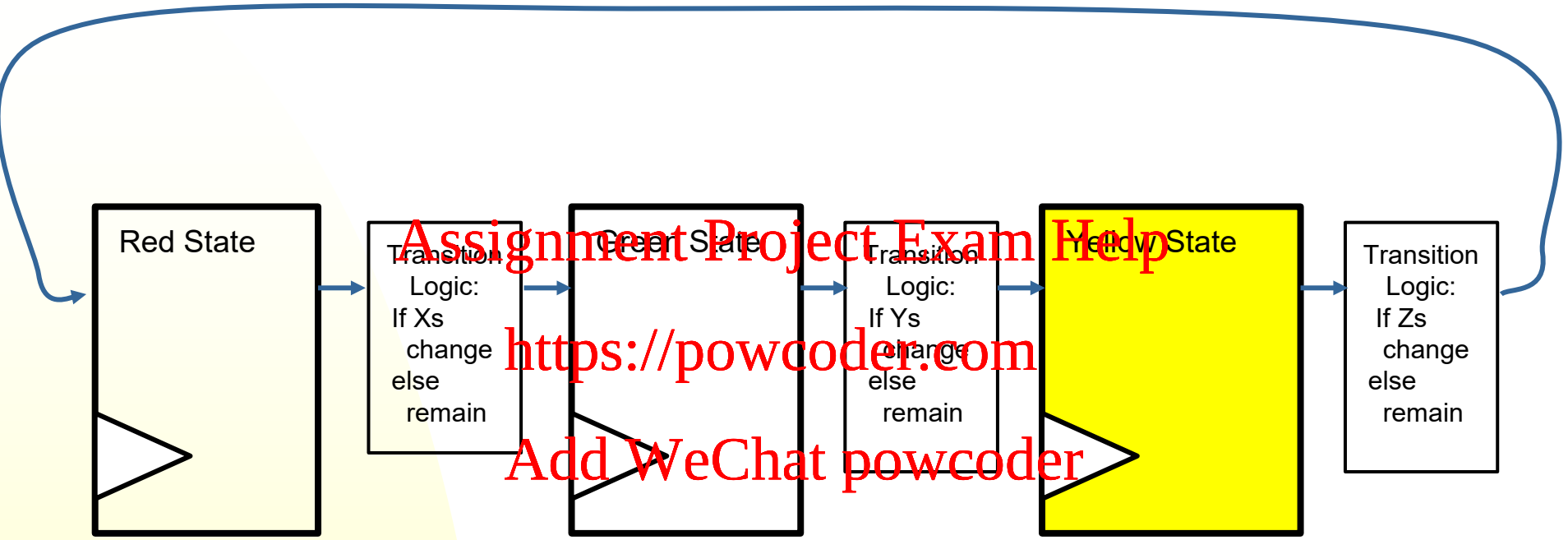


- Start off in default state Red

State Machine: Traffic Light

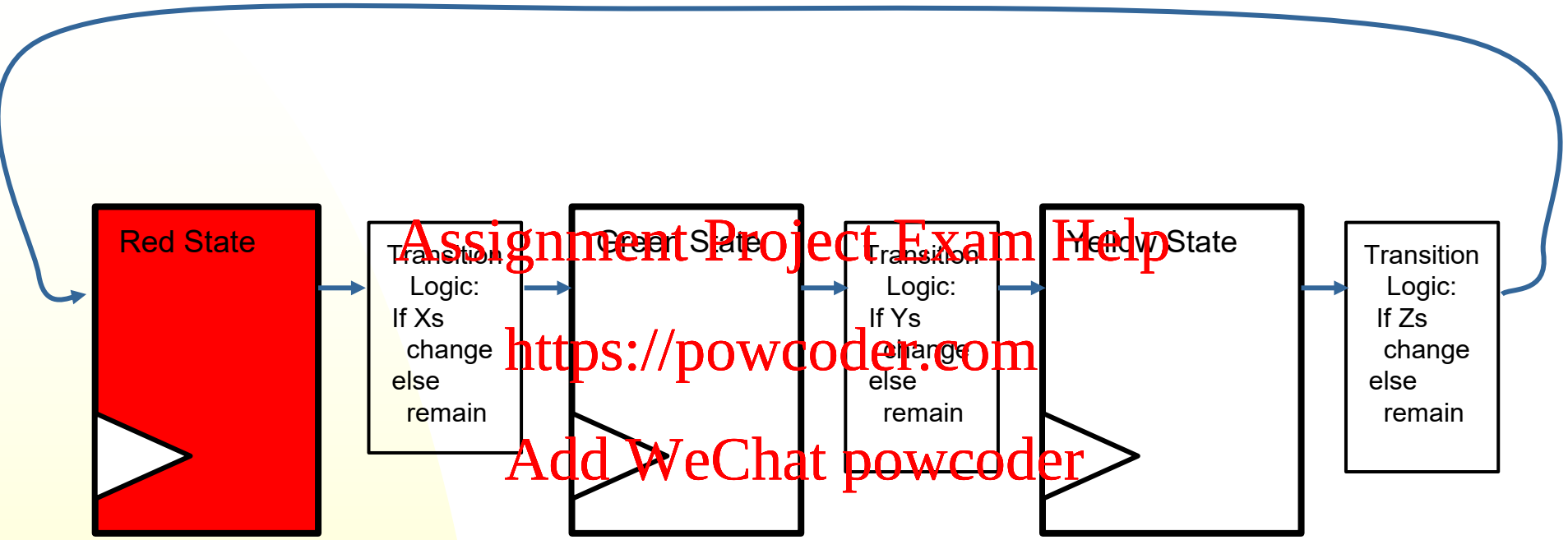
- After a specified time Xs switch to next state

State Machine: Traffic Light



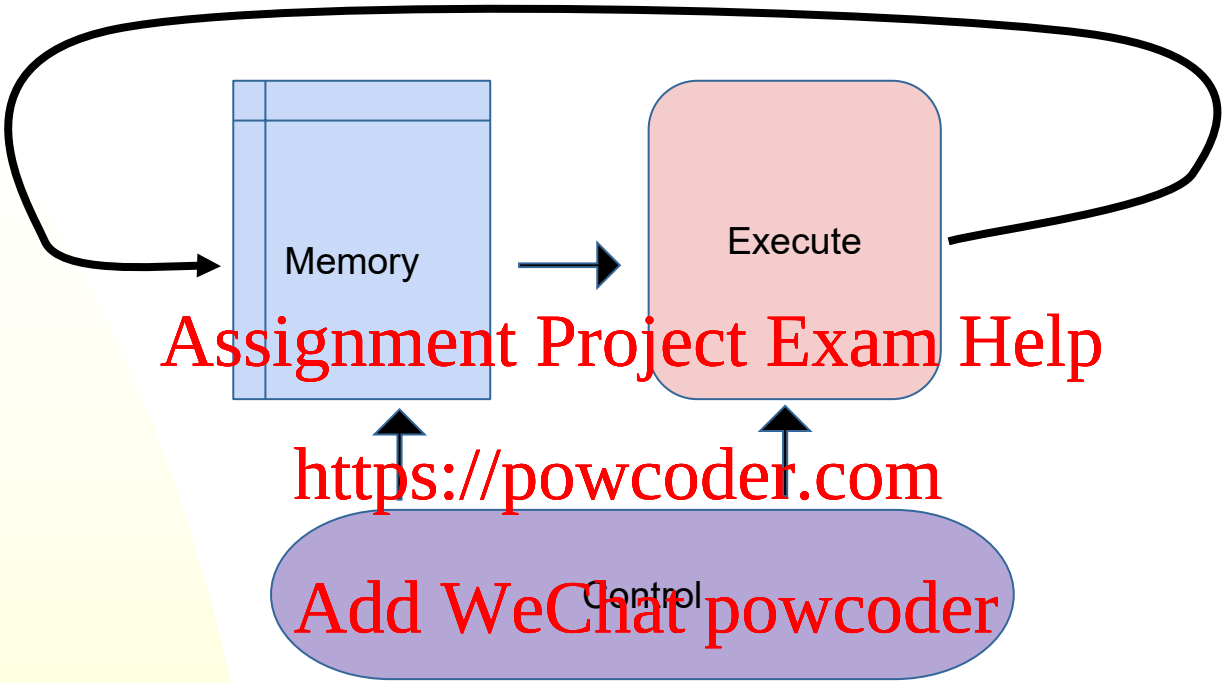
- After a specified time Ys switch to next state

State Machine: Traffic Light



- After a specified time Zs switch to next state which is Red.

Basic Computer



Memory: Could be Flip Flops, SRAM/DRAM, Flash etc
Execute: Combinational Logic (Adder, Shifter, Rotation etc)
Control: Finite State Machine (combination of sequential and combinational logic circuits)

Data Path

Combinational Logic

Storage

State Machine