Digital Logic Design + Computer Architecture

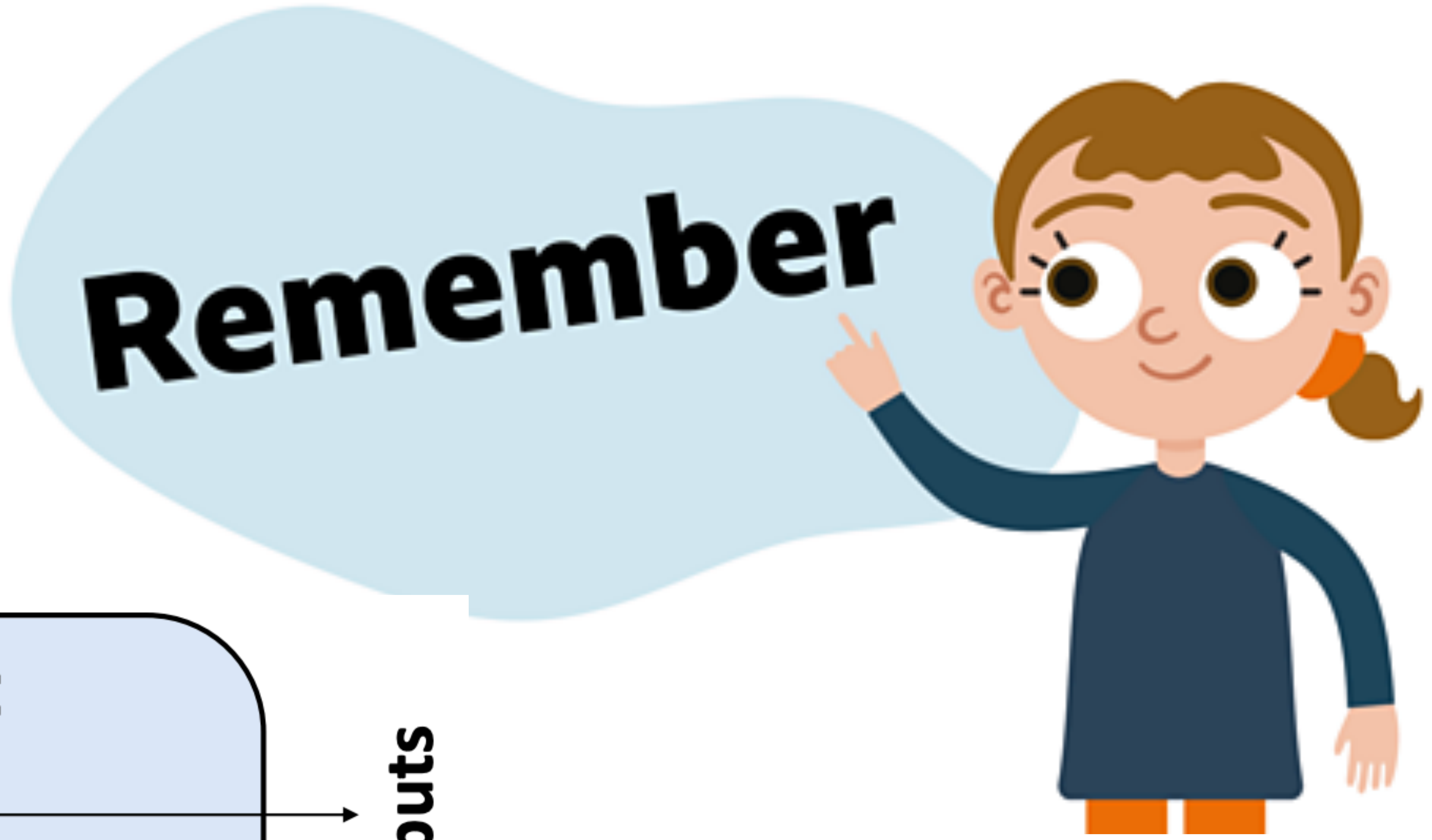
Sayandeep Saha

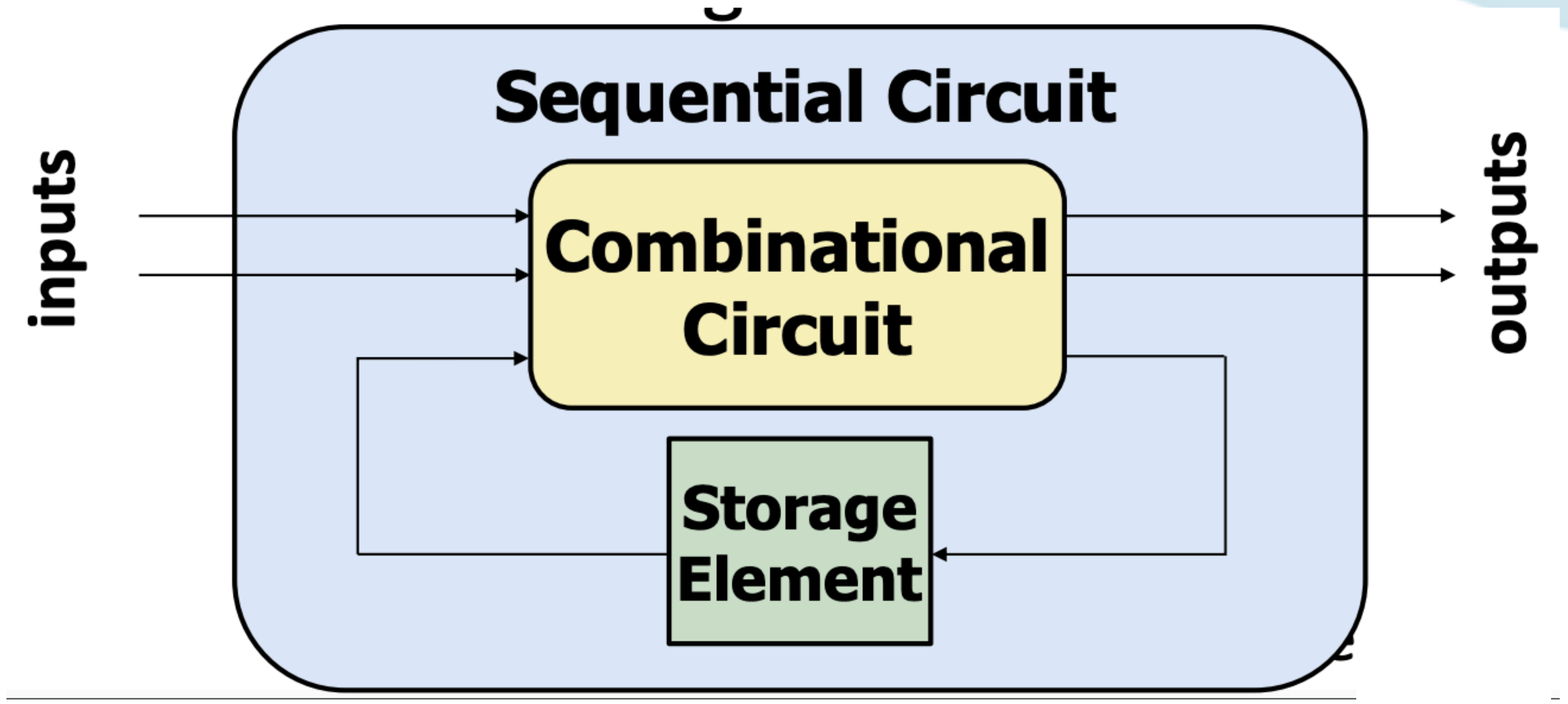
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Sequential Circuits

A Circuit that Remembers

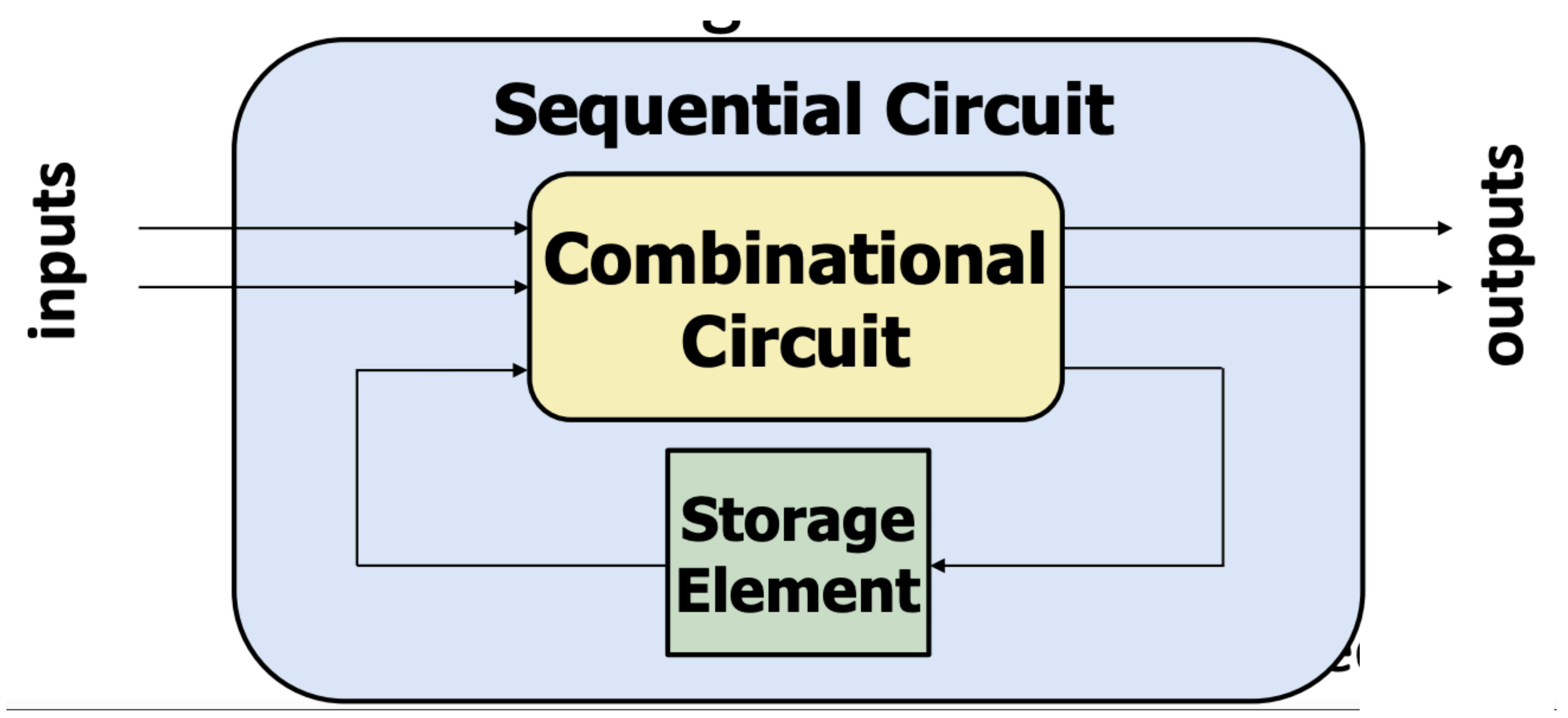
- **How do you remember things?**
  - Memory
- **Can we design a circuit which remembers?**
  - A formal way to model this capability is called a **state**
  - **So we will be modelling circuits to create a state.**





A Circuit that Remembers

- **Every digital logic you see in real life is sequential**
  - Your processors — that you going to see in the rest of the course
  - Your washing machine — it remembers your setting and washes accordingly
  - Your elevator — it remembers which floors to stop
  - Your ATM machine — it remembers your choice and updates your account after despatching money

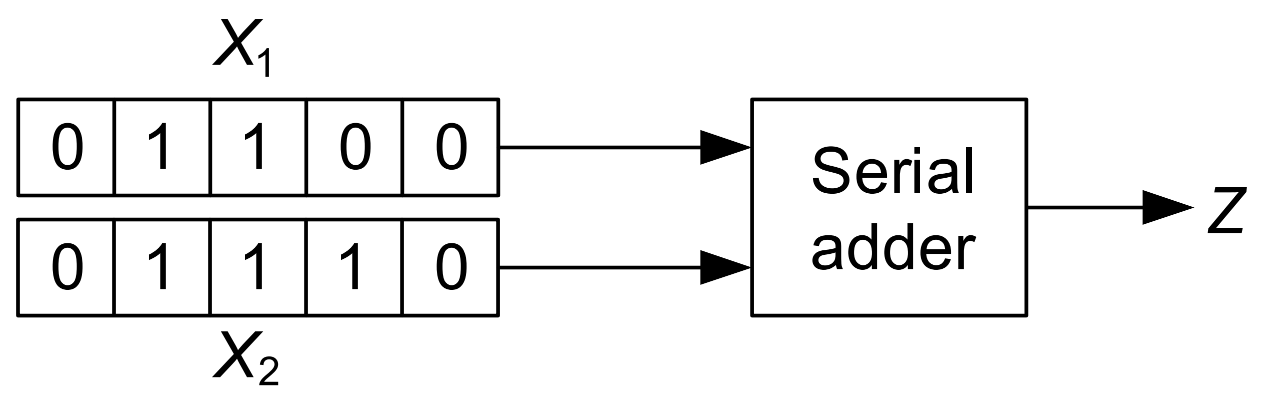


Sequential Circuits and Finite State Machines

Sequential circuit: its outputs a function of external inputs as well as stored information (aka. State)

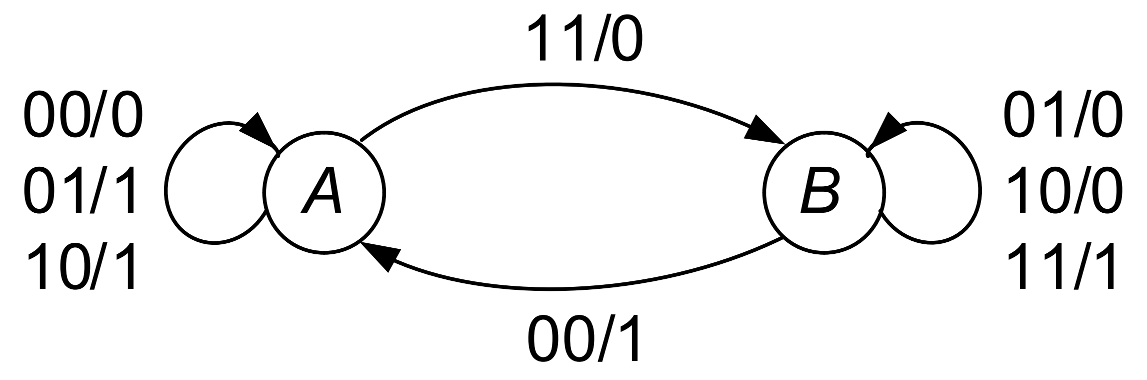
Finite-state machine (FSM): abstract model to describe the synchronous sequential machines. It has finite memory.

Serial binary adder example: block diagram, addition process, state table and state diagram
Let A denote the state of the adder at t_i if the carry 0 is generated at t_{i-1}
Let B denote the state of the adder at t_i if the carry 1 is generated at t_{i-1}



$$\begin{array}{rrrrrrcl}
 & t_5 & t_4 & t_3 & t_2 & t_1 & & \\
 & 0 & 1 & 1 & 0 & 0 & = & X_1 \\
+ & 0 & 1 & 1 & 1 & 0 & = & X_2 \\
\hline
 & 1 & 1 & 0 & 1 & 0 & = & Z
\end{array}$$

PS	NS, z			
	$x_1x_2 = 00$	01	11	10
A	$A, 0$	$A, 1$	$B, 0$	$A, 1$
B	$A, 1$	$B, 0$	$B, 1$	$B, 0$



Sequential Circuits and Finite State Machines

Two states capable of storing information regarding the **presence or absence of carry**:

delay element with input *Y* and output *y*

- Two states: $y = 0$ and $y = 1$
- The capability of the device to store information is the result of the fact that it takes some time for the input signal Y to pass to the output y
  - Compare with a combinational gate where the output changes almost immediately. In this device there is some well-defined time required before the input Y passes to the output y. **During that time-window, the old value stays!**
- Since the **present input value *Y*** of the delay element is equal to its **next output value**: the input value is referred to as the next state of the delay
  - $y(t+1)=Y(t)$

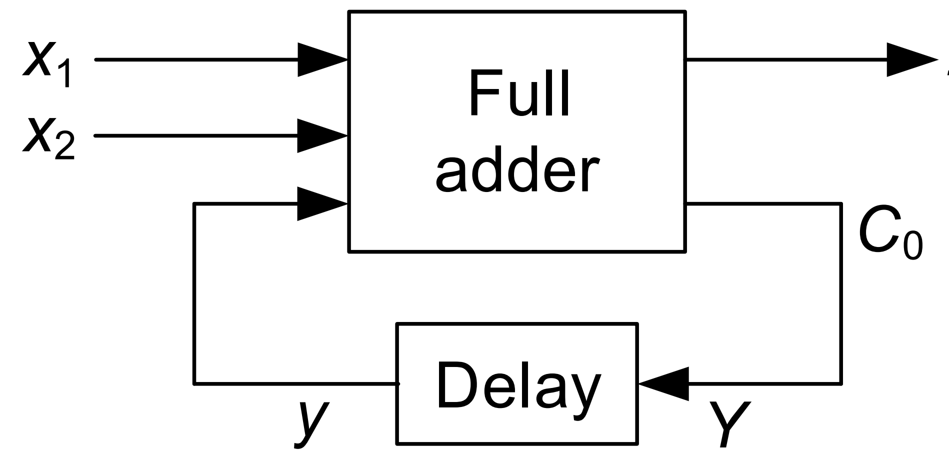
Example: assign state $y = 0$ to state *A* of the adder and $y = 1$ to *B*

- The value of y at t_i corresponds to the value of the carry generated at t_{i-1}
- Process of assigning the states of a physical device to the states of the serial adder: called **state assignment**
- Output value y: referred to as the **state variable**
- **Transition/output table** for the serial adder:

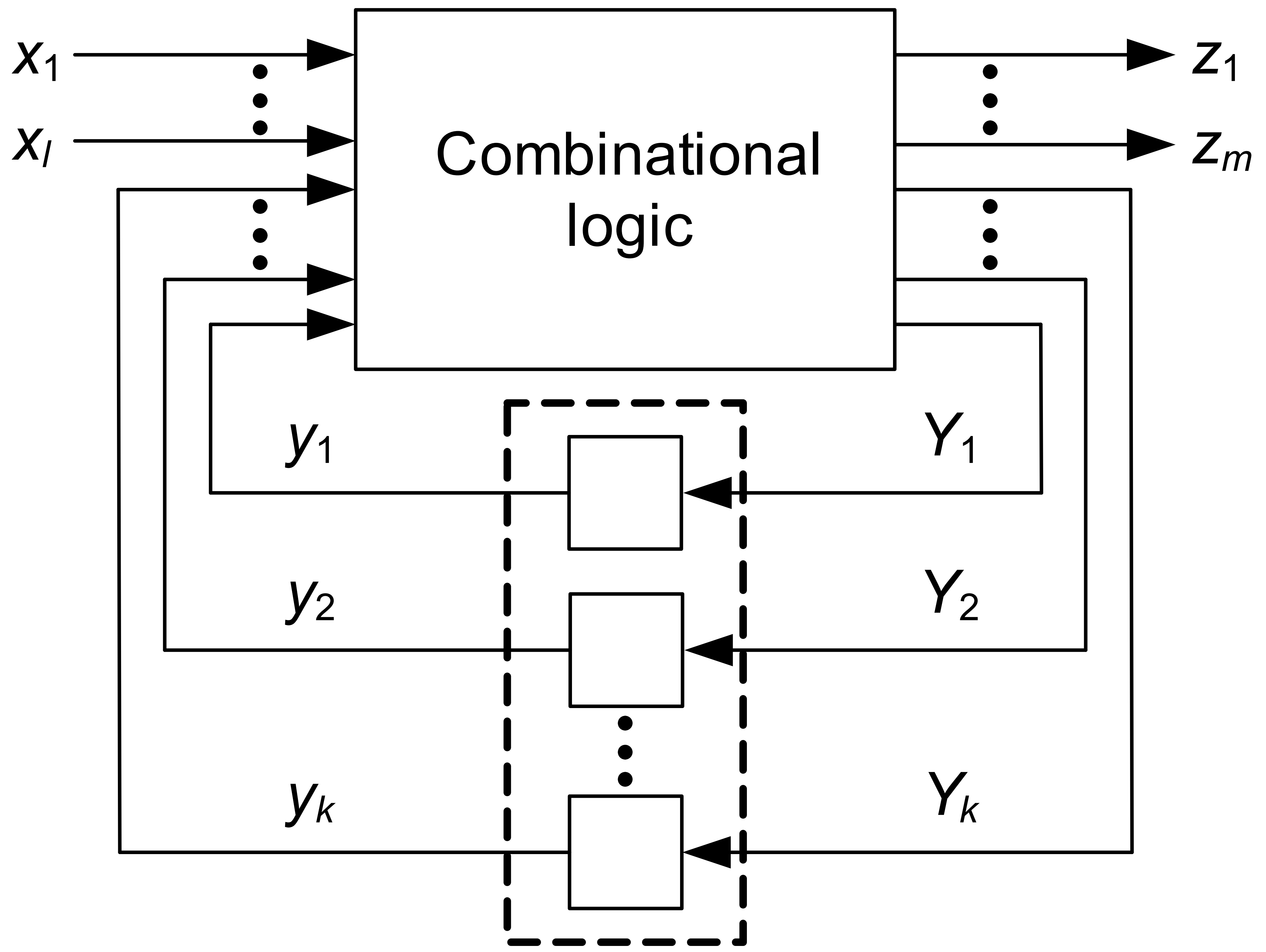
y	Next state Y x_1x_2 00	01	11	10	Output z x_1x_2 00	01	11	10
0	0	0	1	0	0	1	0	1
1	0	1	1	1	1	0	1	0

$$Y = x_1x_2 + x_1y + x_2y$$

$$z = x_1 \oplus x_2 \oplus y$$



Sequential Circuits and Finite State Machines

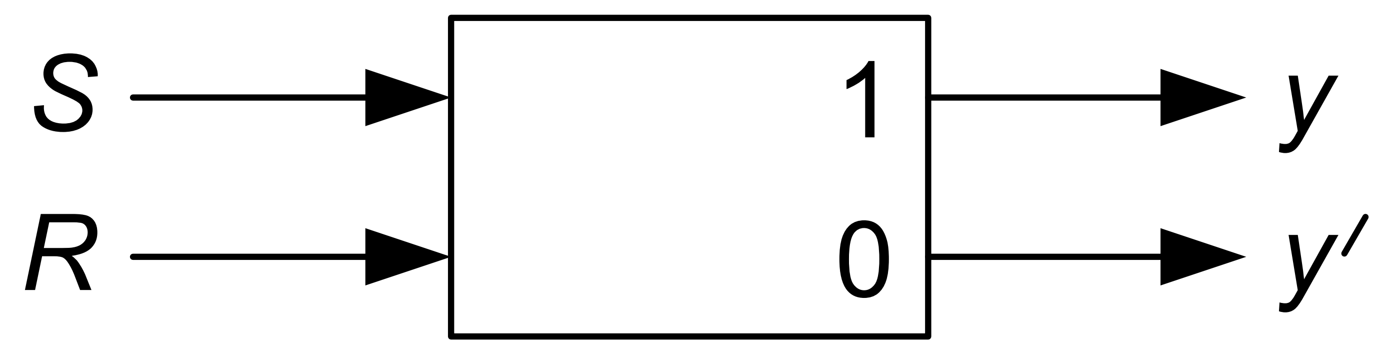


``Memory'' devices

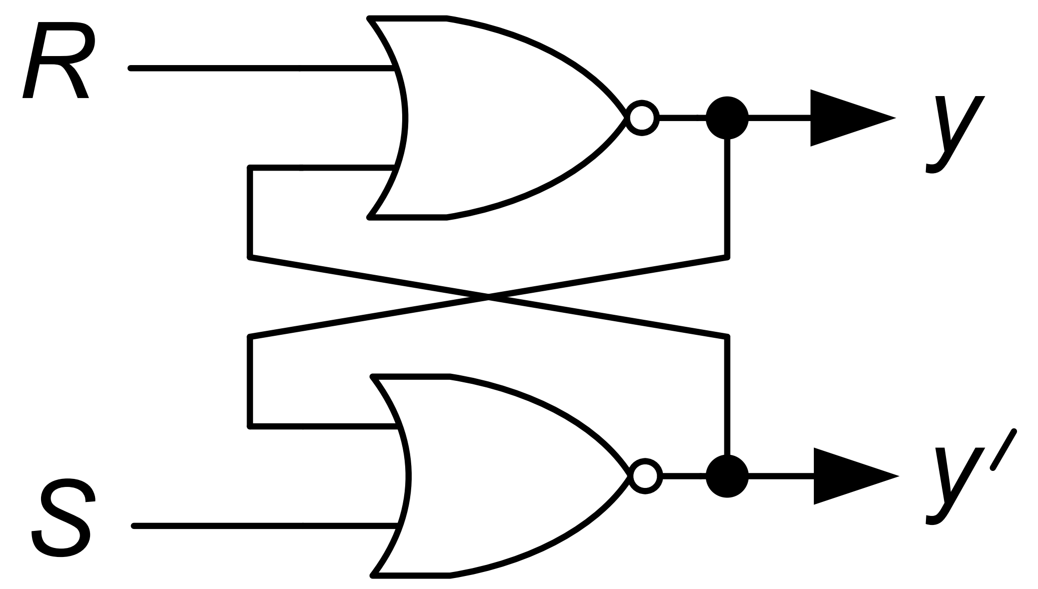
Memory Element: Latches

Latch: remains in one state indefinitely until an input signals directs it to do otherwise

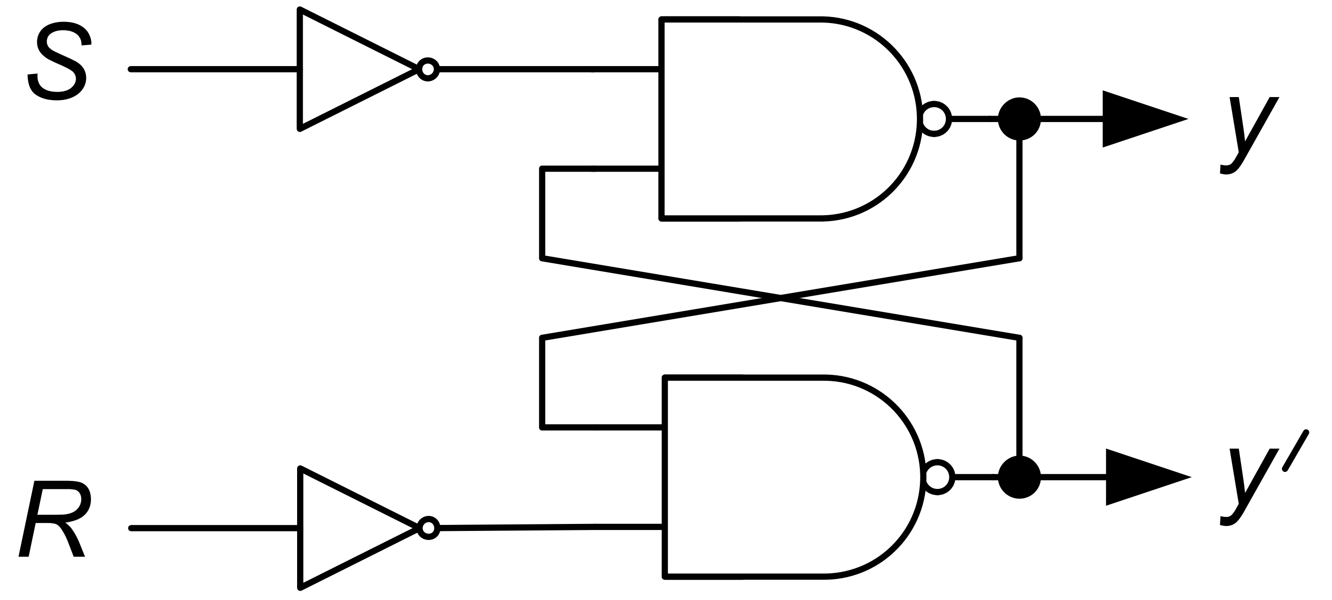
Set-reset of *SR* latch:



(*a*) Block diagram.



(*b*) NOR latch.



(*c*) NAND latch.

Memory Element: Latches

Characteristic table and excitation requirements:

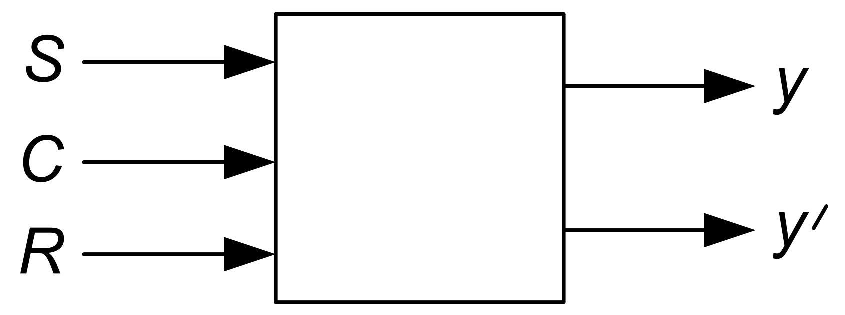
$y(t)$	$S(t)$	$R(t)$	$y(t+1)$
0	0	0	0
0	0	1	0
0	1	1	?
0	1	0	1
1	1	0	1
1	1	1	?
1	0	1	0
1	0	0	1

$$\begin{aligned} RS &= 0 \\ y(t+1) &= R'y(t) + S \end{aligned}$$

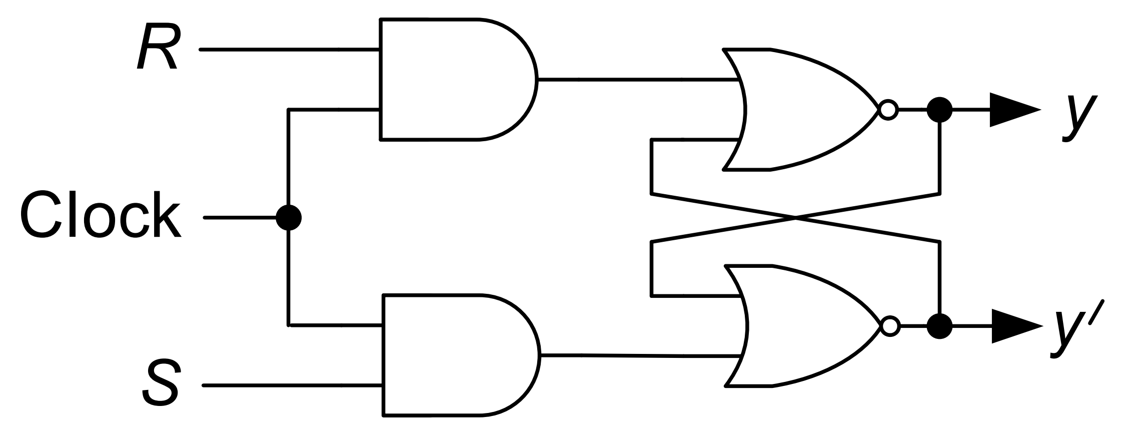
Circuit change		*Required value*	
From:	*To:*	S	R
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0

Clocked *SR* latch: all state changes synchronized to clock pulses

- Restrictions placed on the length and frequency of clock pulses: so that the circuit changes state no more than once for each clock pulse



(*a*) Block diagram.



(*b*) Logic diagram.

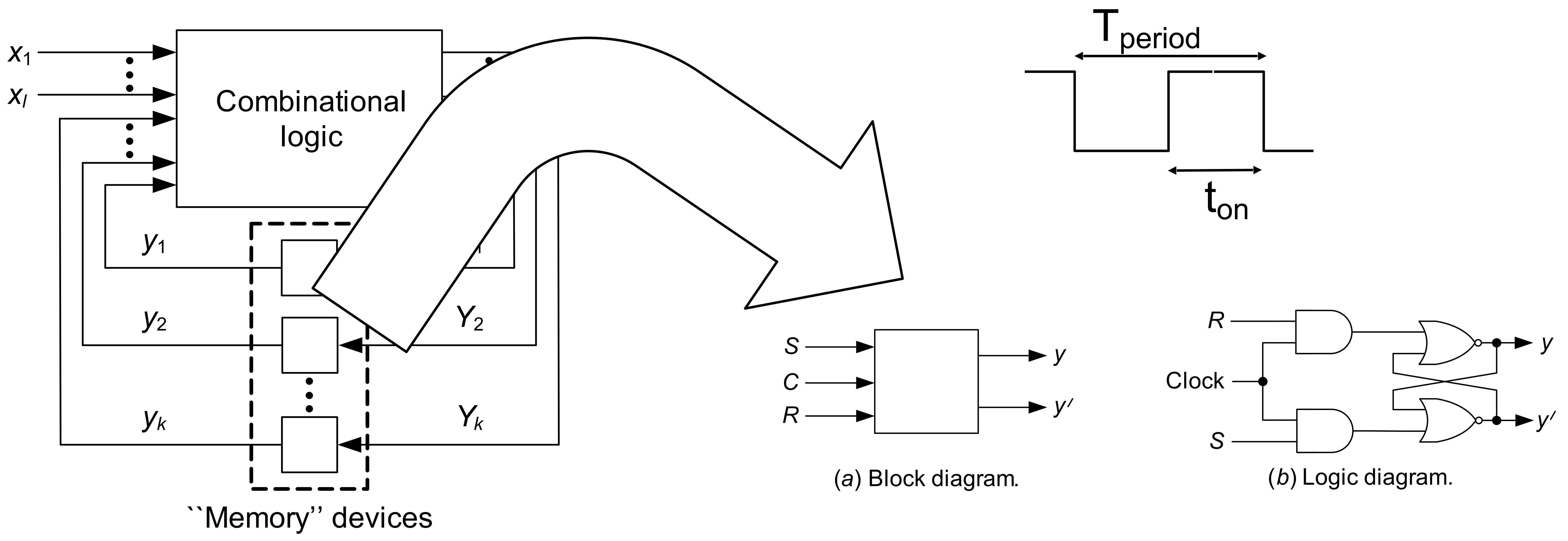
Memory Element: Latches

Why is the (1,1) input forbidden?

$y(t)$	$S(t)$	$R(t)$	$y(t+1)$
0	0	0	0
0	0	1	0
0	1	1	?
0	1	0	1
1	1	0	1
1	1	1	?
1	0	1	0
1	0	0	1

$$\begin{aligned} RS &= 0 \\ y(t+1) &= R'y(t) + S \end{aligned}$$

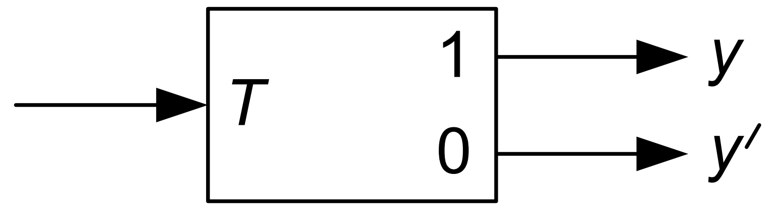
Memory Element: Latches



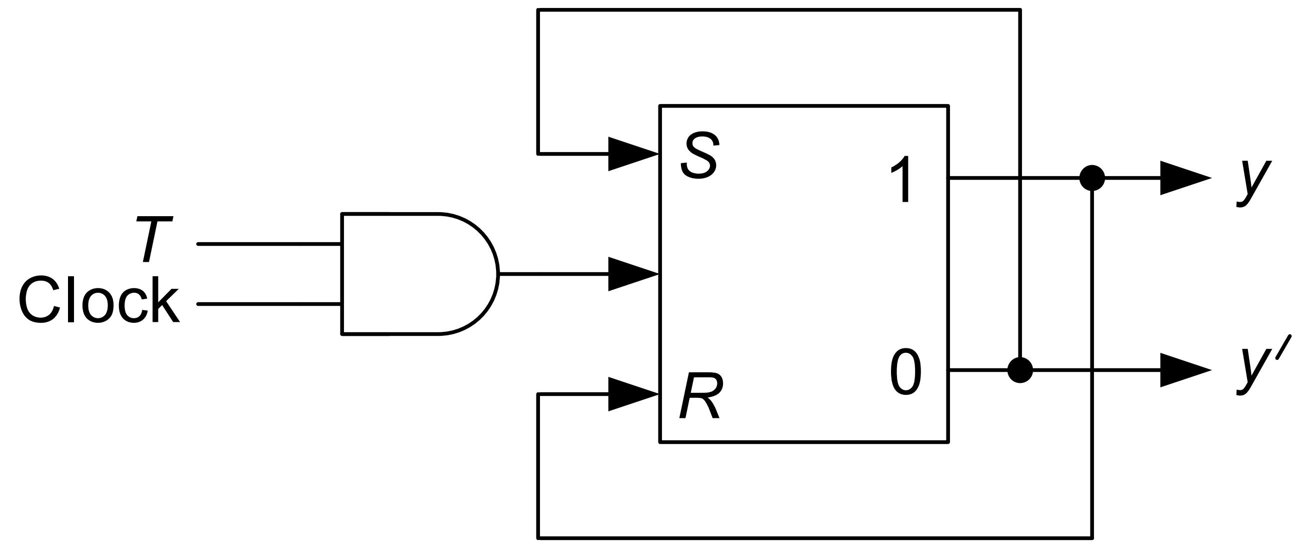
- A **clock** is a periodic signal that is used to keep time in sequential circuits.
- **Duty Cycle** is the ration of t_{on}/T_{period}
- We want to keep t_{on} small so that in the same clock pulse only a single computation is performed.
- We want to keep T_{period} sufficient so that there is enough time for the next input to be computed.

Memory Element: T Latch

Value 1 applied to its input triggers the latch to change state



(*a*) Block diagram.



(*b*) Deriving the T latch from the clocked SR latch.

Excitations requirements:

Circuit change *From:*	*To:*	*Required value T*
0	0	0
0	1	1
1	0	1
1	1	0

"Q" is basically "y"

Characteristic Table

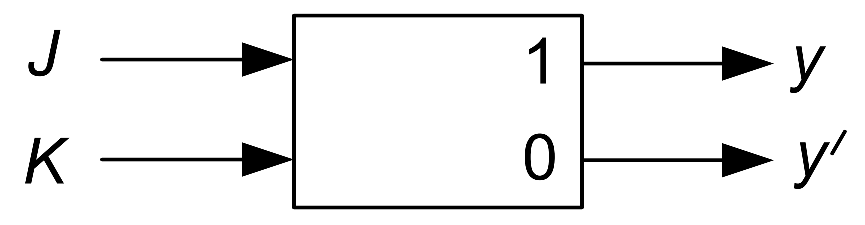
***T* Flip-Flop**

T	**Q(*t* + 1)**	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

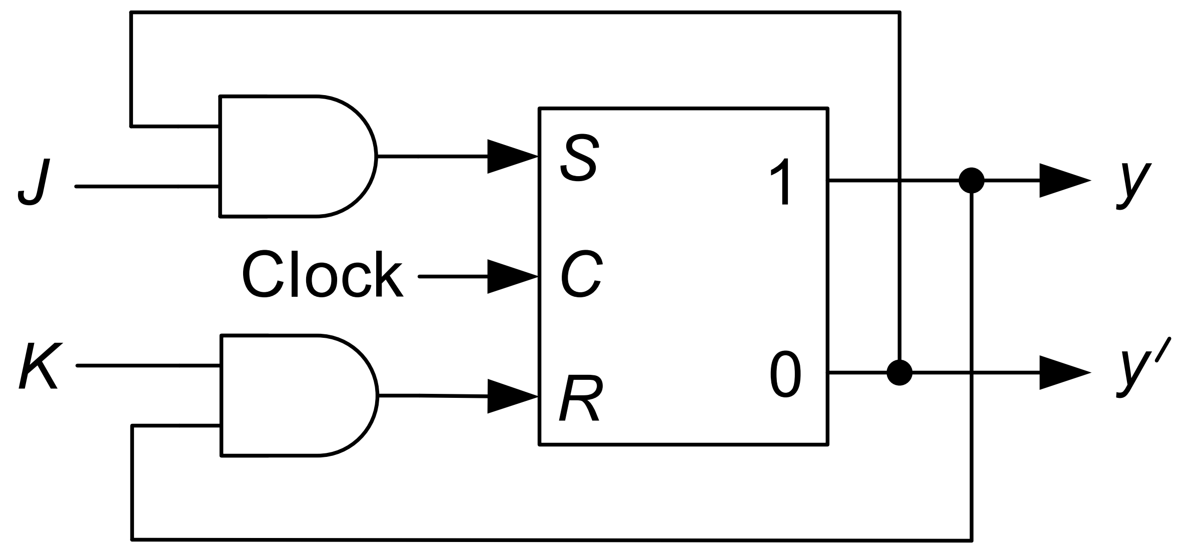
$$y(t+1) = Ty'(t) + T'y(t)$$
$$= T \oplus y(t)$$

Memory Element: JK Latch

Unlike the *SR* latch, *J* = *K* = 1 is permitted: when it occurs, the latch acts like a trigger and switches to the complement state



(*a*) Block diagram.



(*b*) Constructing the JK latch from the clocked SR latch.

Excitation requirements:

Circuit change		*Required value*	
From:	*To:*	*J*	*K*
0	0	0	–
0	1	1	–
1	0	–	1
1	1	–	0

"Q" is basically "y"

Characteristic Table

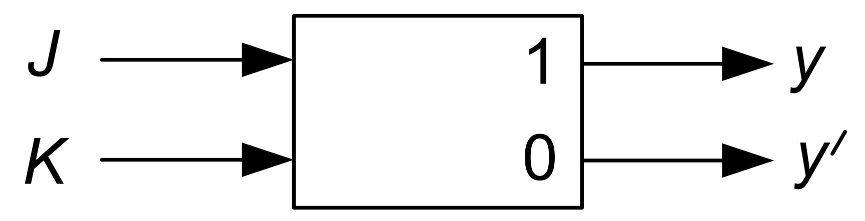
JK Flip-Flop

J	**K**	**Q(t + 1)**	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

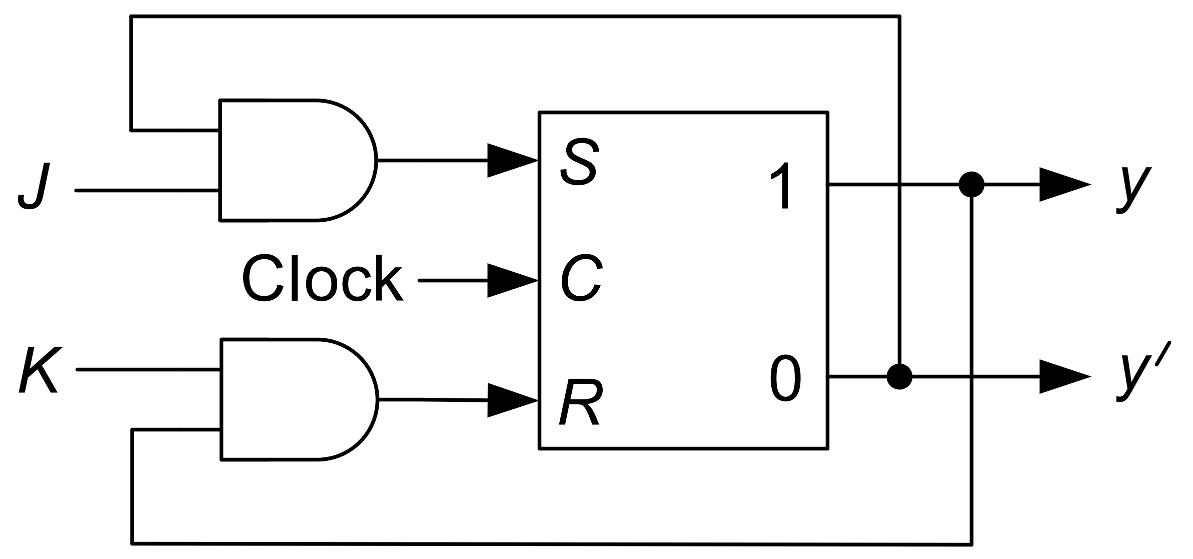
Can you write the characteristic equation?

Memory Element: JK Latch

Unlike the *SR* latch, *J* = *K* = 1 is permitted: when it occurs, the latch acts like a trigger and switches to the complement state



(*a*) Block diagram.



(*b*) Constructing the JK latch from the clocked SR latch.

Excitation requirements:

Circuit change		*Required value*	
From:	*To:*	*J*	*K*
0	0	0	–
0	1	1	–
1	0	–	1
1	1	–	0

"Q" is basically "y"

Characteristic Table

JK Flip-Flop

J	*K*	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

Can you write the characteristic equation?

$$y(t+1) = Jy(t)' + K'y(t)$$

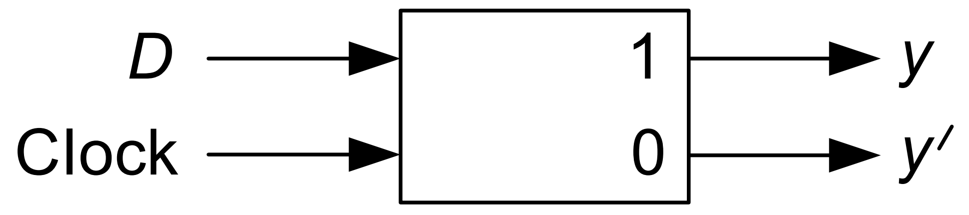
D Latch — The Latch of Your Life

The next state of the *D* latch is equal to its present excitation:

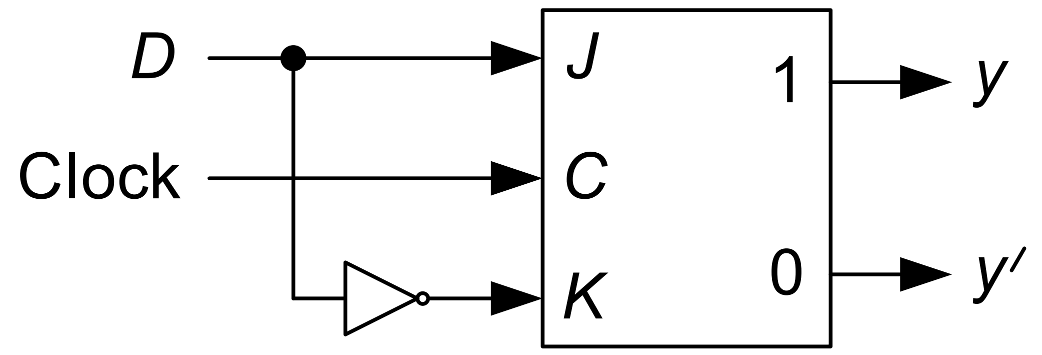
$$y(t+1) = D(t)$$

D Flip-Flop

D	$Q(t+1)$	
0	0	Reset
1	1	Set



(*a*) Block diagram.



(*b*) Transforming the JK latch to the D latch.

Excitation Table

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

How is Your Clock?

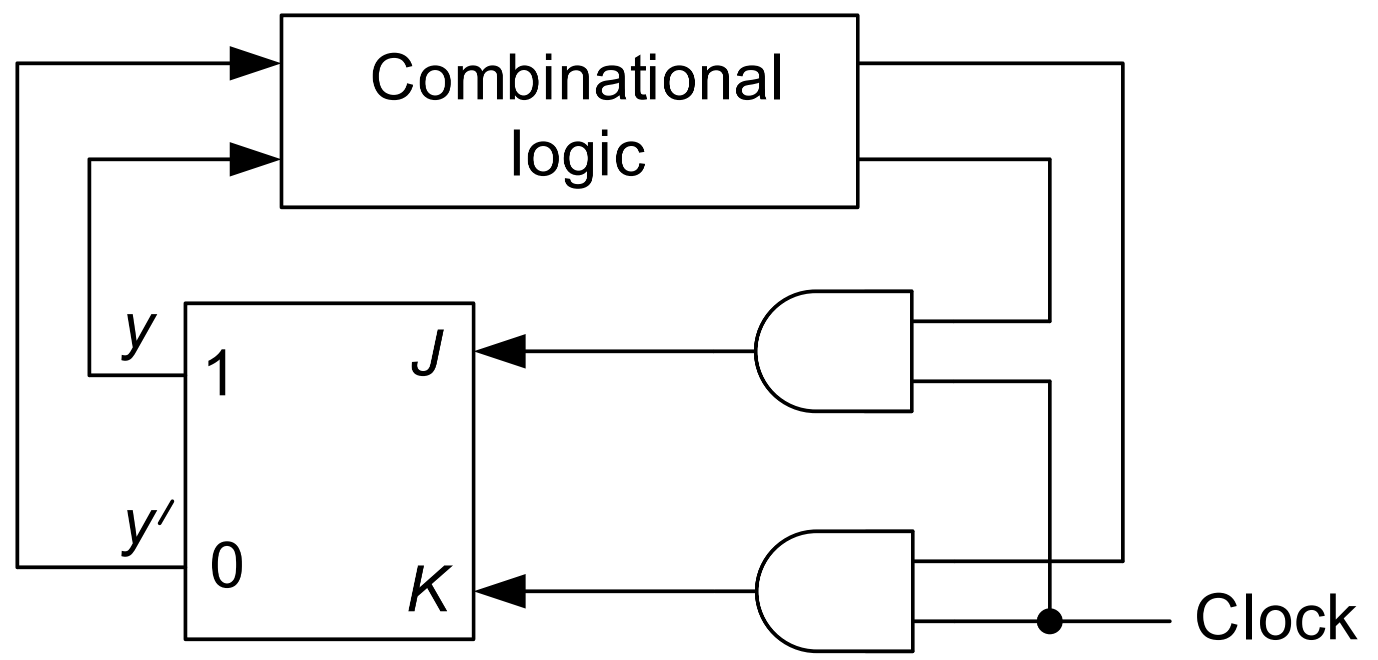
Clocked latch: changes state only in synchronization with the clock pulse and no more than once during each occurrence of the clock pulse

Duration of clock pulse: determined by circuit delays and signal propagation time through the latches

- Must be long enough to allow latch to change state, and
- Short enough so that the latch will not change state twice due to the same excitation

Excitation of a *JK* latch within a sequential circuit:

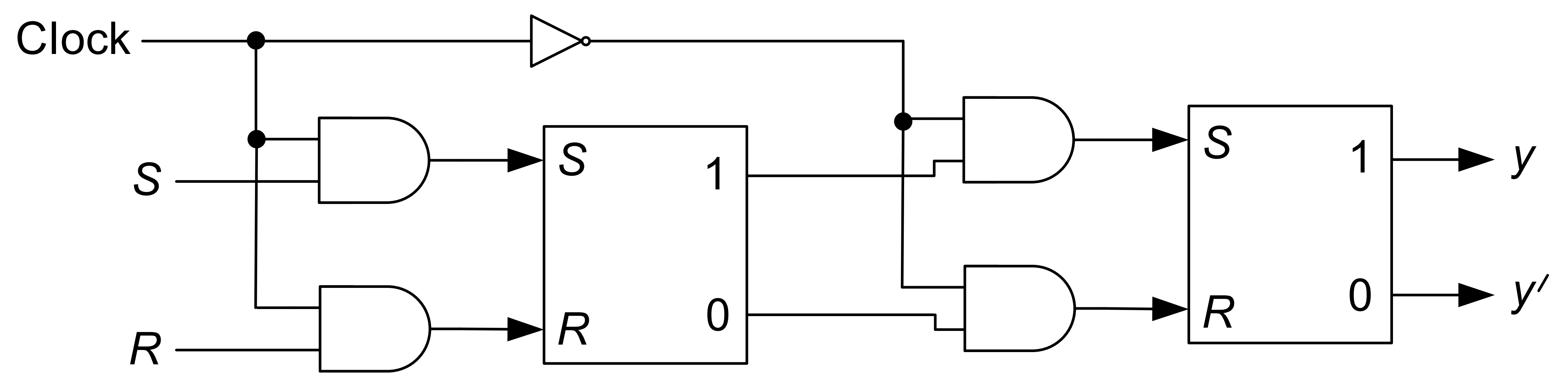
- Length of the clock pulse must allow the latch to generate the *y*'s
- But should not be present when the values of the *y*'s have propagated through the combinational circuit



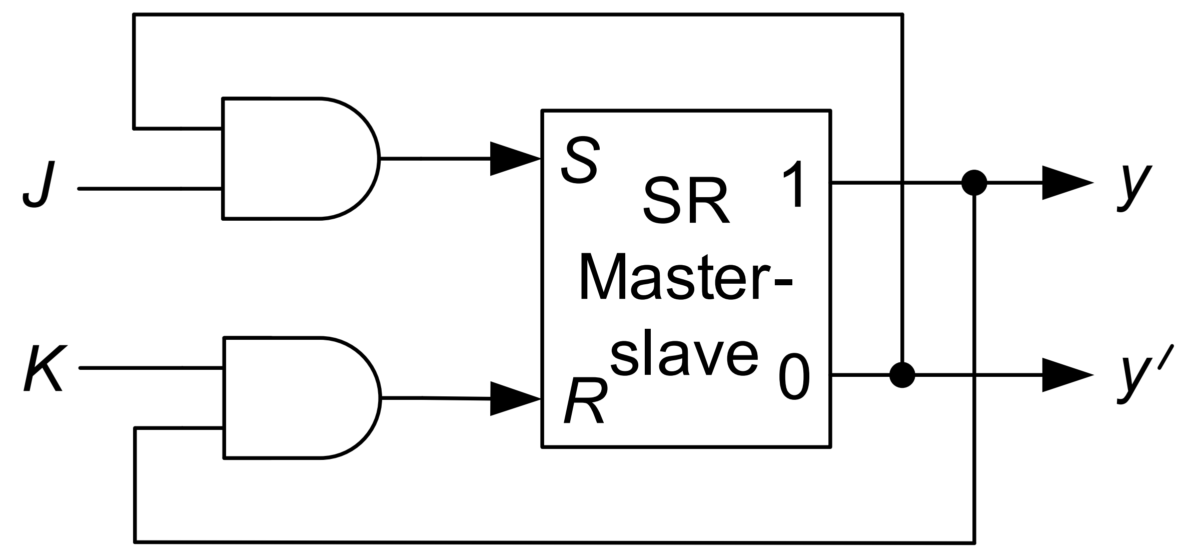
Master Slave Flip-Flop

Master-slave flip-flop: a type of synchronous memory element that eliminates the timing problems by isolating its inputs from its outputs

Master-slave *SR* flip-flop:



Master-slave *JK* flip-flop: since master-slave *SR* flip-flop suffers from the problem that both its inputs cannot be 1, it can be converted to a *JK* flip-flip



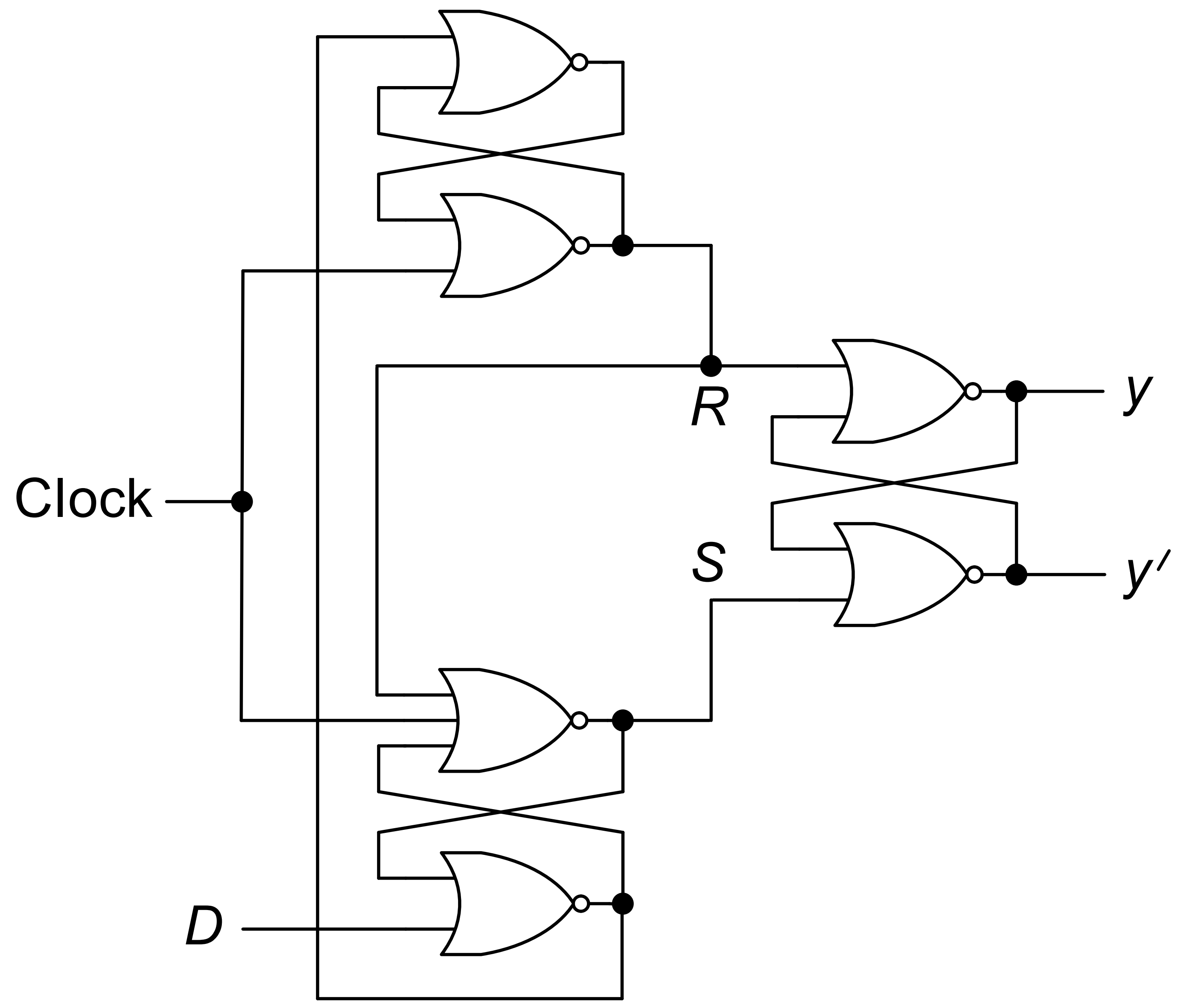
Edge Triggered Flip-Flop

Positive (negative) edge-triggered *D* flip-flip: stores the value at the *D* input when the clock makes a 0 -> 1 (1 -> 0) transition

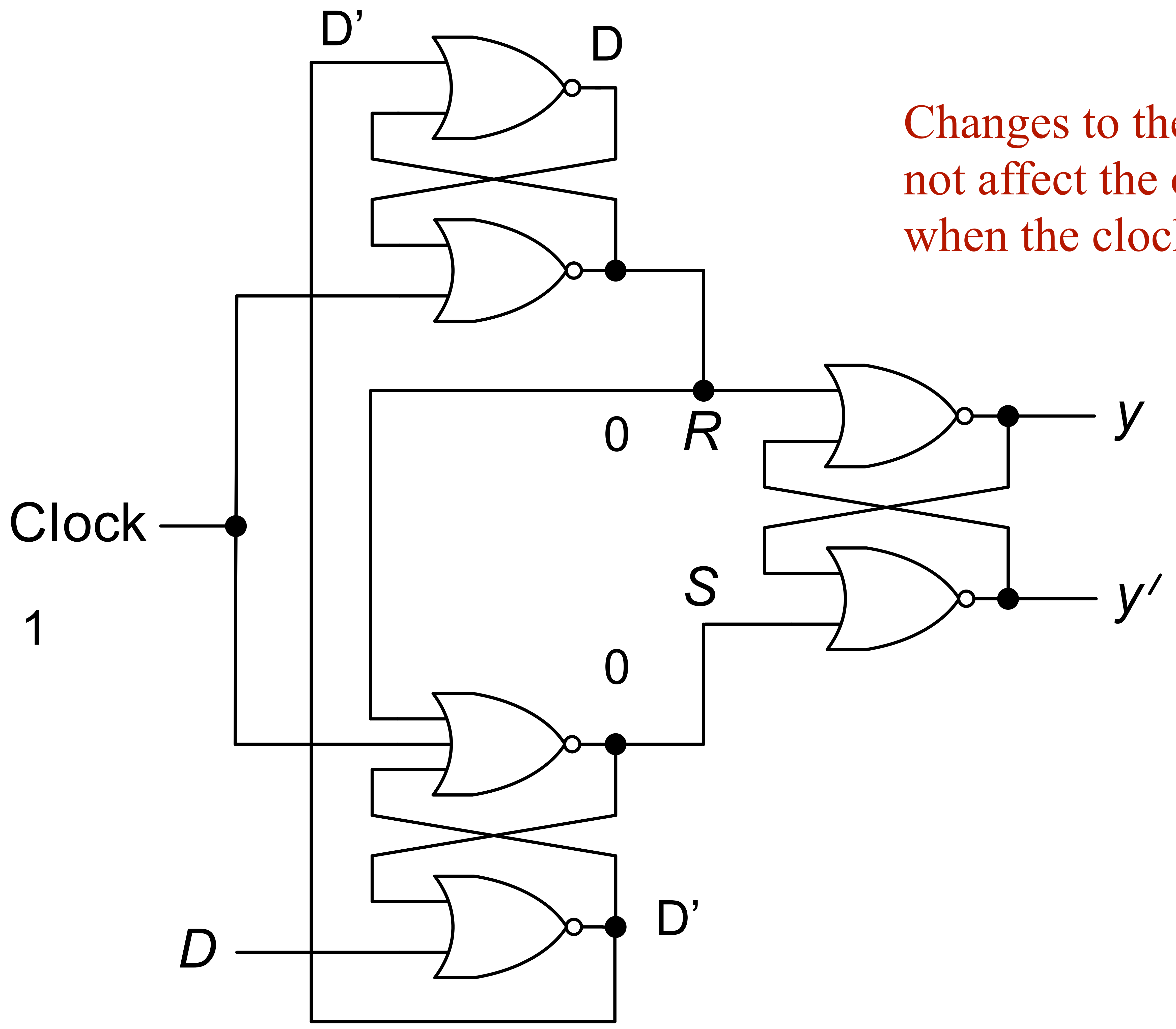
- Any change at the *D* input after the clock has made a transition does not have any effect on the value stored in the flip-flop

A negative edge-triggered *D* flip-flop:

- When the clock is high, the output of the bottommost (topmost) NOR gate is at *D'* (*D*), whereas the *S*-*R* inputs of the output latch are at 0, causing it to hold previous value
- When the clock goes low, the value from the bottommost (topmost) NOR gate gets transferred as *D* (*D'*) to the *S* (*R*) input of the output latch
  - Thus, output latch stores the value of *D*
- If there is a change in the value of the *D* input after the clock has made its transition, the bottommost NOR gate attains value 0
  - However, this cannot change the *SR* inputs of the output latch

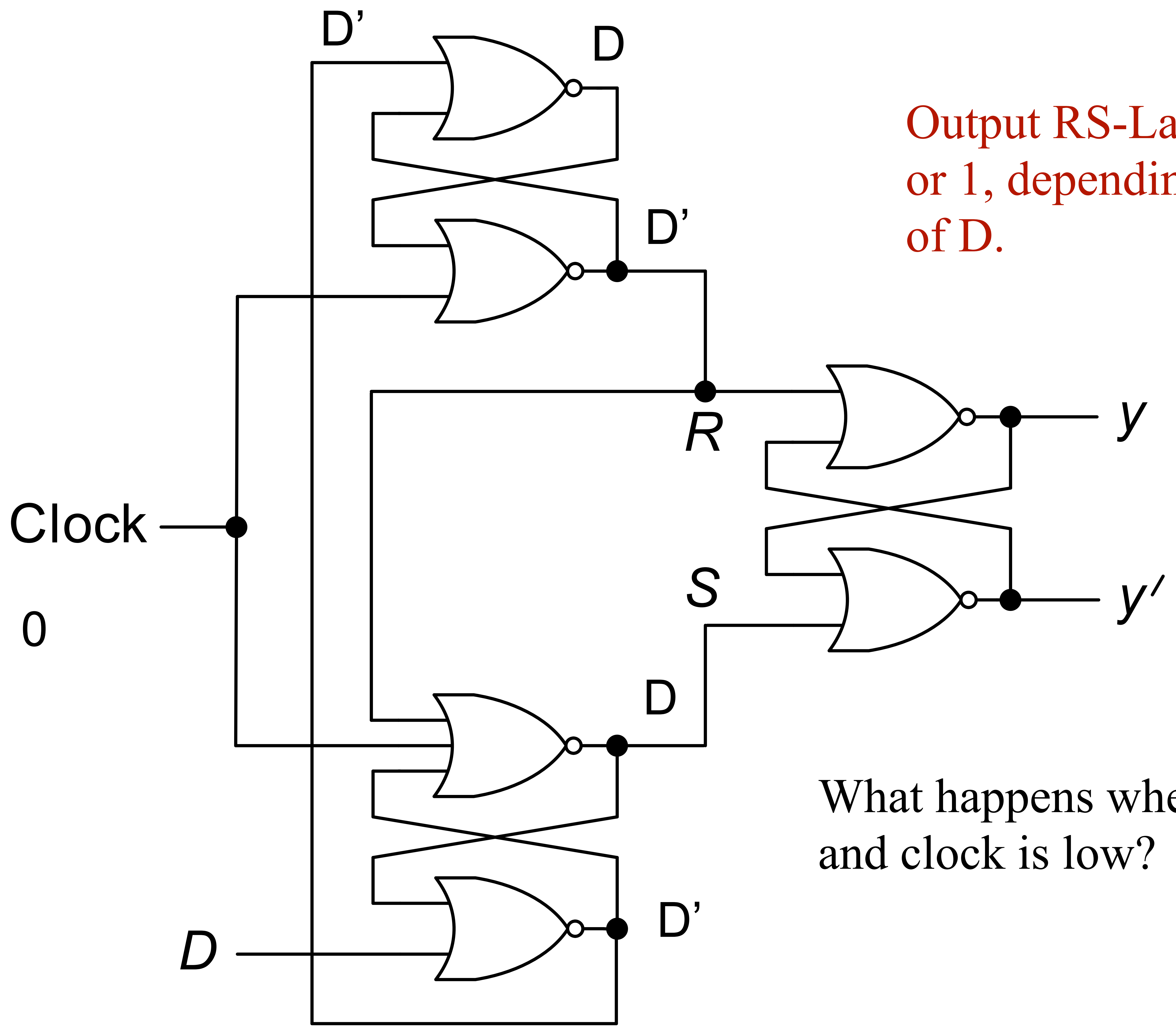


Edge Triggered Flip-Flop



Changes to the input D does not affect the output y and y' when the clock is high.

Edge Triggered Flip-Flop



Output RS-Latch stores a 0 or 1, depending on the value of D.

What happens when D changes and clock is low?

Synthesis of Synchronous Sequential Circuits

Main steps:

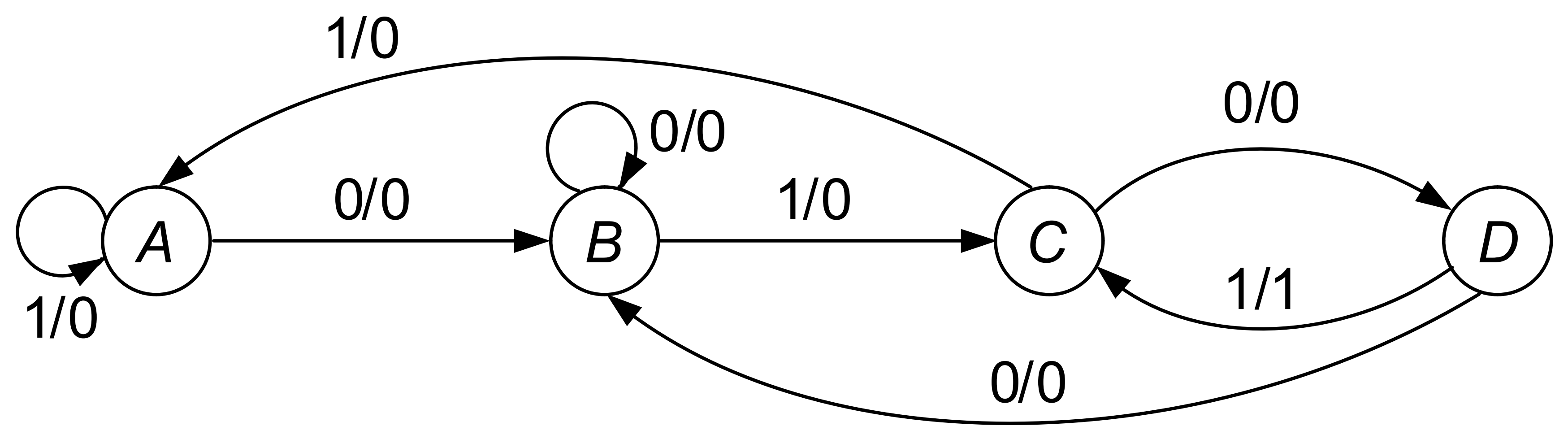
1. From a word description of the problem, form a **state diagram** or **table**
2. Check the table to determine if it contains any **redundant states**
    - If so, remove them (We will see this briefly)
3. Select a **state assignment** and determine the type of memory elements
4. Derive **transition** and **output** tables
5. Derive an **excitation table** and obtain **excitation** and **output functions** from their respective tables
6. Draw a **circuit diagram**

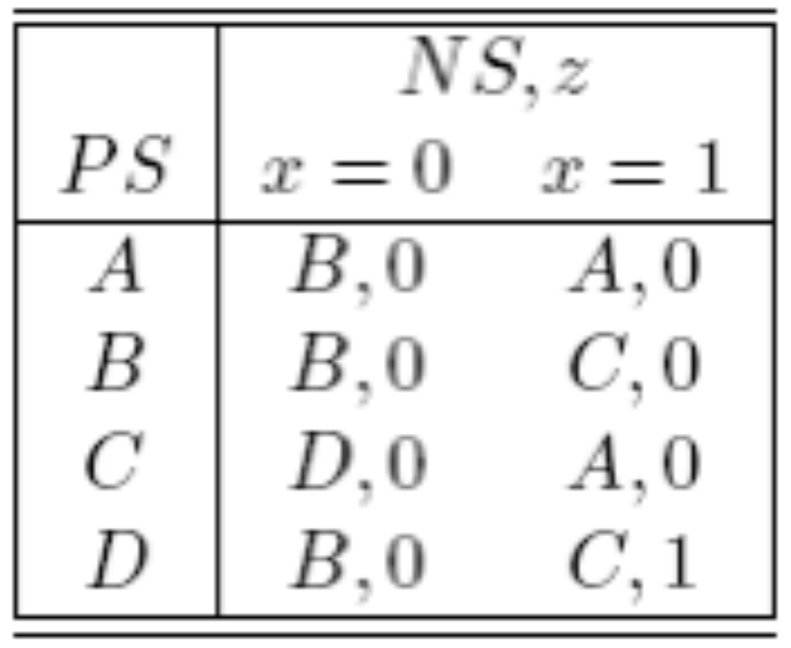
Synthesis of Synchronous Sequential Circuits

One-input/one-output sequence detector: produces output value 1 every time sequence 0101 is detected, else 0

- Example: 010101 -> 000101

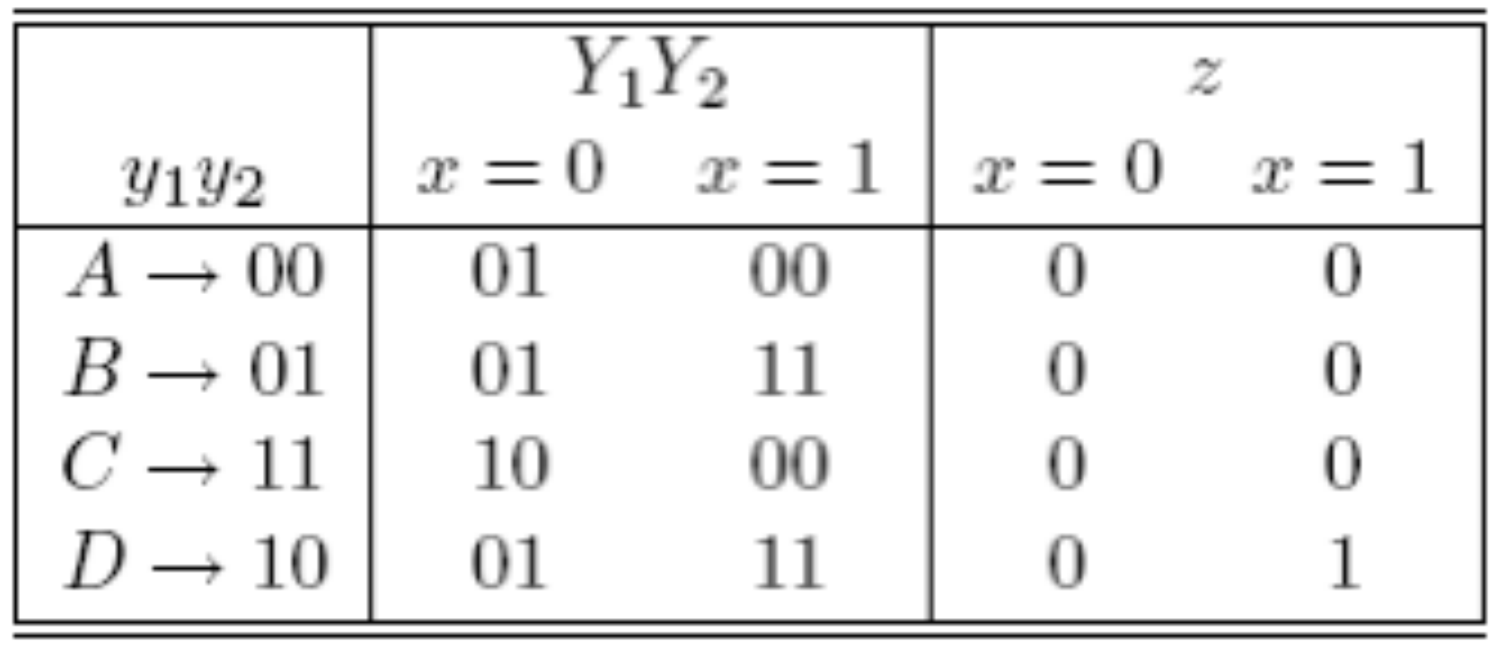
State diagram and state table:





PS	NS, z	
	$x = 0$	$x = 1$
A	$B, 0$	$A, 0$
B	$B, 0$	$C, 0$
C	$D, 0$	$A, 0$
D	$B, 0$	$C, 1$

Transition and output tables:



y_1y_2	Y_1Y_2		z	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$A \rightarrow 00$	01	00	0	0
$B \rightarrow 01$	01	11	0	0
$C \rightarrow 11$	10	00	0	0
$D \rightarrow 10$	01	11	0	1

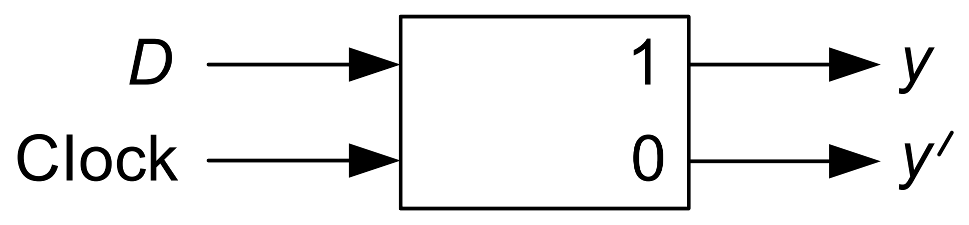
D Latch — The Latch of Your Life

The next state of the *D* latch is equal to its present excitation:

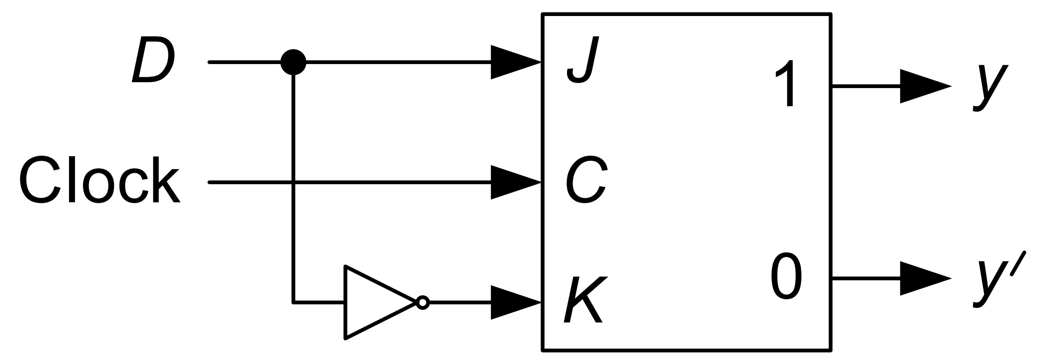
$$y(t+1) = D(t)$$

***D* Flip-Flop**

D	$Q(t+1)$	
0	0	Reset
1	1	Set



(*a*) Block diagram.



(*b*) Transforming the JK latch to the D latch.

Excitation Table

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Synthesis of Synchronous Sequential Circuits

Excitation and output maps:

y_1y_2	Y_1Y_2 $x=0$	Y_1Y_2 $x=1$	z $x=0$	z $x=1$
$A \rightarrow 00$	01	00	0	0
$B \rightarrow 01$	01	11	0	0
$C \rightarrow 11$	10	00	0	0
$D \rightarrow 10$	01	11	0	1

y_1y_2 \ x	0	1
00		
01		
11		
10		1

(*a*) *z* map.

y_1y_2 \ x	0	1
00		
01		1
11	1	
10		1

(*b*) Y_1 map.

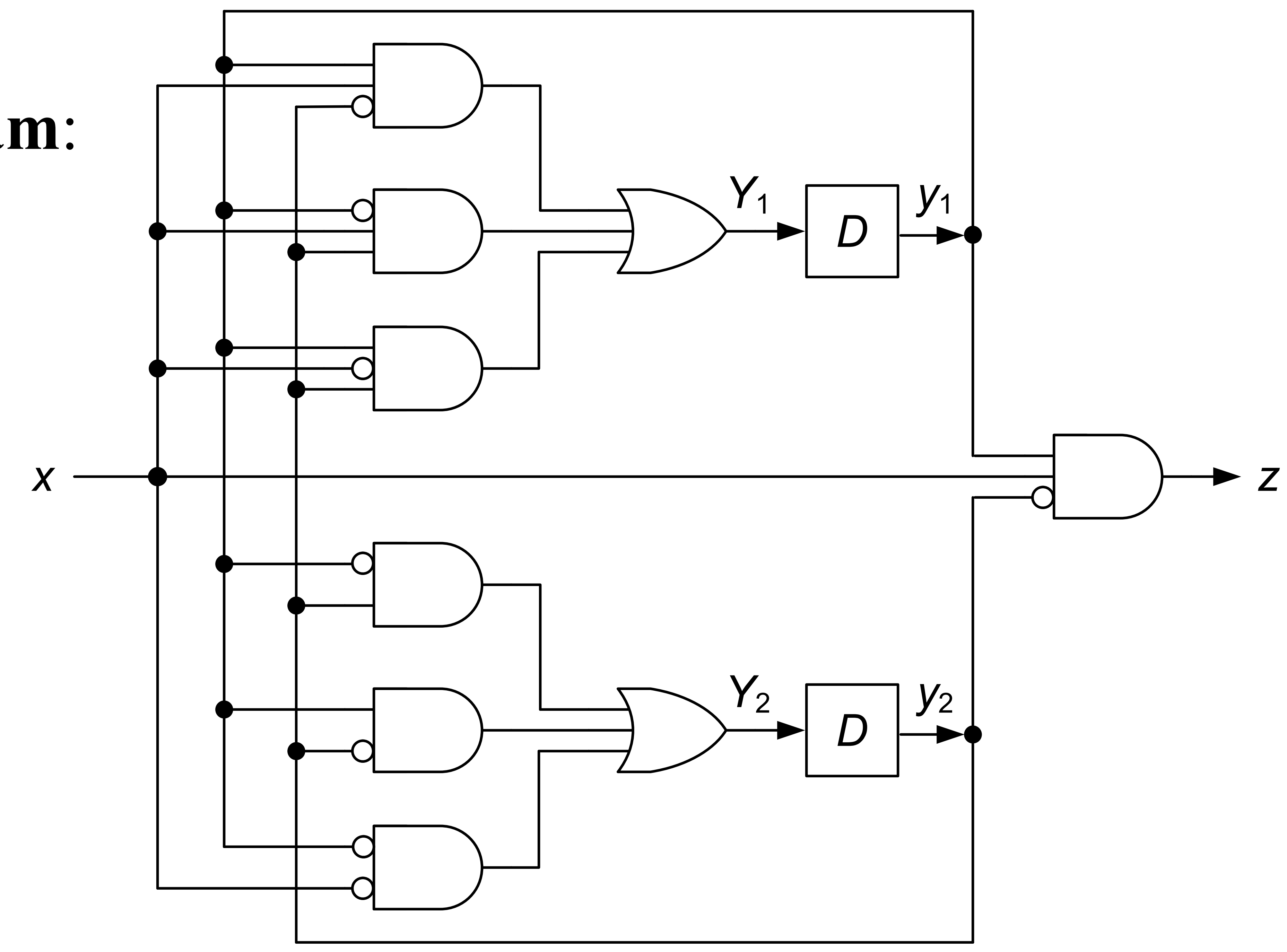
y_1y_2 \ x	0	1
00	1	
01	1	1
11		
10	1	1

(*c*) Y_2 map.

$$z = xy_1y_2'$$
$$y_1 = x'y_1y_2 + xy_1'y_2 + xy_1y_2'$$
$$y_2 = y_1y_2' + x'y_1' + y_1'y_2$$

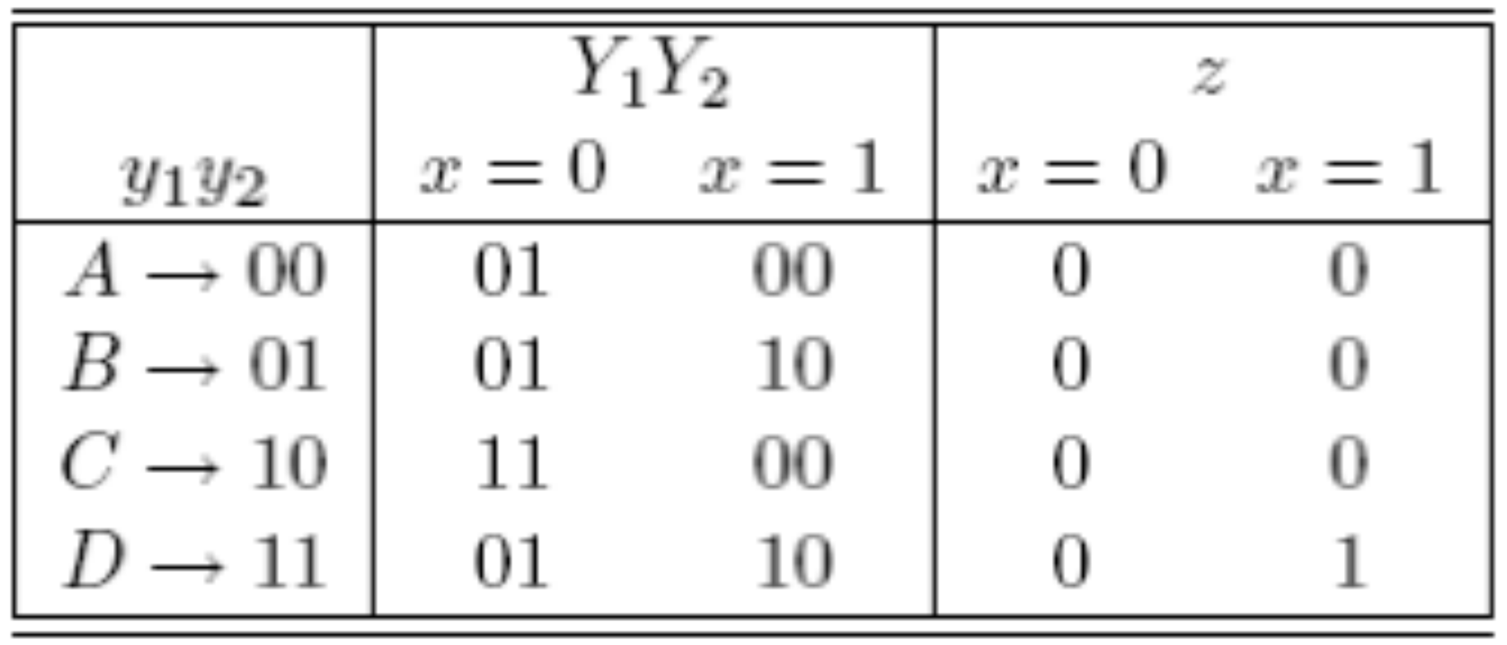
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Logic diagram:



Synthesis of Synchronous Sequential Circuits

Another state assignment:



	Y_1Y_2		z	
y_1y_2	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$A \rightarrow 00$	01	00	0	0
$B \rightarrow 01$	01	10	0	0
$C \rightarrow 10$	11	00	0	0
$D \rightarrow 11$	01	10	0	1

$z = xy_1y_2$

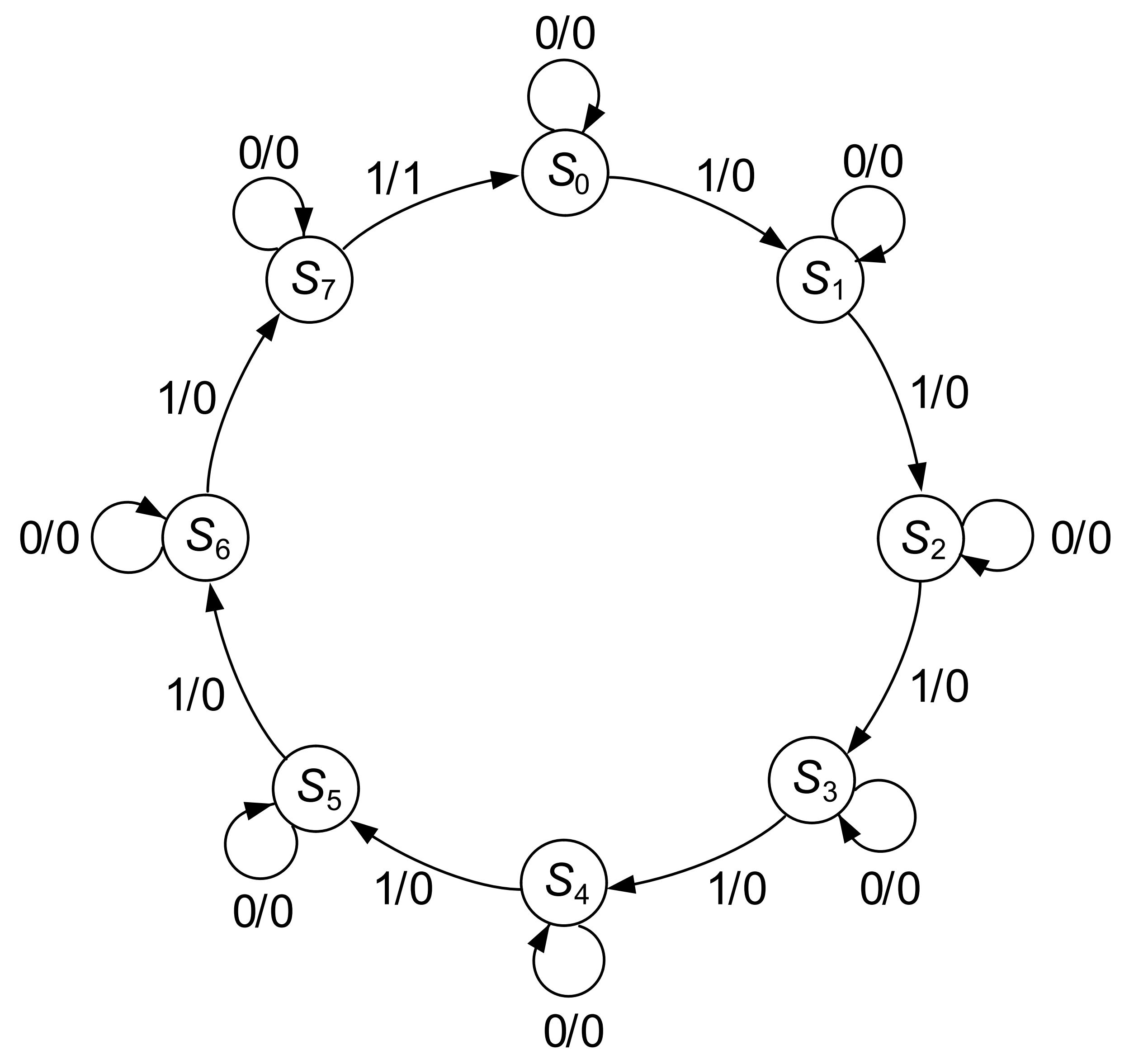
$Y_1 = x'y_1y_2' + xy_2$

$Y_2 = x'$

Binary Counter

One-input/one-output modulo-8 binary counter: produces output value 1 for every eighth input 1 value

State diagram and state table:



PS	NS		$Output$	
	$x=0$	$x=1$	$x=0$	$x=1$
S_0	S_0	S_1	0	0
S_1	S_1	S_2	0	0
S_2	S_2	S_3	0	0
S_3	S_3	S_4	0	0
S_4	S_4	S_5	0	0
S_5	S_5	S_6	0	0
S_6	S_6	S_7	0	0
S_7	S_7	S_0	0	1

Binary Counter

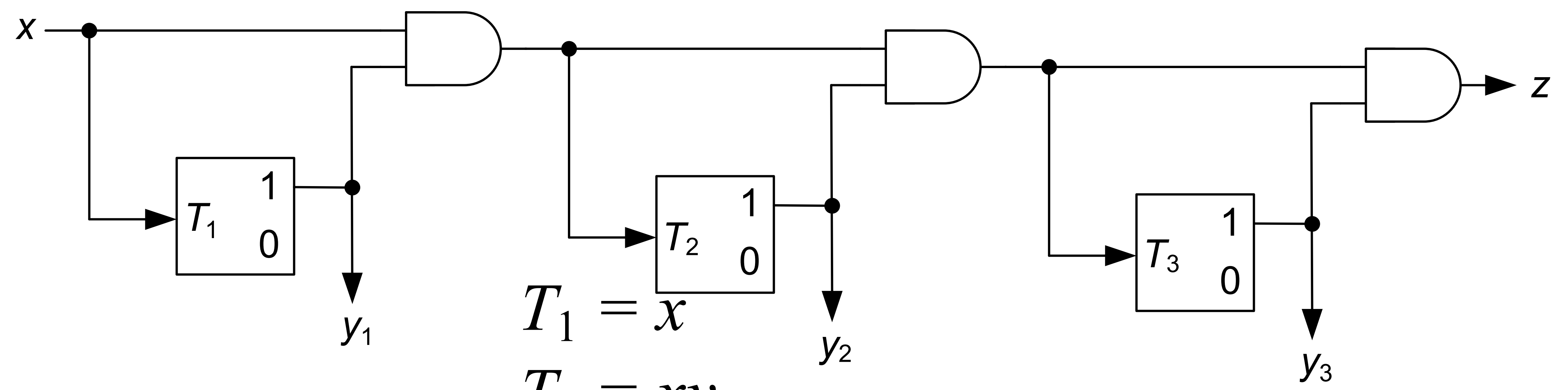
Transition and output tables:

PS	*NS*		*z*	
$y_3y_2y_1$	$x=0$	$x=1$	$x=0$	$x=1$
000	000	001	0	0
001	001	010	0	0
010	010	011	0	0
011	011	100	0	0
100	100	101	0	0
101	101	110	0	0
110	110	111	0	0
111	111	000	0	1

	$T_3T_2T_1$	
$y_3y_2y_1$	$x=0$	$x=1$
000	000	001
001	000	011
010	000	001
011	000	111
100	000	001
101	000	011
110	000	001
111	000	111

Excitation table for *T*

Circuit change		*Required*
From:	*To:*	*value T*
0	0	0
0	1	1
1	0	1
1	1	0



$$T_1 = x$$
$$T_2 = xy_1$$
$$T_3 = xy_1y_2$$
$$z = xy_1y_2y_3$$

Binary Counter with SR Flip Flops

Transition and output tables:

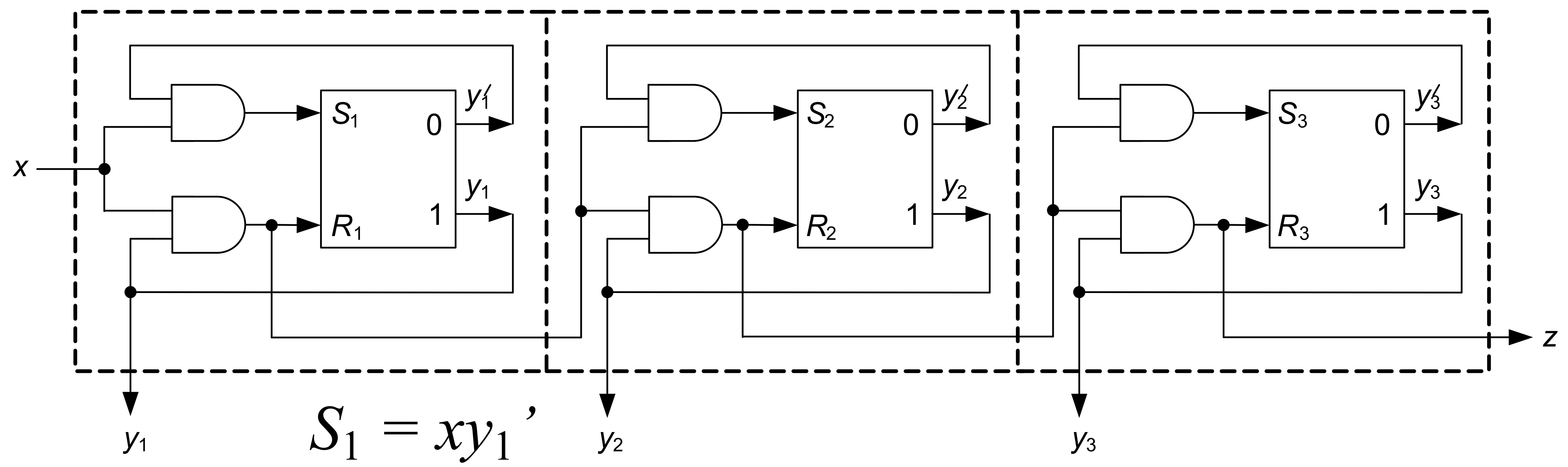
PS	*NS*		*z*	
$y_3y_2y_1$	$x=0$	$x=1$	$x=0$	$x=1$
000	000	001	0	0
001	001	010	0	0
010	010	011	0	0
011	011	100	0	0
100	100	101	0	0
101	101	110	0	0
110	110	111	0	0
111	111	000	0	1

	$x=0$			$x=1$		
$y_3y_2y_1$	S_3R_3	S_2R_2	S_1R_1	S_3R_3	S_2R_2	S_1R_1
000	0–	0–	0–	0–	0–	10
001	0–	0–	–0	0–	10	01
010	0–	–0	0–	0–	–0	10
011	0–	–0	–0	10	01	01
100	–0	0–	0–	–0	0–	10
101	–0	0–	–0	–0	10	01
110	–0	–0	0–	–0	–0	10
111	–0	–0	–0	01	01	01

Excitation table for *SR* flip-flops and logic diagram:

- **Trivially extensible to modulo-16 counter**

Circuit change		*Required value*	
From:	*To:*	*S*	*R*
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0



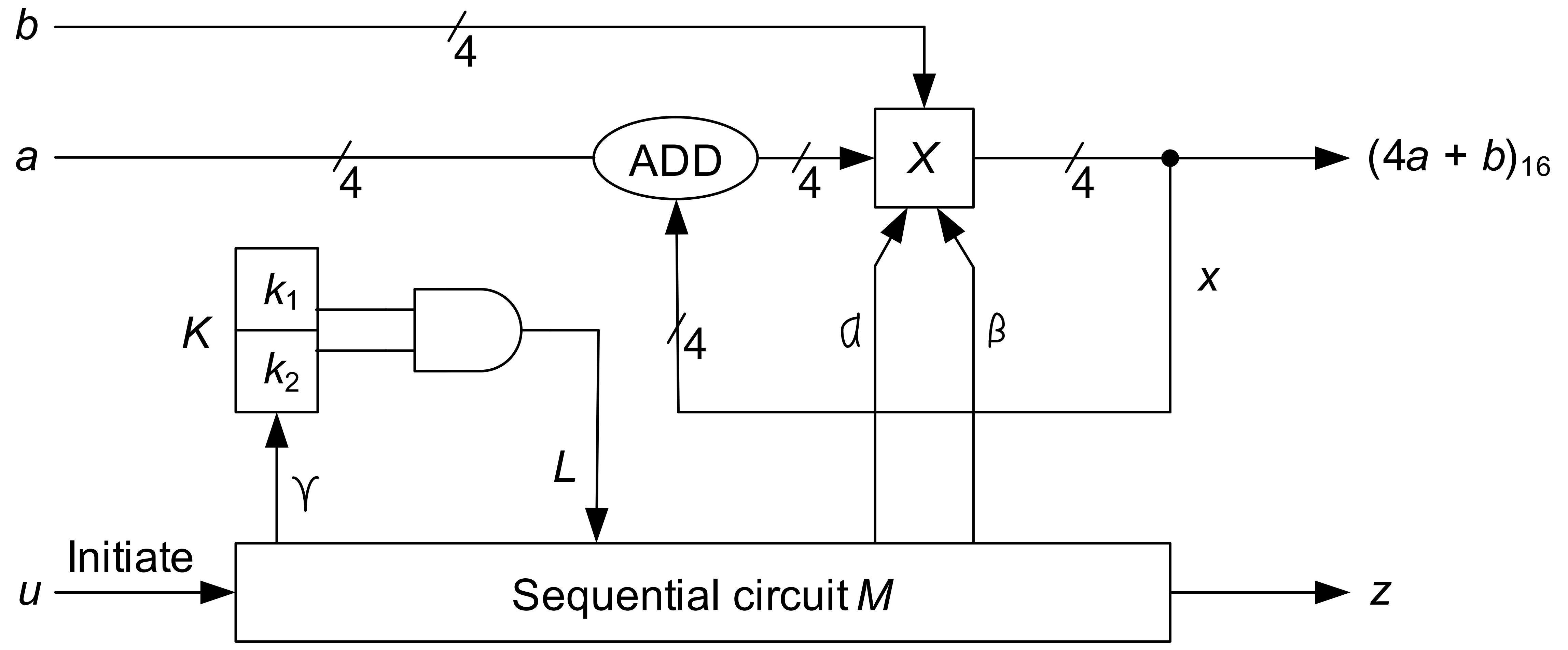
$$S_1 = xy_1'$$
$$R_1 = xy_1$$
$$S_2 = xy_1y_2'$$
$$R_2 = xy_1y_2$$
$$S_3 = xy_1y_2y_3'$$
$$R_3 = z = xy_1y_2y_3$$

Sequential Circuit as a Controller

Control element: streamlines computation by providing appropriate control signals

Example: digital system that computes the value of $(4a + b)$ modulo 16

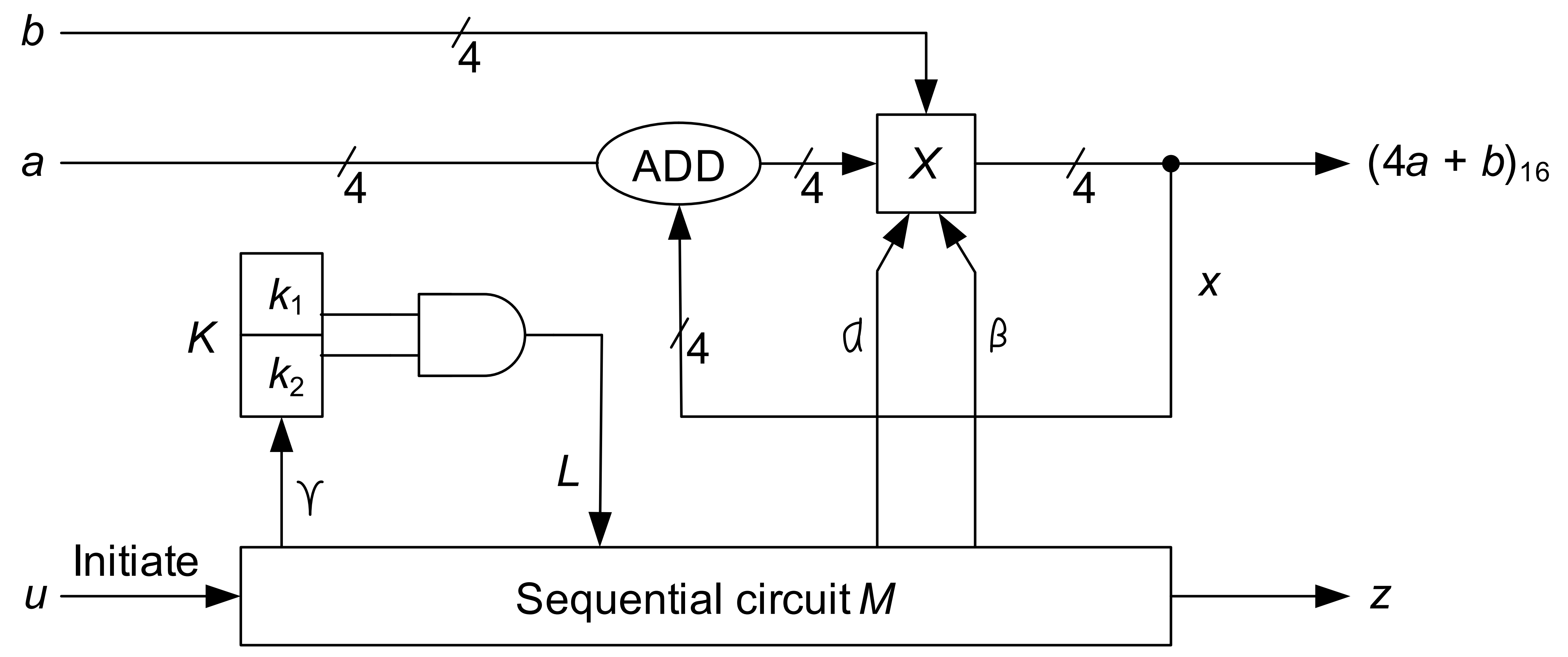
- a, b: four-bit binary number
- X: register containing four flip-flops
- x: number stored in X
- Register can be loaded with: either b or $a + x$
- Addition performed by: a four-bit parallel adder
- K: modulo-4 binary counter, whose output L equals 1 whenever the count is 3 modulo 4



Sequential Circuit as a Controller

Sequential circuit *M*:

- Input u: initiates computation
- Input L: gives the count of K
- Outputs: α, β, γ, z
- When $\alpha = 1$: contents of b transferred to X
- When $\beta = 1$: values of x and a added and transferred back to X
- When $\gamma = 1$: count of K increased by 1
- $z = 1$: whenever final result available in X

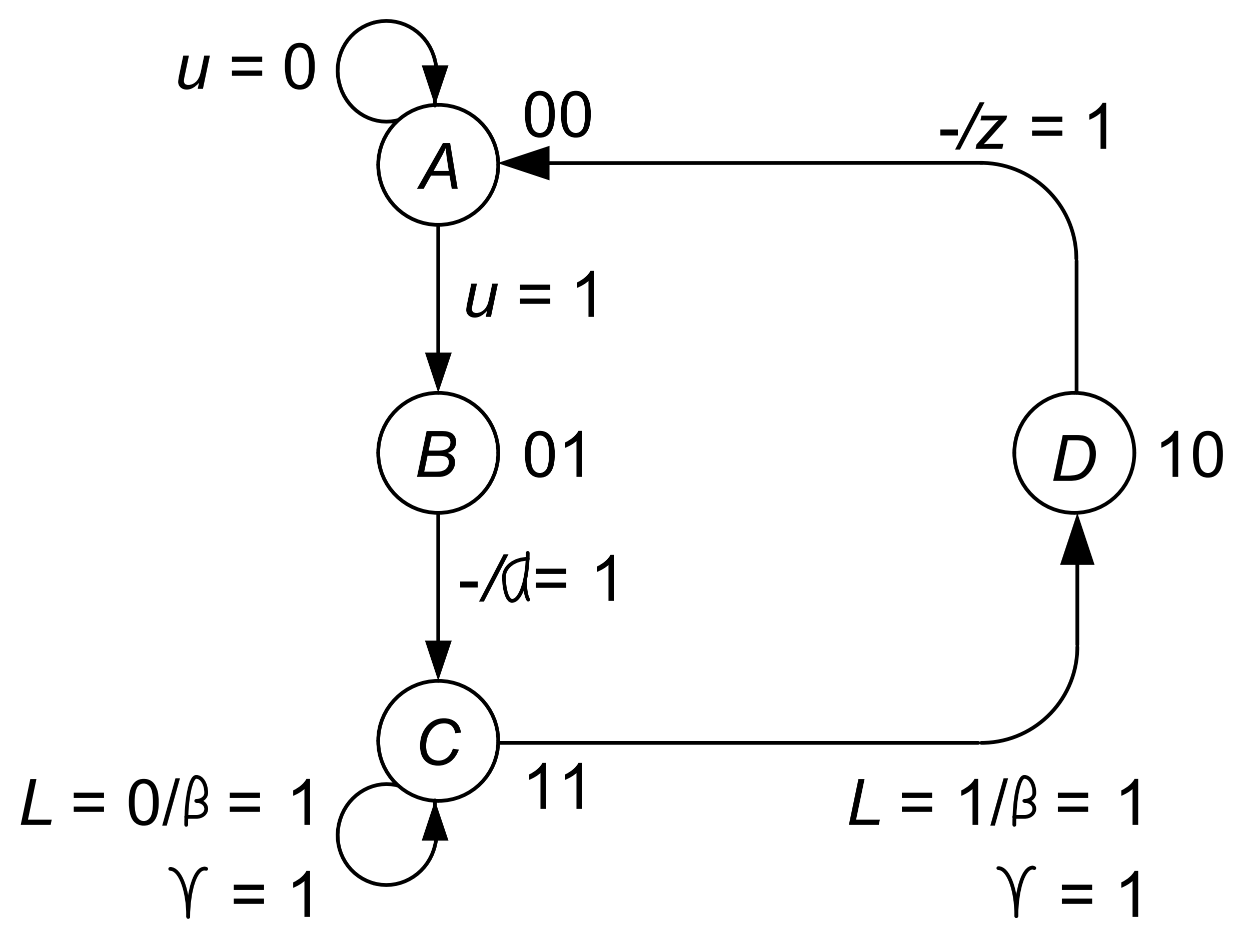


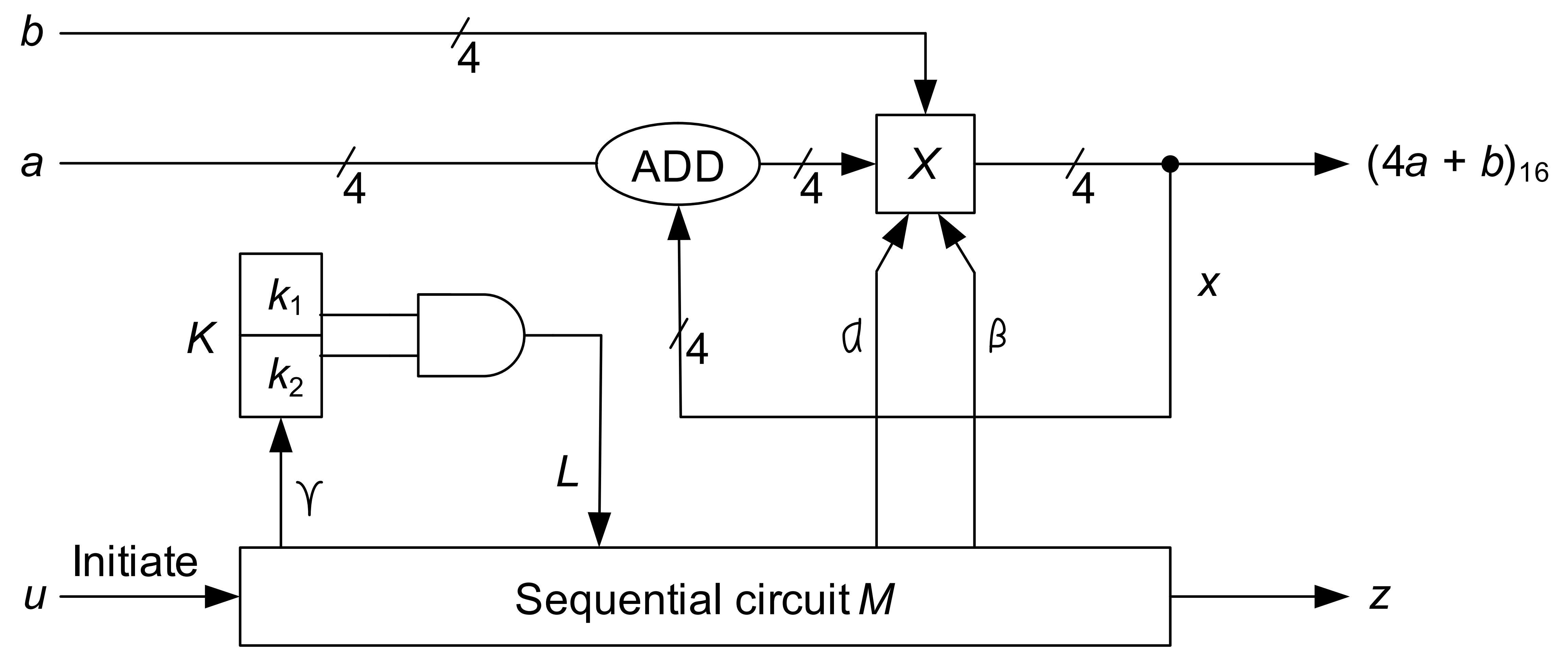
Sequential Circuit as a Controller

Sequential circuit *M*:

- *K*, *u, z*: initially at 0
- When $u = 1$: computation starts by setting $\alpha = 1$
  - Causes *b* to be loaded into *X*
- To add *a* to *x*: set $\beta = 1$ and $\gamma = 1$ to keep track of the number of times *a* has been added to *x*
- After four such additions: $z = 1$ and the computation is complete
- At this point: $K = 0$ to be ready for the next computation

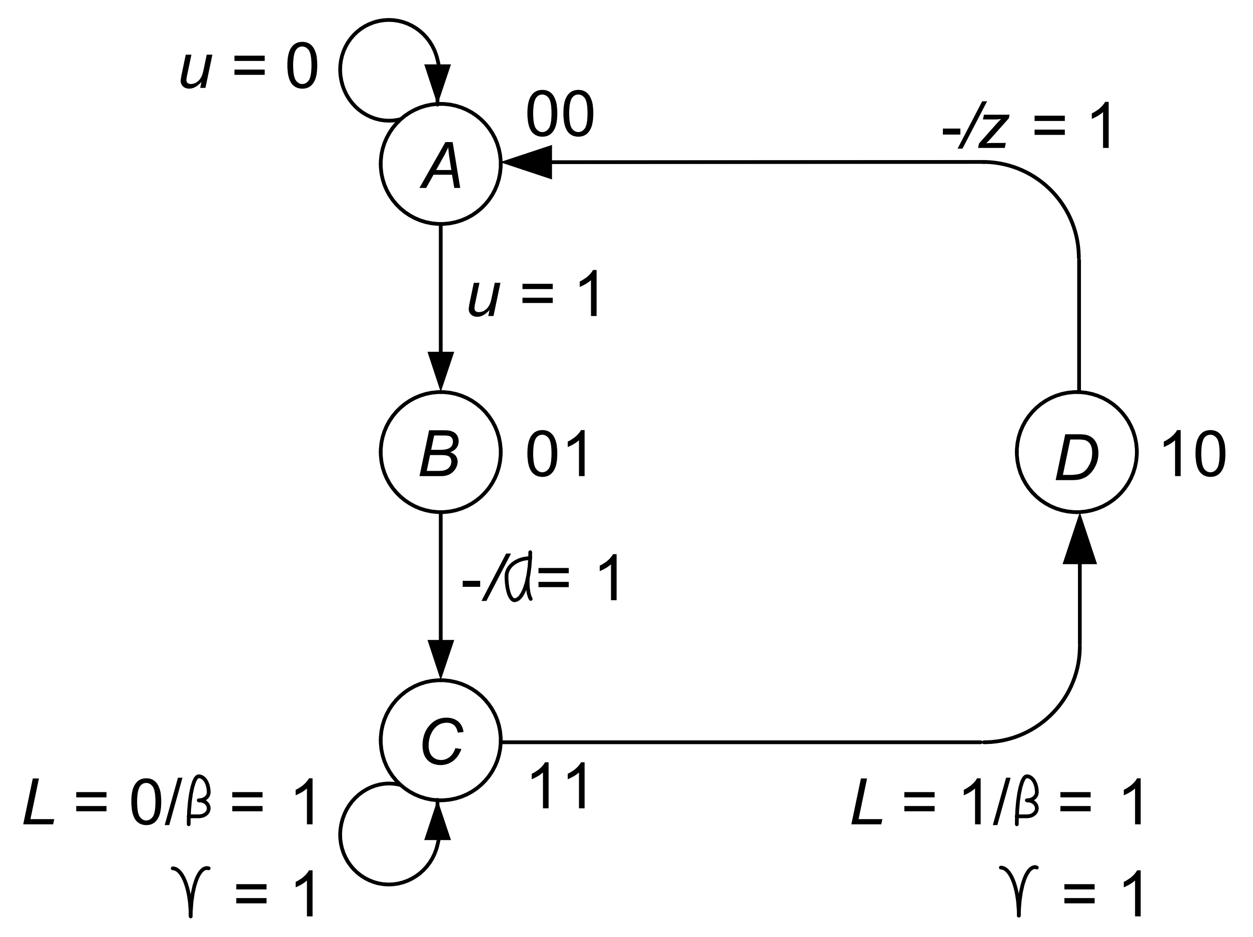
State diagram:





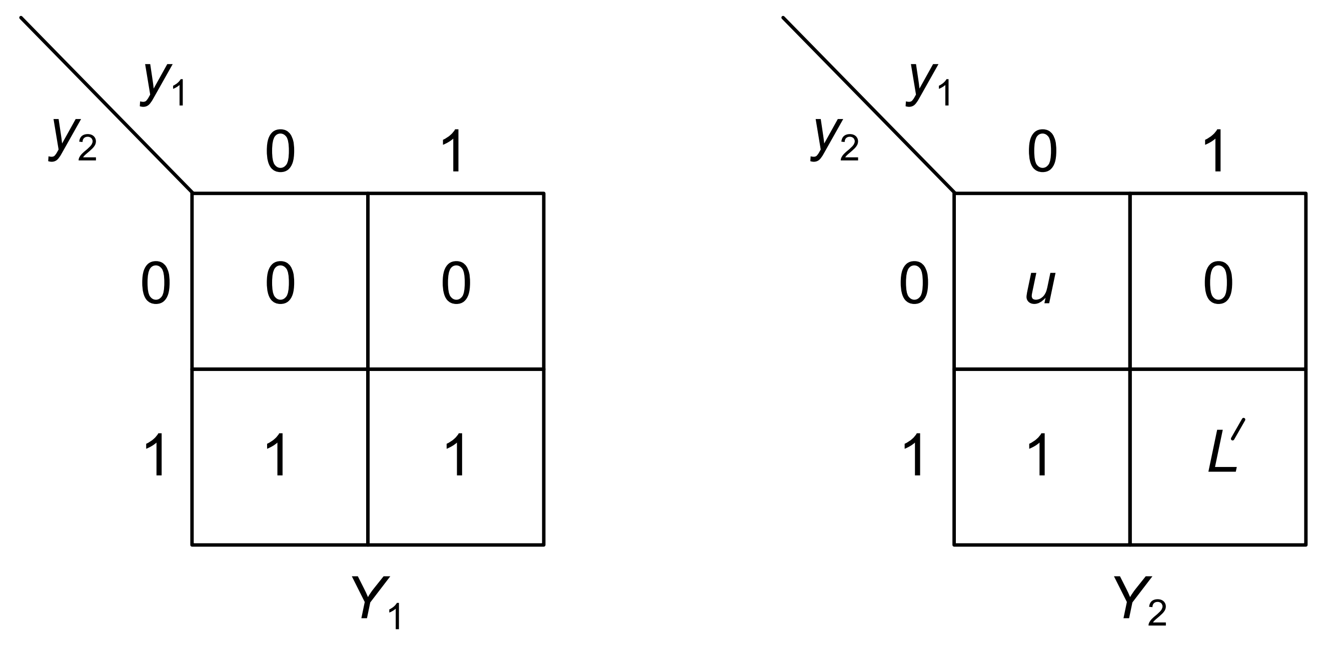
Sequential Circuit as a Controller

State assignment, transition table, maps and logic diagram:



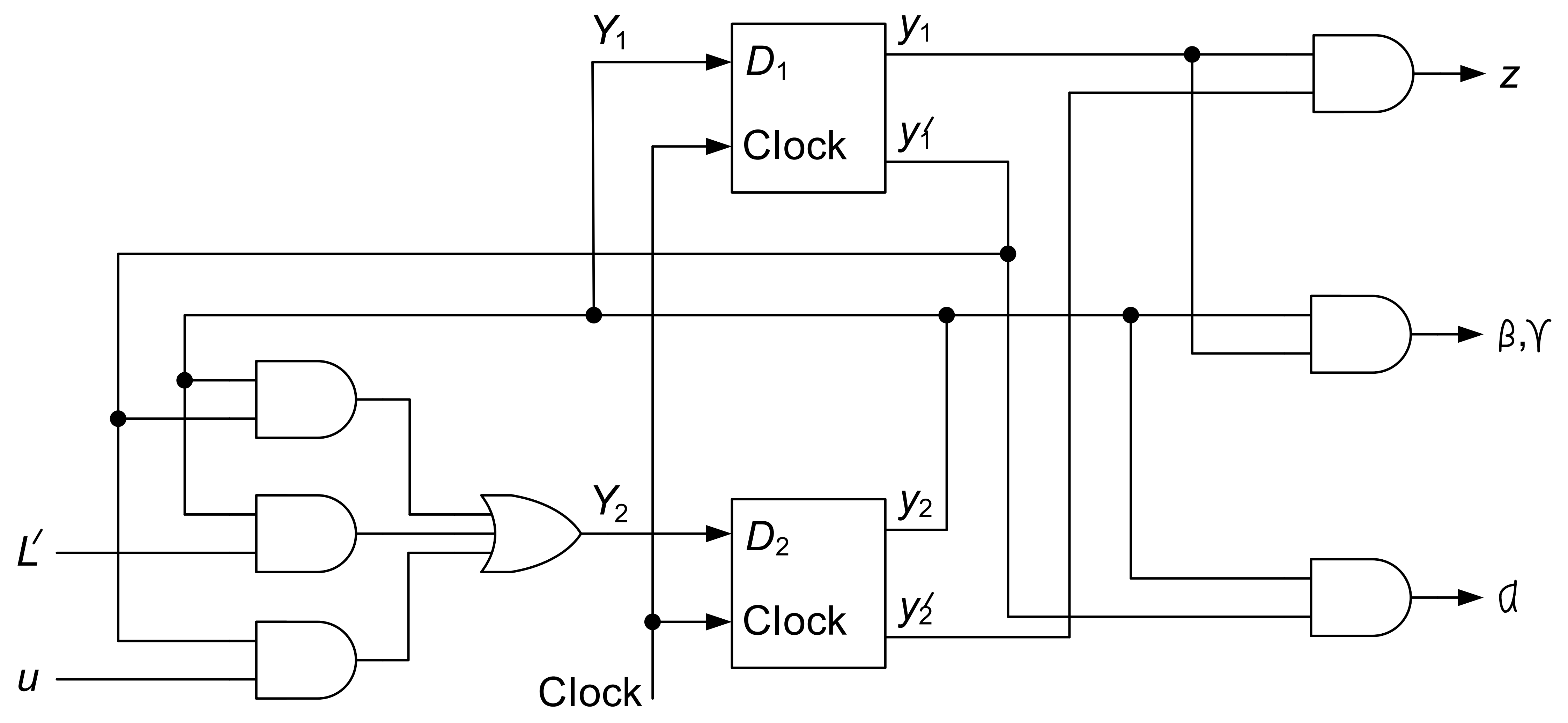
PS y_1y_2	*NS* Y_1Y_2
00	0*u*
01	11
11	1*L*′
10	00

(*a*) Transition table.



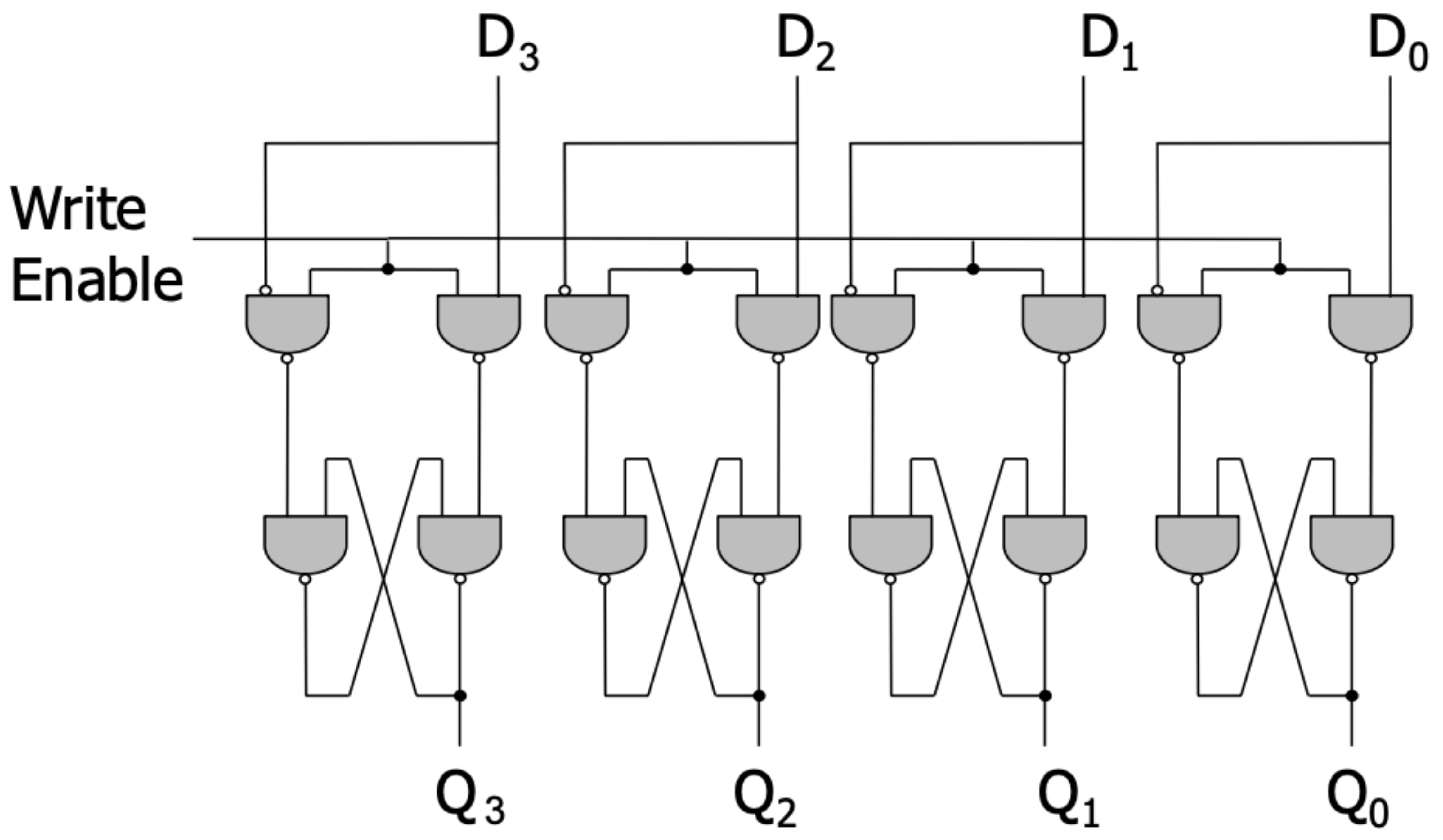
(*b*) Maps for Y_1 and Y_2.

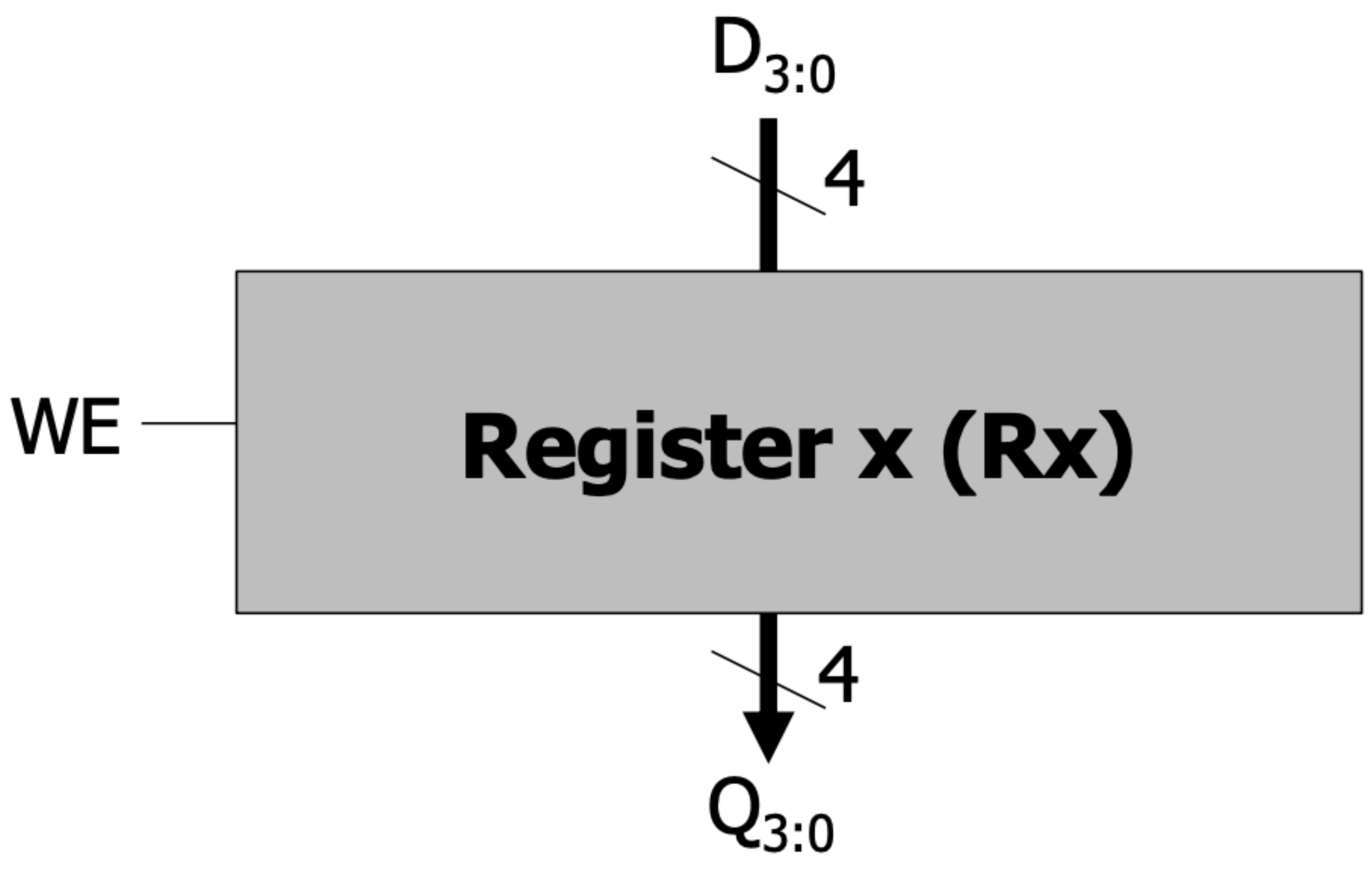
$$\alpha = y_1'y_2$$
$$\beta = \gamma = y_1y_2$$
$$z = y_1y_2'$$
$$Y_1 = y_2$$
$$Y_2 = y_1'y_2 + uy_1' + L'y_2$$



(*c*) Logic diagram.

Registers: Your Main Sequential Element

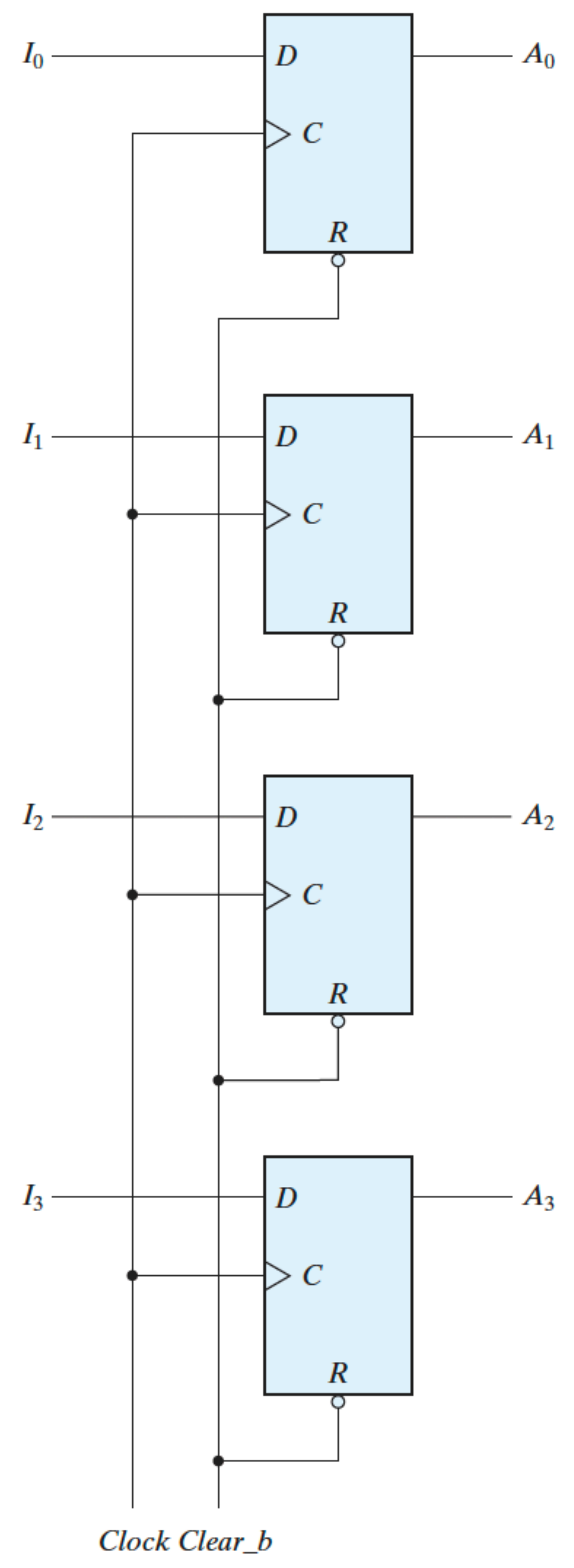




- Used to store data
- Basically an **array of D-flip-flops**
- You can **load data**, **reset it to zero**, and **shift it to left and right**

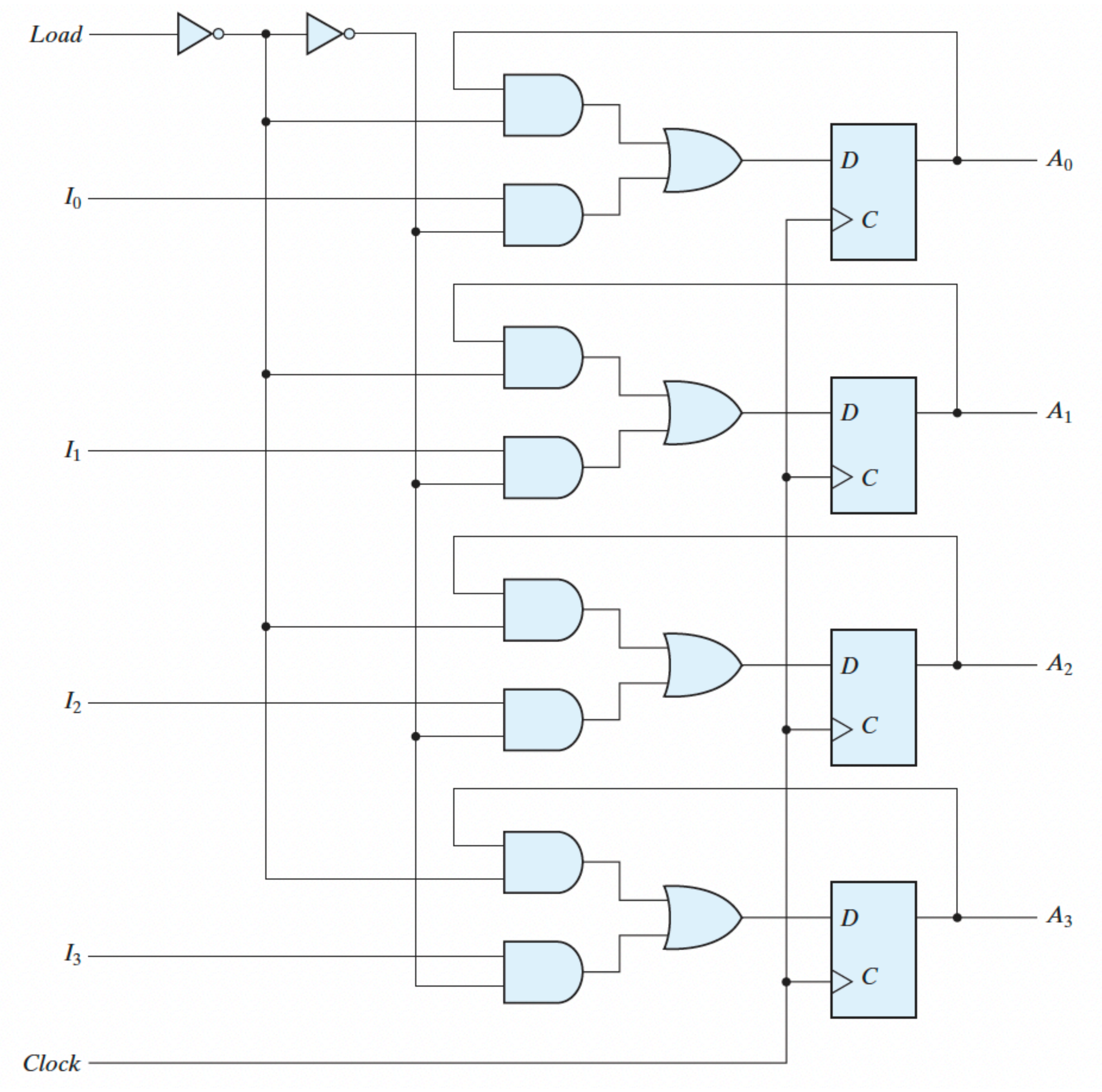
Registers: Your Main Sequential Element

- 4-bit register
- Asynchronous Reset
- On a clock tick, the data in I0, I1, I2, I3 gets available in A0, A1, A2, A3.



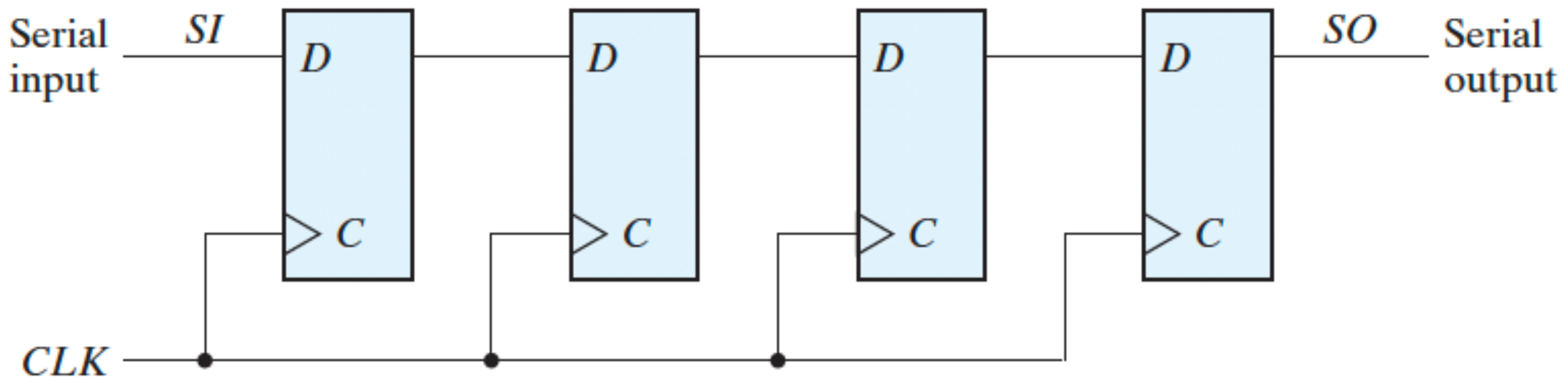
Registers: Your Main Sequential Element

- 4-bit register with parallel load
- Asynchronous Reset
- The main difference from the previous design is that in the former case the stored data deliberately changes at every clock tick. But here we have a control through the *Load* line.
- This is your "**the building block**"
- Don't worry you do not need to code it down. Verilog does this internally for you.
  - But maybe you should give it a try
- Your processor in the rest of the course contains such registers!!!



Shift Registers

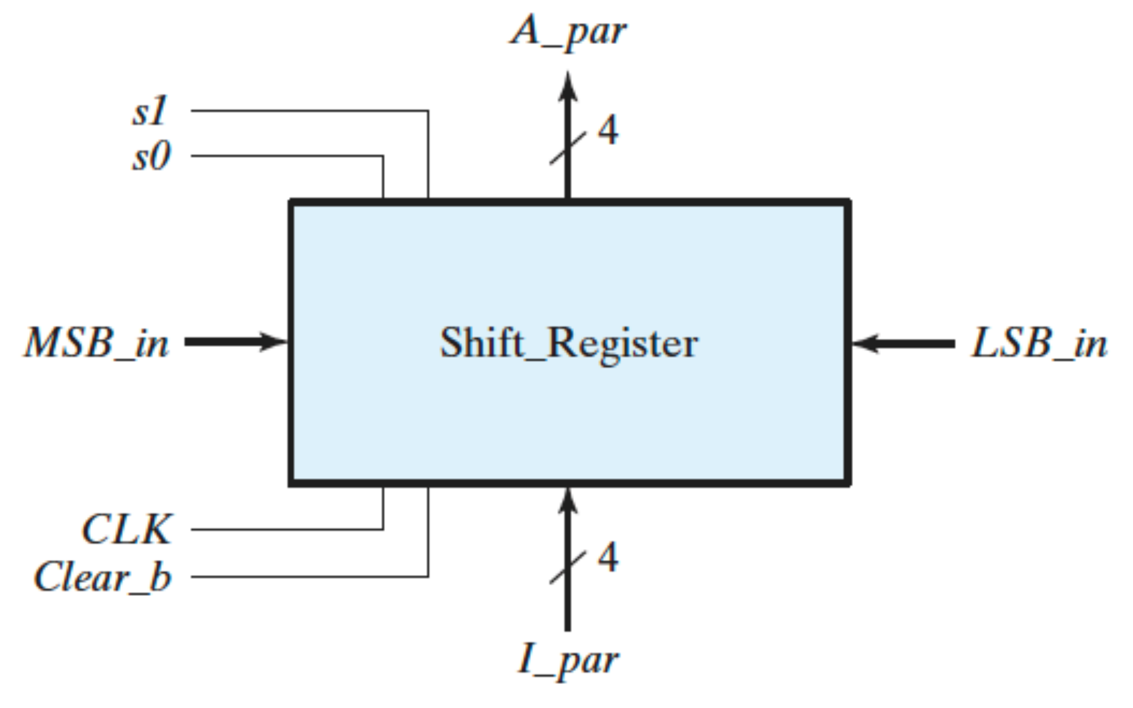
- Shift the bits left and right
- Again, very easy to write in Verilog



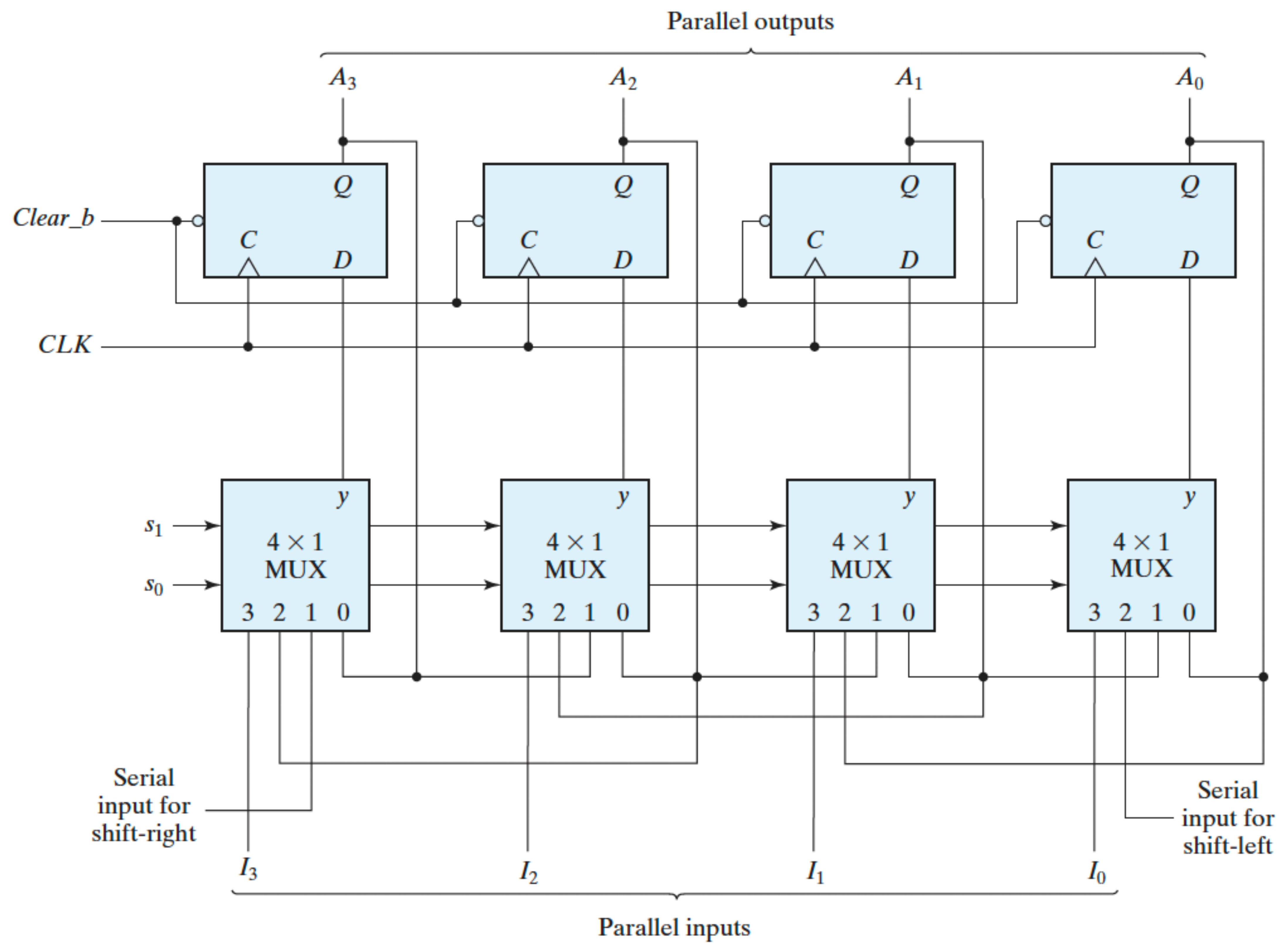
Shift Registers

- Universal Shift Register

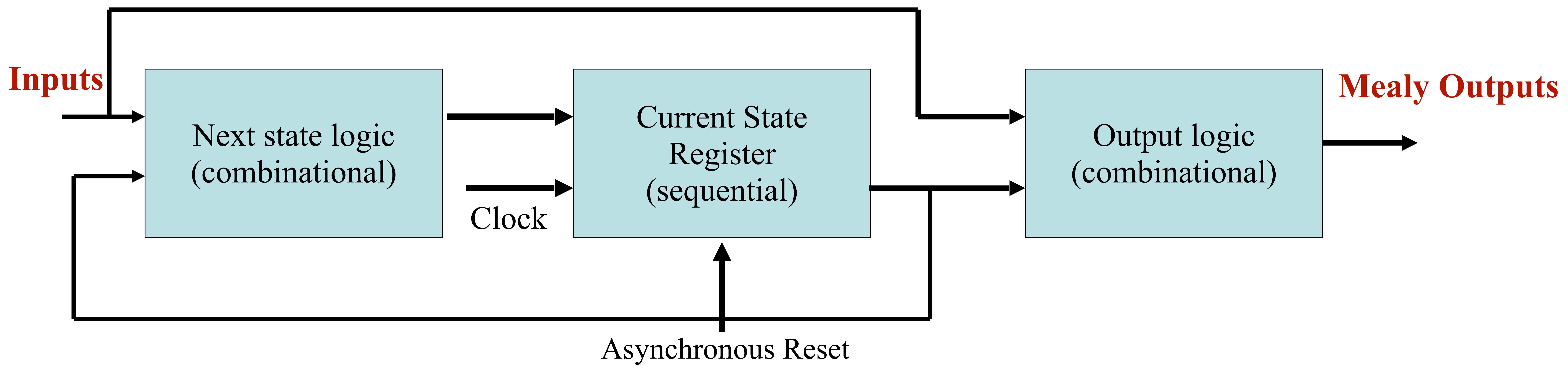
Mode Control		
s_1	s_0	**Register Operation**
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

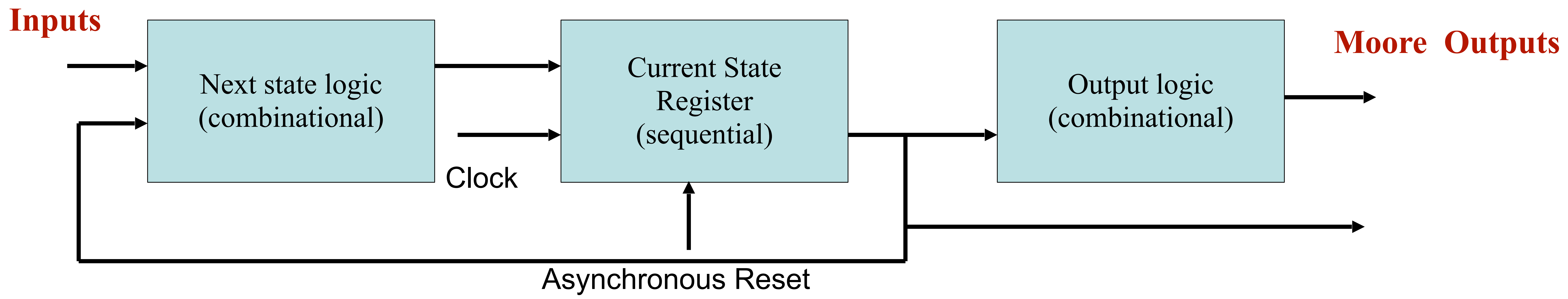


(a)



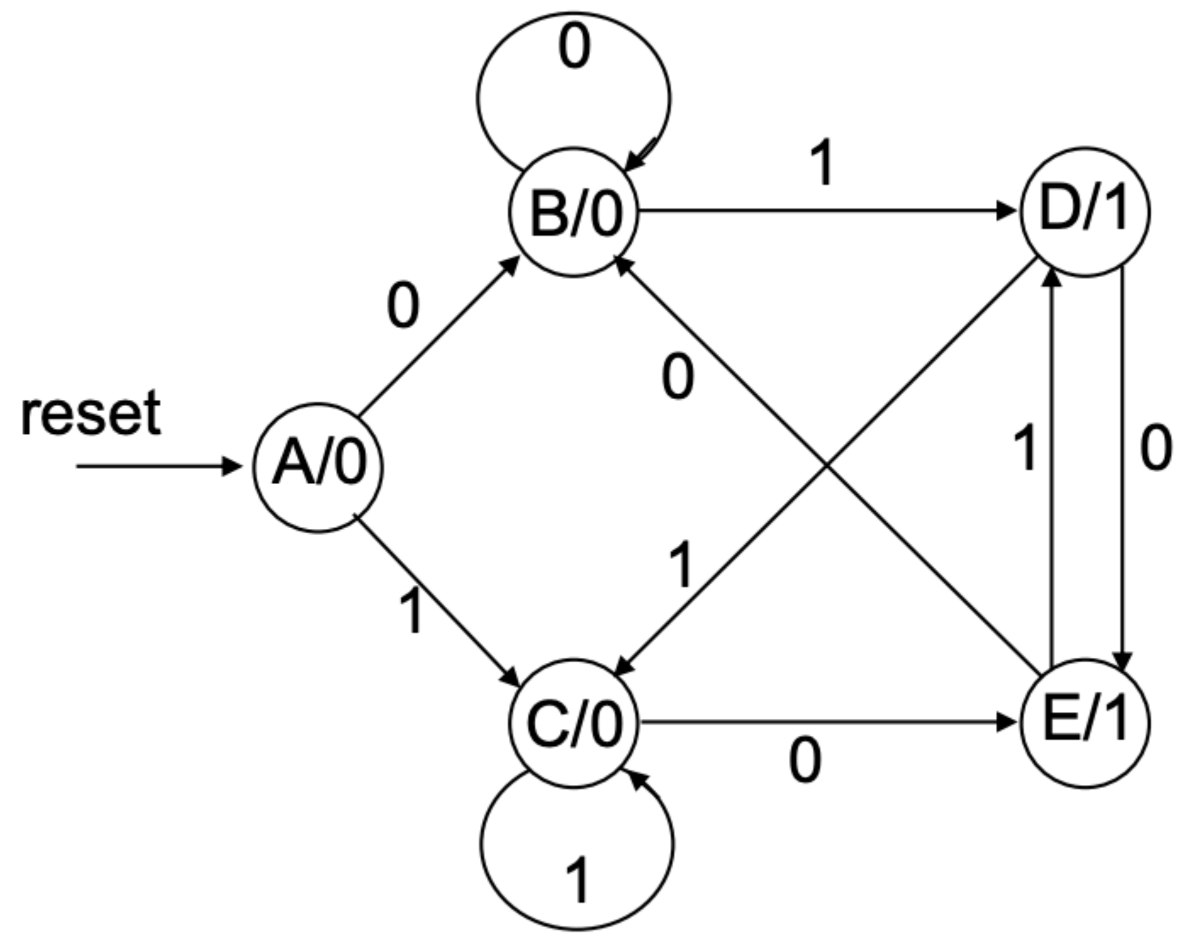
Mealy and Moore Machines


Inputs
Next state logic (combinational)
Clock
Current State Register (sequential)
Asynchronous Reset
Output logic (combinational)
Mealy Outputs


Inputs
Next state logic (combinational)
Clock
Current State Register (sequential)
Asynchronous Reset
Output logic (combinational)
Moore Outputs

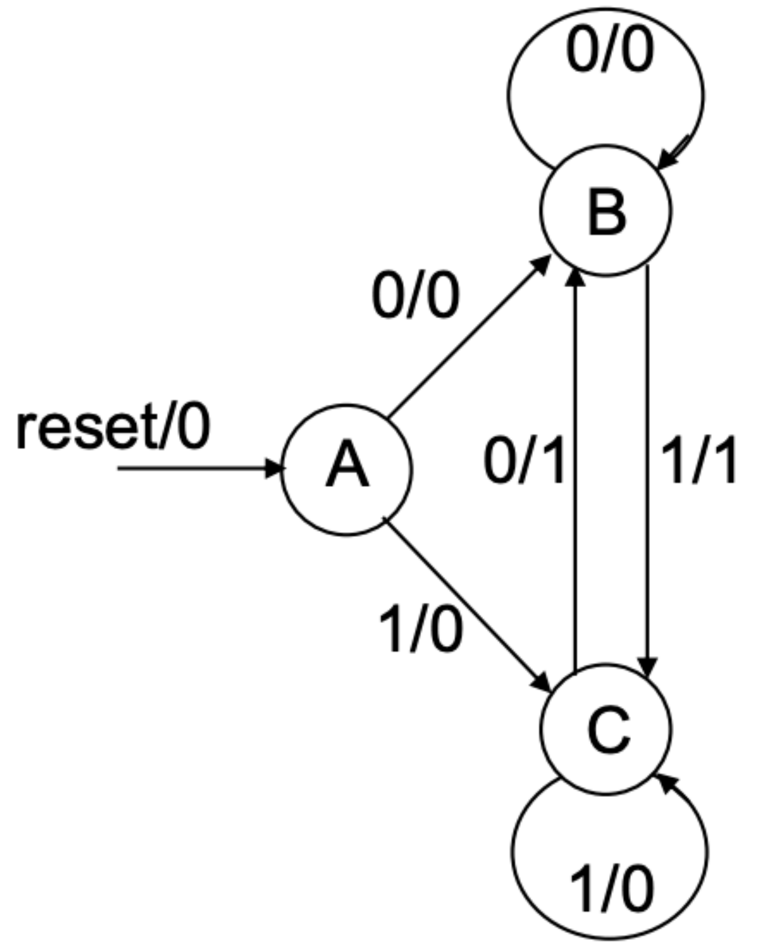
Example: 01/10 Detector

Moore



reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

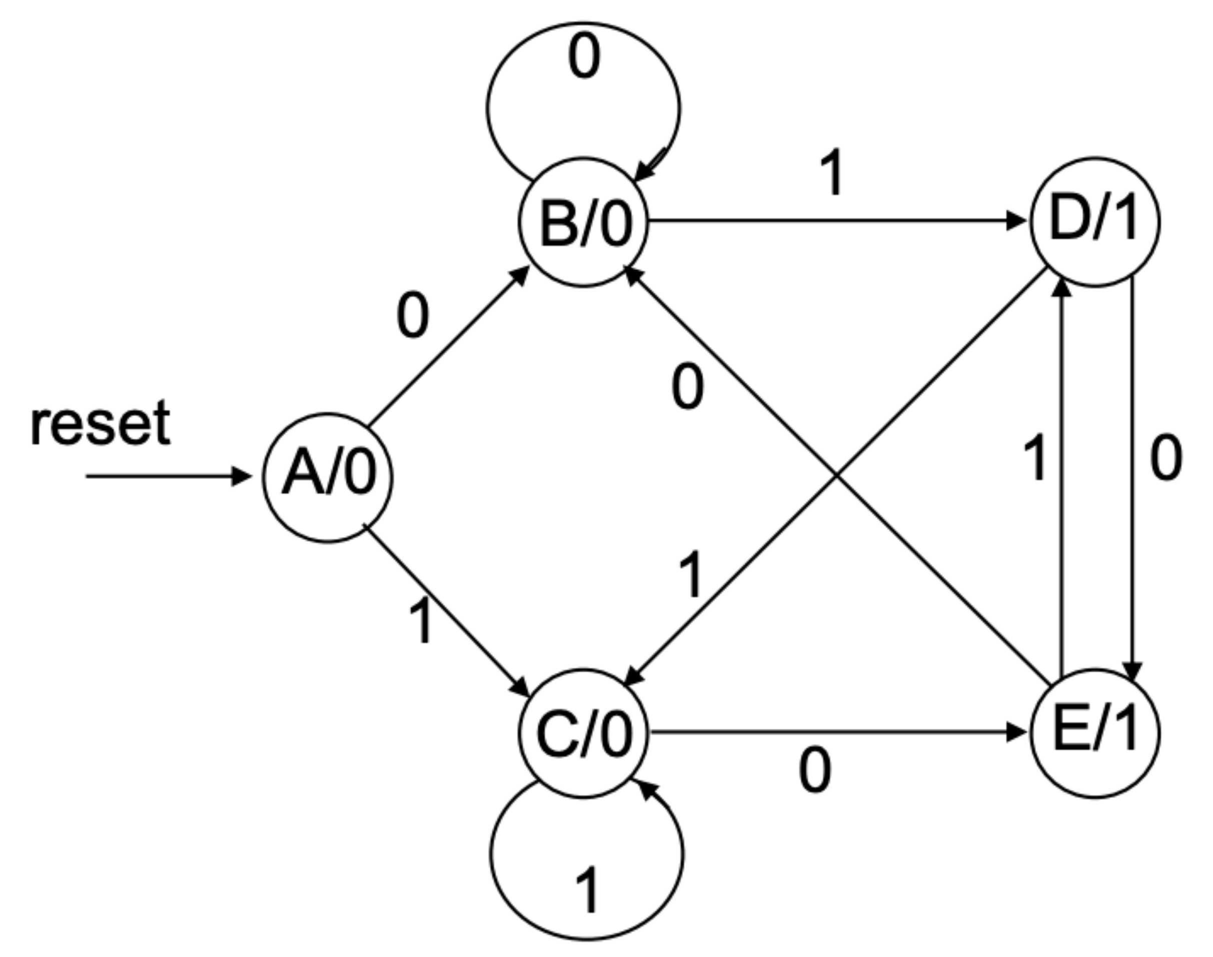
Mealy



reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

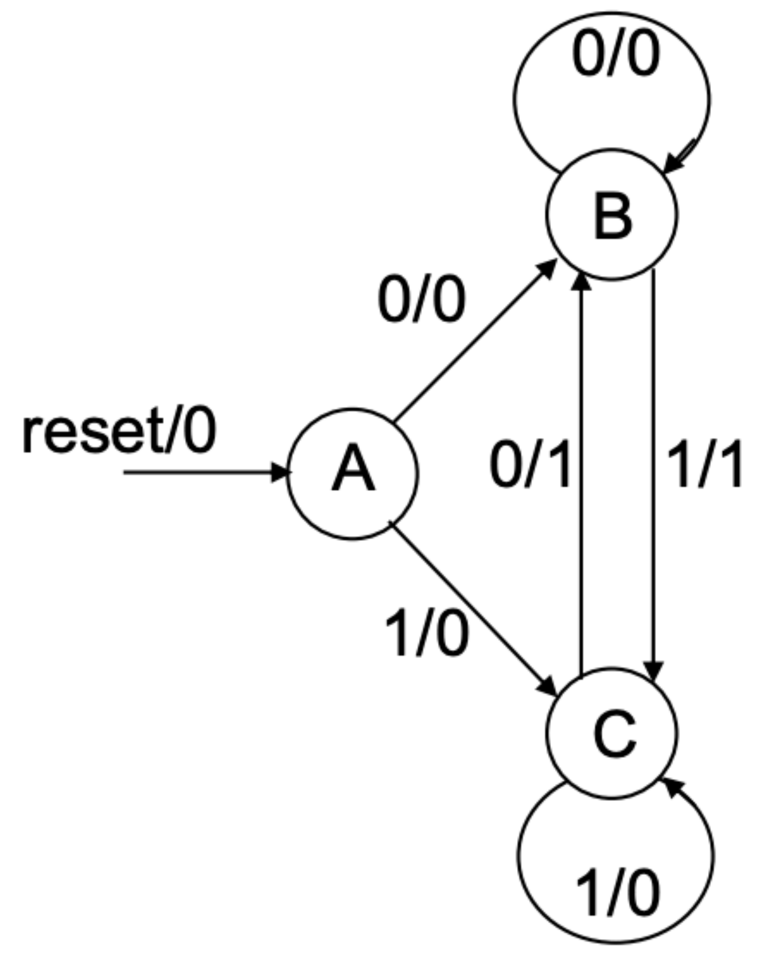
Example: 01/10 Detector

Moore



reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

Mealy



reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

State Machine Minimization

More states $\longrightarrow$ **more flip-flops and logic**:

- Cost saving is always important
- Remember circuit minimization
- We shall now see how to minimize the number of states in a state machine
- Again some complex stuff: Sorry :P

State Machine Minimization

n-state machine $\longrightarrow$ **ceil[log$_2$(n)] state variables**

- Sometimes we have redundant states
- Redundant states are also called *equivalent states*

- ***k*-distinguishable states**: Two states S_i and S_j of a machine M are distinguishable iff there exists at least one finite input sequence that, when applied to M, causes different output sequence depending on whether S_i or S_j is the initial state. M is called *k-distinguishable* if the length of the distinguishing sequence is **k.**

	NS, z	
PS	$x = 0$	$x = 1$
A	$E, 0$	$D, 1$
B	$F, 0$	$D, 0$
C	$E, 0$	$B, 1$
D	$F, 0$	$B, 0$
E	$C, 0$	$F, 1$
F	$B, 0$	$C, 0$

- The pair (AB) is 1-distinguishable
- (AE) is 3-distinguishable for the input X = 111 — check it!!

State Machine Minimization: State Equivalence

- **Equivalent states**: Two states S_i and S_j of a machine M are equivalent iff for every possible input sequence the same output sequence is produced regardless of whether S_i or S_j is the initial state.
- **Theorem**: If two state S_i and S_j in M are distinguishable, then they are distinguishable by a sequence of length n-1, where n is the number of states in M.
  - In other words, if two states are k-equivalent for all $k \leq n-1$, then they are equivalent
- If $S_i = S_j$ and $S_j = S_k$, then $S_i = S_k$. Also, $S_i = S_j \implies S_j = S_i$, and $S_i = S_i$, so this is an *equivalence relation.*
- **Therefore, the set of states can be partitioned into disjoint equivalence classes**
  - This is the key idea used in machine minimization
- If the machine is **completely specified**, (that is, its state transitions and outputs are defined for all inputs) then this equivalence partition is **unique**, —> there is a unique minimal machine.
- But if some of the states and outputs are **not specified**, it is **not unique**
  - Why? Simple — because you have to fill in the unspecified states and outputs (just like don't cares) and you can do it in many ways.

A Simple Special Case

Input Sequence	Present State	Next State $X = 0$	Next State $X = 1$	Present Output $X = 0$	Present Output $X = 1$
reset	*A*	*B*	*C*	0	0
0	*B*	*D*	*E*	0	0
1	*C*	*F*	*G*	0	0
00	*D*	*H*	*I*	0	0
01	*E*	*J*	*K*	0	0
10	*F*	*L*	*M*	0	0
11	*G*	*N*	*P*	0	0
000	*H*	*A*	*A*	0	0
001	*I*	*A*	*A*	0	0
010	*J*	*A*	*A*	0	1
011	*K*	*A*	*A*	0	0
100	*L*	*A*	*A*	0	1
101	*M*	*A*	*A*	0	0
110	*N*	*A*	*A*	0	0
111	*P*	*A*	*A*	0	0

- **Observation**: H, I, K, M, N, P all goes back to A and have the same outputs
- **Observation**: J, L goes back to A and have the same outputs
- So, these states (H, I, K, M, N, P) are clearly equivalent. So is the set (J, L)

A Simple Special Case

Input Sequence	Present State	Next State		Present Output	
		$X = 0$	$X = 1$	$X = 0$	$X = 1$
reset	A	B	C	0	0
0	B	D	E	0	0
1	C	F	G	0	0
00	D	H	I	0	0
01	E	J	K	0	0
10	F	L	M	0	0
11	G	N	P	0	0
000	H	A	A	0	0
001	I	A	A	0	0
010	J	A	A	0	1
011	K	A	A	0	0
100	L	A	A	0	1
101	M	A	A	0	0
110	N	A	A	0	0
111	P	A	A	0	0

Present State	Next State		Present Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
A	B	C	0	0
B	D	E	0	0
C	~~F~~ E	~~G~~ D	0	0
D	H	~~I~~ H	0	0
E	J	~~K~~ H	0	0
~~F~~	~~L J~~	~~M H~~	~~0~~	~~0~~
~~G~~	~~N H~~	~~P H~~	~~0~~	~~0~~
H	A	A	0	0
~~I~~	~~A~~	~~A~~	~~0~~	~~0~~
J	A	A	0	1
~~K~~	~~A~~	~~A~~	~~0~~	~~0~~
~~L~~	~~A~~	~~A~~	~~0~~	~~1~~
~~M~~	~~A~~	~~A~~	~~0~~	~~0~~
~~N~~	~~A~~	~~A~~	~~0~~	~~0~~
~~P~~	~~A~~	~~A~~	~~0~~	~~0~~

A Simple Special Case

Present	Next State		Output	
State	$X = 0$	$X = 1$	$X = 0$	$X = 1$
A	B	C	0	0
B	D	E	0	0
C	E	D	0	0
D	H	H	0	0
E	J	H	0	0
H	A	A	0	0
J	A	A	0	1

- **Remember**: This simple technique only works when the circuit resets to its initial state after receiving a fixed number of inputs. So this is a special case only