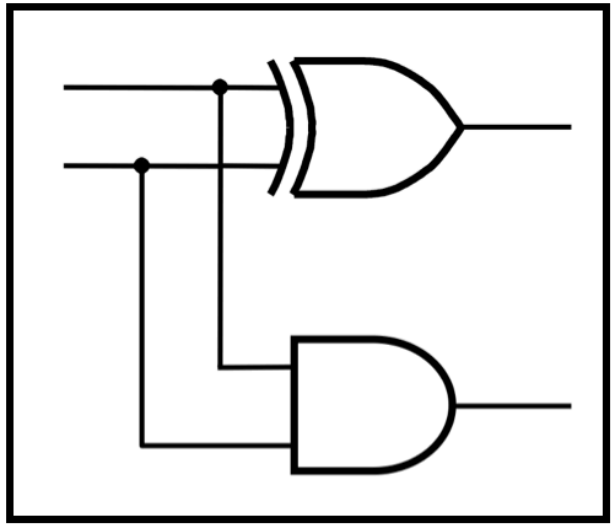


CprE 281: Digital Logic

Instructor: Alexander Stoytchev

http://www.ece.iastate.edu/~alexs/classes/

Review for the Final Exam

CprE 281: Digital Logic
Iowa State University, Ames, IA

Administrative Stuff

- **The FINAL exam is scheduled for**

- **Saturday Nov 21 @ 2:15 – 4:15 PM**

Final Exam Format

- **The exam will cover: Chapter 1 to Chapter 6, and Sections 7.1-7.2, register machines, and i281 CPU**
- **Emphasis will be on Chapter 5, 6, and 7**
- **The exam will be closed book but open notes.**
- **You can bring up to 5 pages of handwritten or typed notes.**

Final Exam Format

- **The exam will be out of 135 points**
- **You need 95 points to get an A on this exam**
- **It will be great if you can score more than 100 points.**
  - **but you can't roll over your extra points ☹**

Topics for the Final Exam

- K-maps for 2, 3, and 4 variables
- Multiplexers (circuits and function)
- Synthesis of logic functions using multiplexers
- Shannon's Expansion Theorem
- 1's complement and 2's complement representation
- Addition and subtraction of binary numbers
- Circuits for adding and subtracting
- Serial adder
- Latches (circuits, behavior, timing diagrams)
- Flip-Flops (circuits, behavior, timing diagrams)
- Counters (up, down, synchronous, asynchronous)
- Registers and Register Files

Topics for the Final Exam

- Synchronous Sequential Circuits
- FSMs
- Moore Machines
- Mealy Machines
- State diagrams, state tables, state-assigned tables
- State minimization
- Designing a counter
- Arbiter Circuits
- Reverse engineering a circuit
- ASM Charts
- Register Machines and programs for them
- ALU, PC, and control for a simple processor (i281 CPU)
- Assembly and machine language (i281 assembly)
- Something from Star Wars and/or the Matrix

How to Study for the Final Exam

- Form a study group
- Go over the slides for this class
- Go over the homeworks again
- Go over the problems at the end of Ch 5 & 6
- Exercise
- Get some sleep

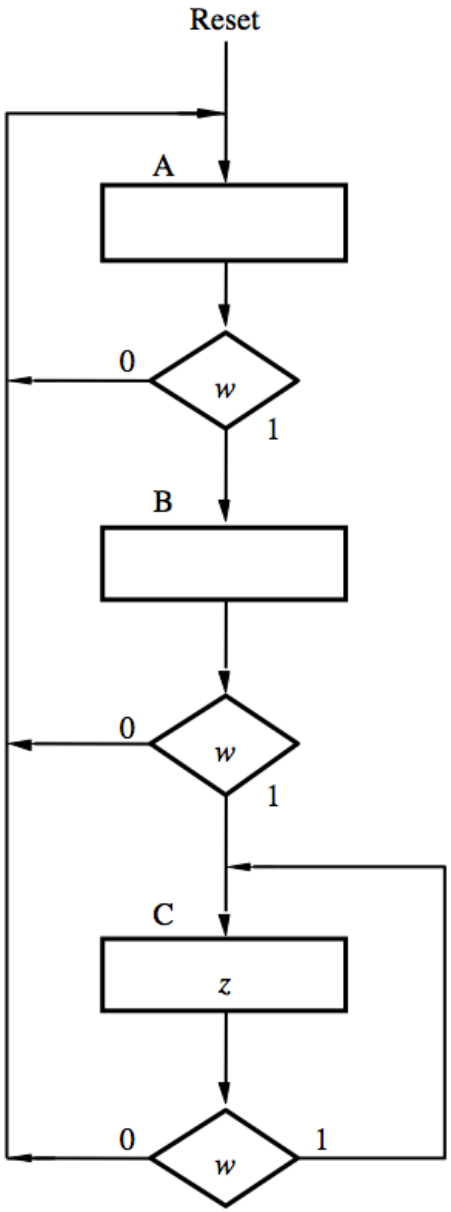
Administrative Stuff

- **Please check your grades on Canvas**
- **Let me know if something is wrong or missing**

Sample Problems

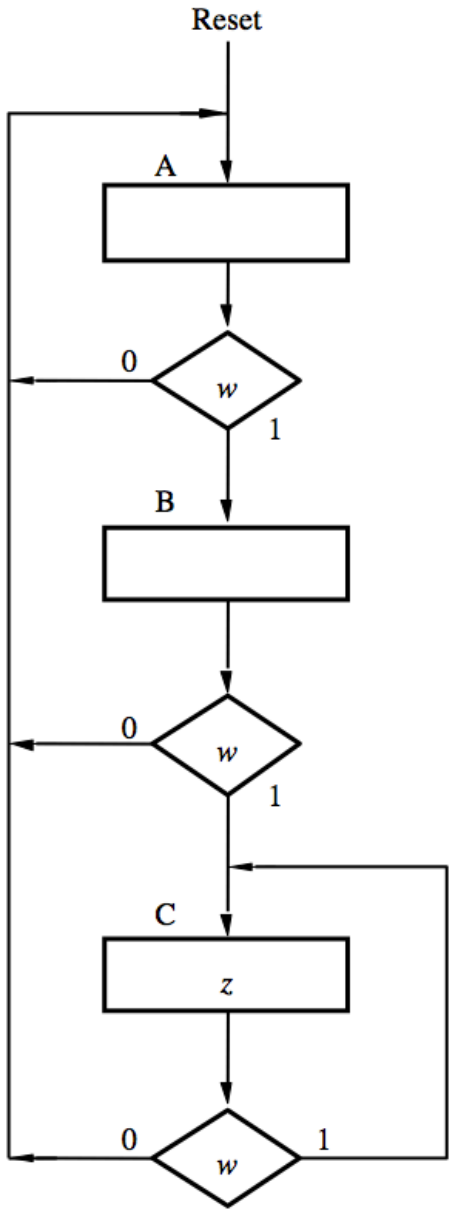
ASM Charts

Given an ASM chart draw the corresponding FSM

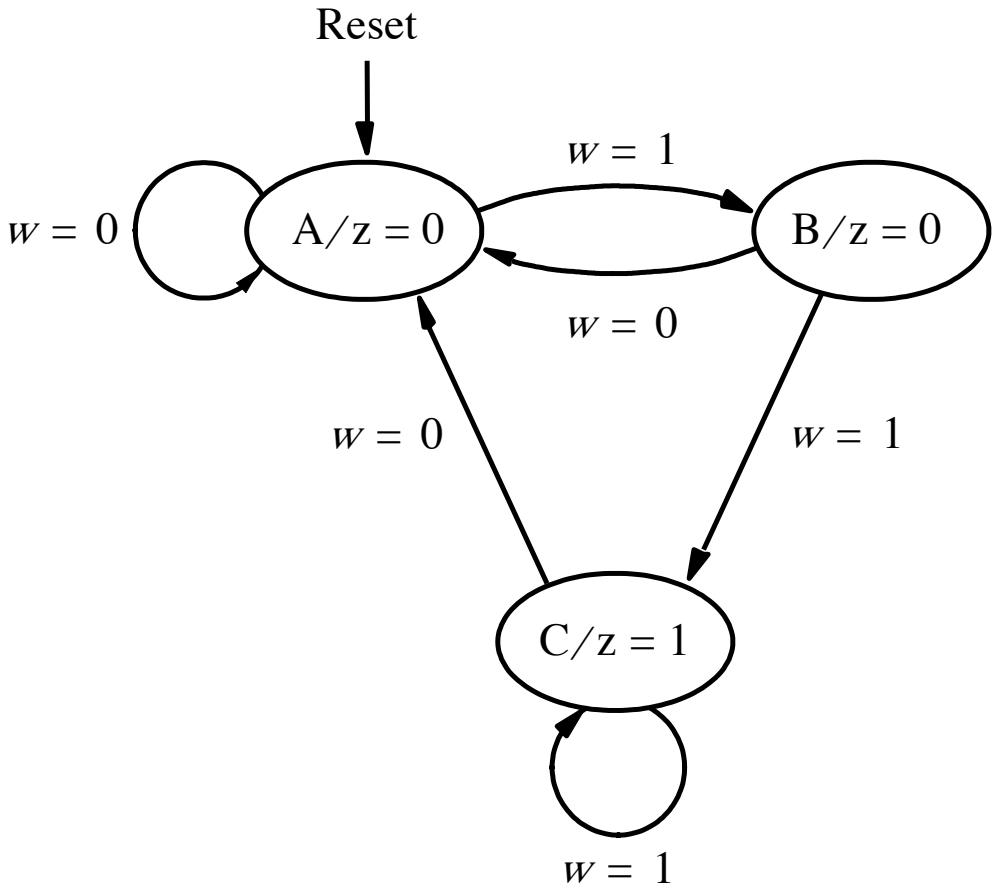


ASM Charts

Given an ASM chart draw the corresponding FSM



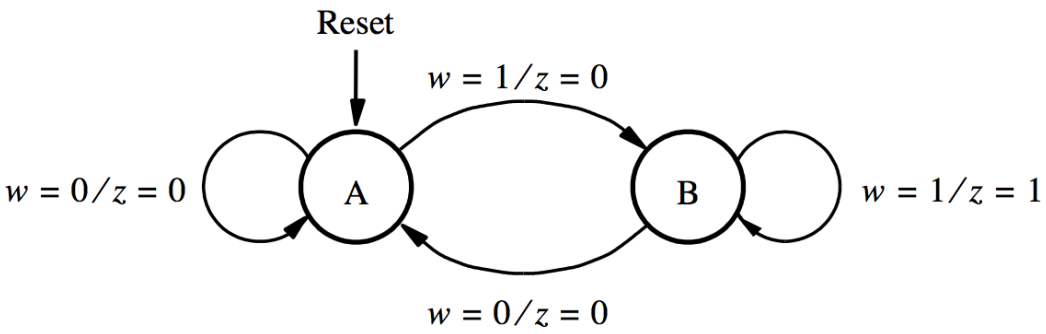
[Figure 6.82 from the textbook]



[Figure 6.3 from the textbook]

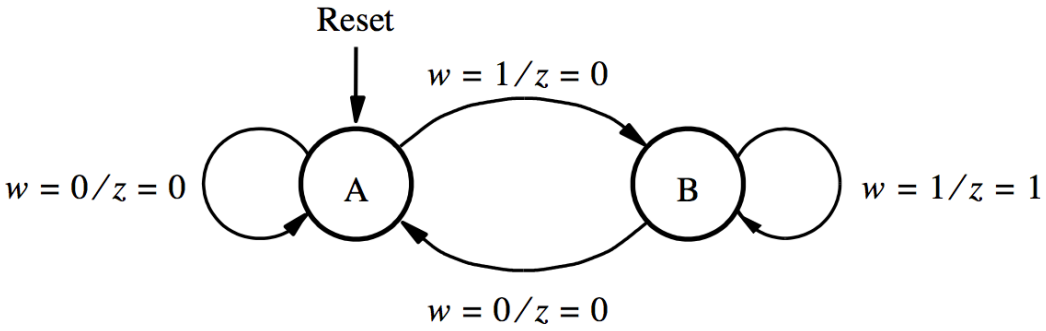
ASM Charts

Given an FSM draw the corresponding ASM Chart

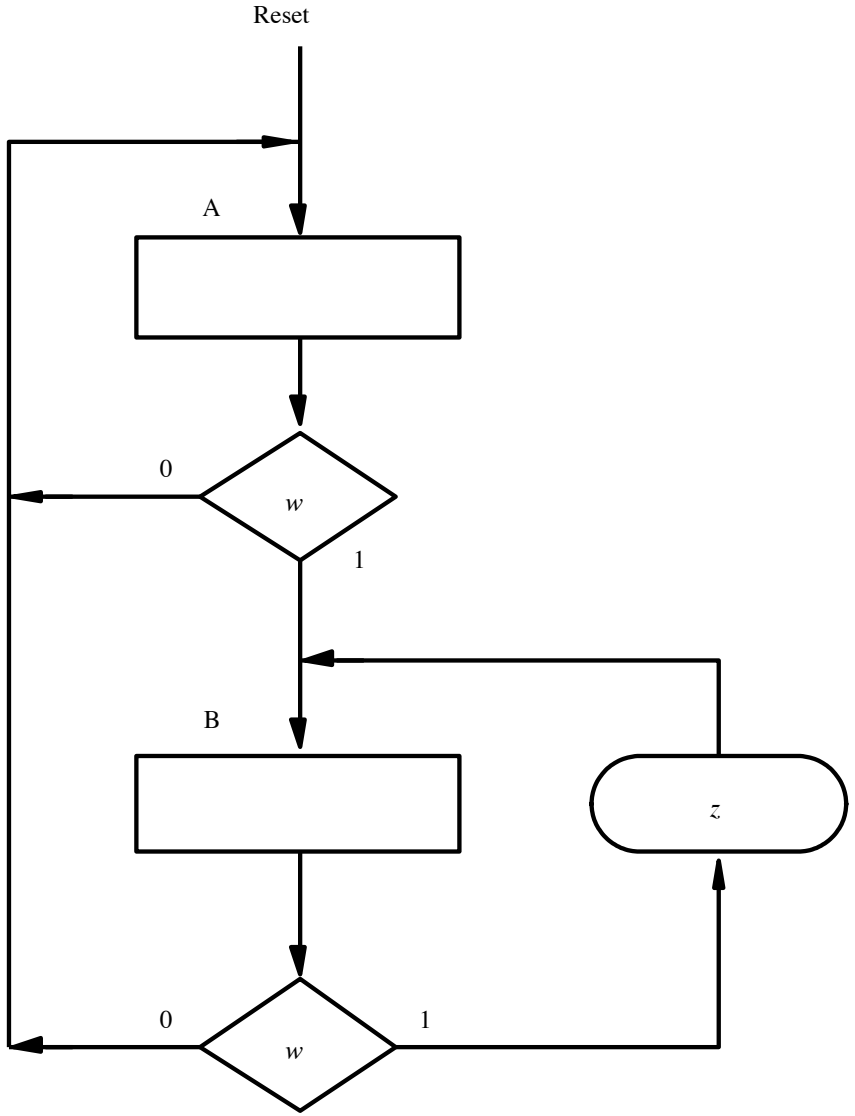


ASM Charts

Given an FSM draw the corresponding ASM Chart



[Figure 6.23 from the textbook]



[Figure 6.83 from the textbook]

Circuit Implementation of FSMs

Implement this state-assigned Table using JK flip-flops

	Present state $y_3y_2y_1$	Next state		Output z
		$w = 0$	$w = 1$	
		$Y_3Y_2Y_1$	$Y_3Y_2Y_1$	
A	000	100	110	0
B	100	101	110	0
C	101	101	110	1
D	110	100	111	0
E	111	100	111	1

Circuit Implementation of FSMs

Implement this state-assigned Table using JK flip-flops

$$J_1 = wy_2 + \overline{w}y_3\overline{y}_2$$

$$K_1 = \overline{w}y_2 + wy_1\overline{y}_2$$

$$J_2 = w$$

$$K_2 = \overline{w}$$

$$J_3 = 1$$

$$K_3 = 0$$

$$z = y_1$$

	Present state $y_3y_2y_1$	Next state $w=0$ $Y_3Y_2Y_1$	Next state $w=1$ $Y_3Y_2Y_1$	Output z
A	000	100	110	0
B	100	101	110	0
C	101	101	110	1
D	110	100	111	0
E	111	100	111	1

Circuit Implementation of FSMs

Implement this state-assigned Table using JK flip-flops

	Present state $y_3y_2y_1$	Flip-flop inputs								Output z
		$w = 0$				$w = 1$				
		$Y_3Y_2Y_1$	J_3K_3	J_2K_2	J_1K_1	$Y_3Y_2Y_1$	J_3K_3	J_2K_2	J_1K_1	
A	000	100	$1d$	$0d$	$0d$	110	$1d$	$1d$	$0d$	0
B	100	101	$d0$	$0d$	$1d$	110	$d0$	$1d$	$0d$	0
C	101	101	$d0$	$0d$	$d0$	110	$d0$	$1d$	$d1$	1
D	110	100	$d0$	$d1$	$0d$	111	$d0$	$d0$	$1d$	0
E	111	100	$d0$	$d1$	$d1$	111	$d0$	$d0$	$d0$	1

Excitation table with JK flip-flops

[Figure 6.94 from the textbook]

Register Machines:

What does this program do?

How many balls are left in each register at the end of the program?

Register 1

Register 2

Register 3

STEP	INSTRUCTION	REGISTER	GO TO STEP	[BRANCH TO STEP]
1.	Deb	3	1	2
2.	Deb	2	3	4
3.	Inc	3	2	
4.	End			

Register Machines:

Move the contents of register 2 to register 3

Register 1 Register 2 Register 3

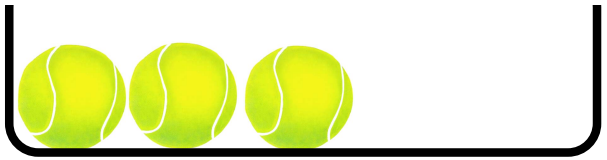
STEP	INSTRUCTION	REGISTER	GO TO STEP	[BRANCH TO STEP]
1.	Deb	3	1	2
2.	Deb	2	3	4
3.	Inc	3	2	
4.	End			

Register Machines:

What does this program do?

How many balls are left in each register at the end of the program?

Register 1



Register 2

Register 3

STEP	INSTRUCTION	REGISTER	GO TO STEP	[BRANCH TO STEP]
1.	Deb	3	1	2
2.	Deb	2	2	3
3.	Deb	1	4	6
4.	Inc	3	5	
5.	Inc	2	3	
6.	Deb	2	7	8
7.	Inc	1	6	
8.	End			

Register Machines:

Copy the contents of register 1 to register 3 using register 2 as a temporary storage

Register 1

Register 2

Register 3

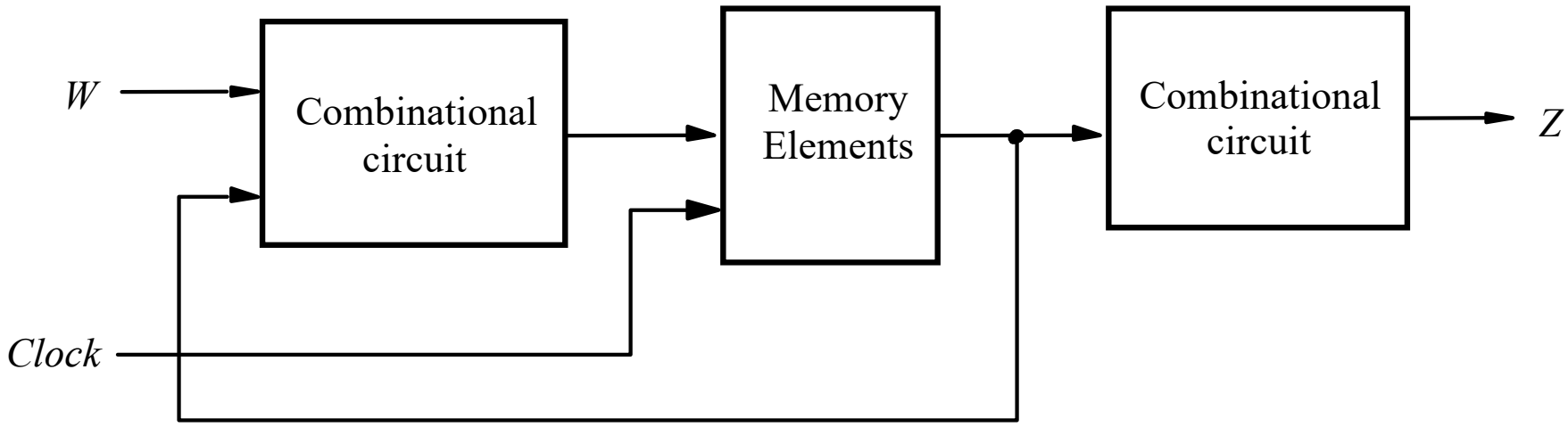
STEP	INSTRUCTION	REGISTER	GO TO STEP	[BRANCH TO STEP]
1.	Deb	3	1	2
2.	Deb	2	2	3
3.	Deb	1	4	6
4.	Inc	3	5	
5.	Inc	2	3	
6.	Deb	2	7	8
7.	Inc	1	6	
8.	End			

Moore Machine Implementation

The state diagram is just an illustration to help us describe and reason about how the FSM will behave in each of its states.

So, how do we turn it into a circuit?

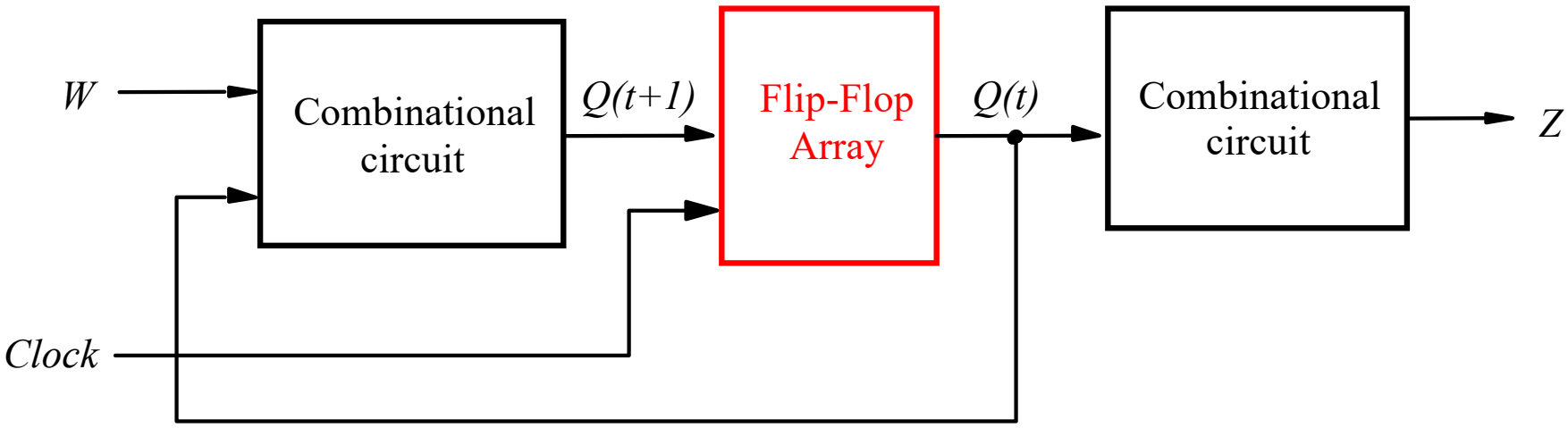
Moore Machine Implementation



Note: The *W* and *Z* lines need not be wires. They can be buses.

[Figure 6.1 from the textbook]

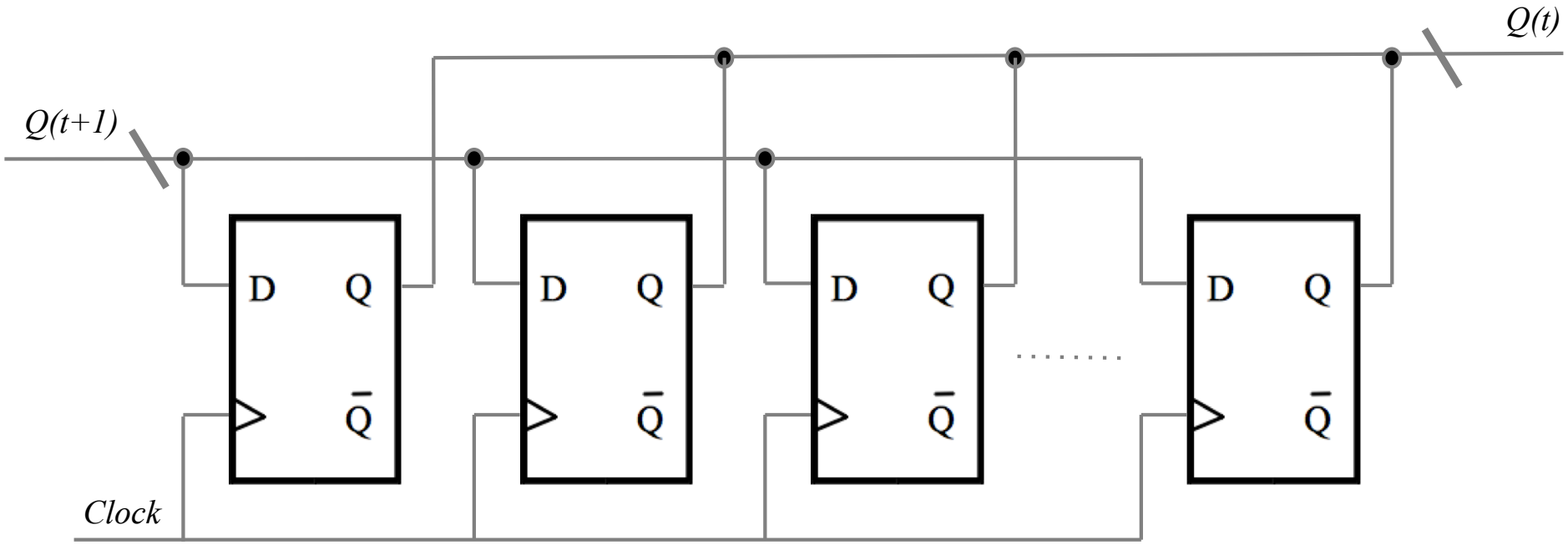
State Storage



Any usable "memory" of the preceding input sequence is encoded in the flip-flop array.

FSM States

The Flip-Flop array stores an encoding of the current state.



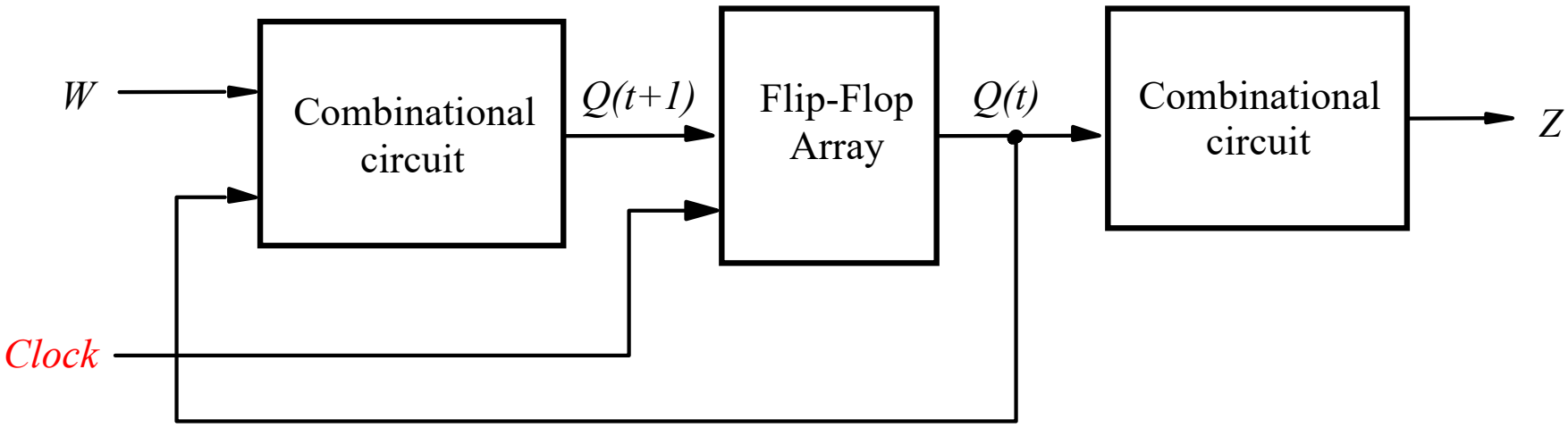
State Encoding

Each of the states in our design is identified by a distinct code.

If we use *3* flip-flops, then the FSM can have up to $2^3 = 8$ distinct states.

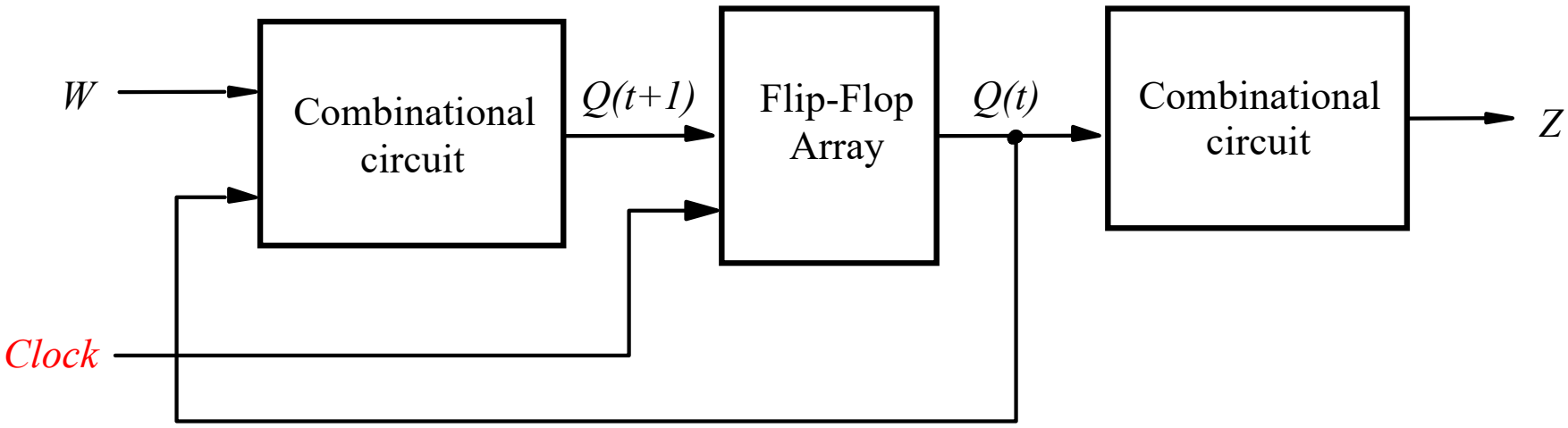
So, when the flip-flop array contains the code *011*, we say that the machine is in state *011*.

Synchronous Design



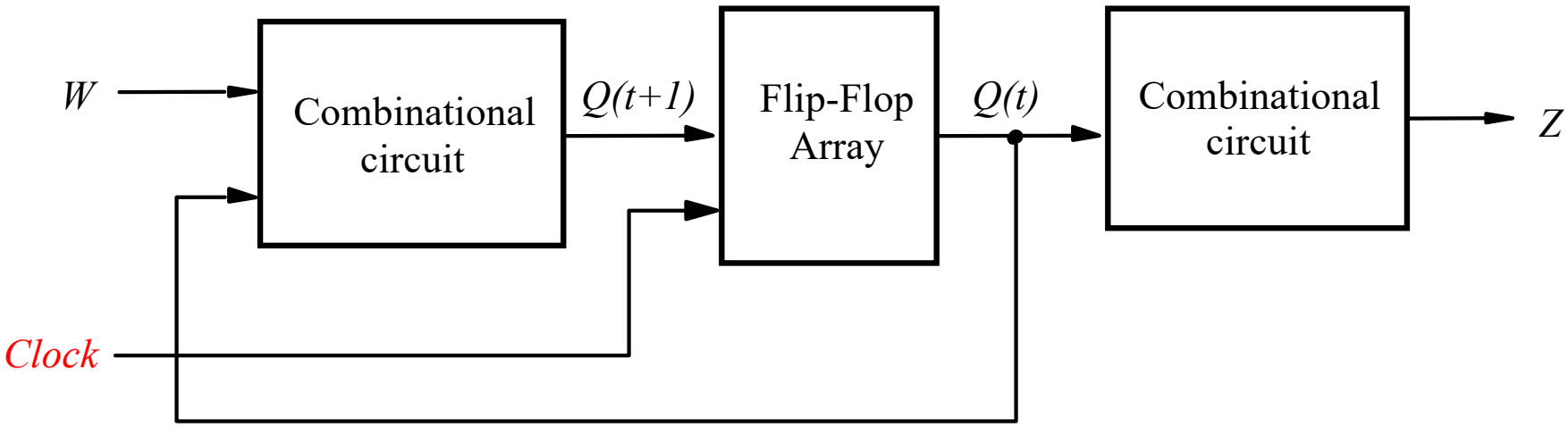
Every *active clock edge* causes a state transition.

Synchronous Design



We expect the input signals to be stable before the *active clock edge* occurs.

Synchronous Design



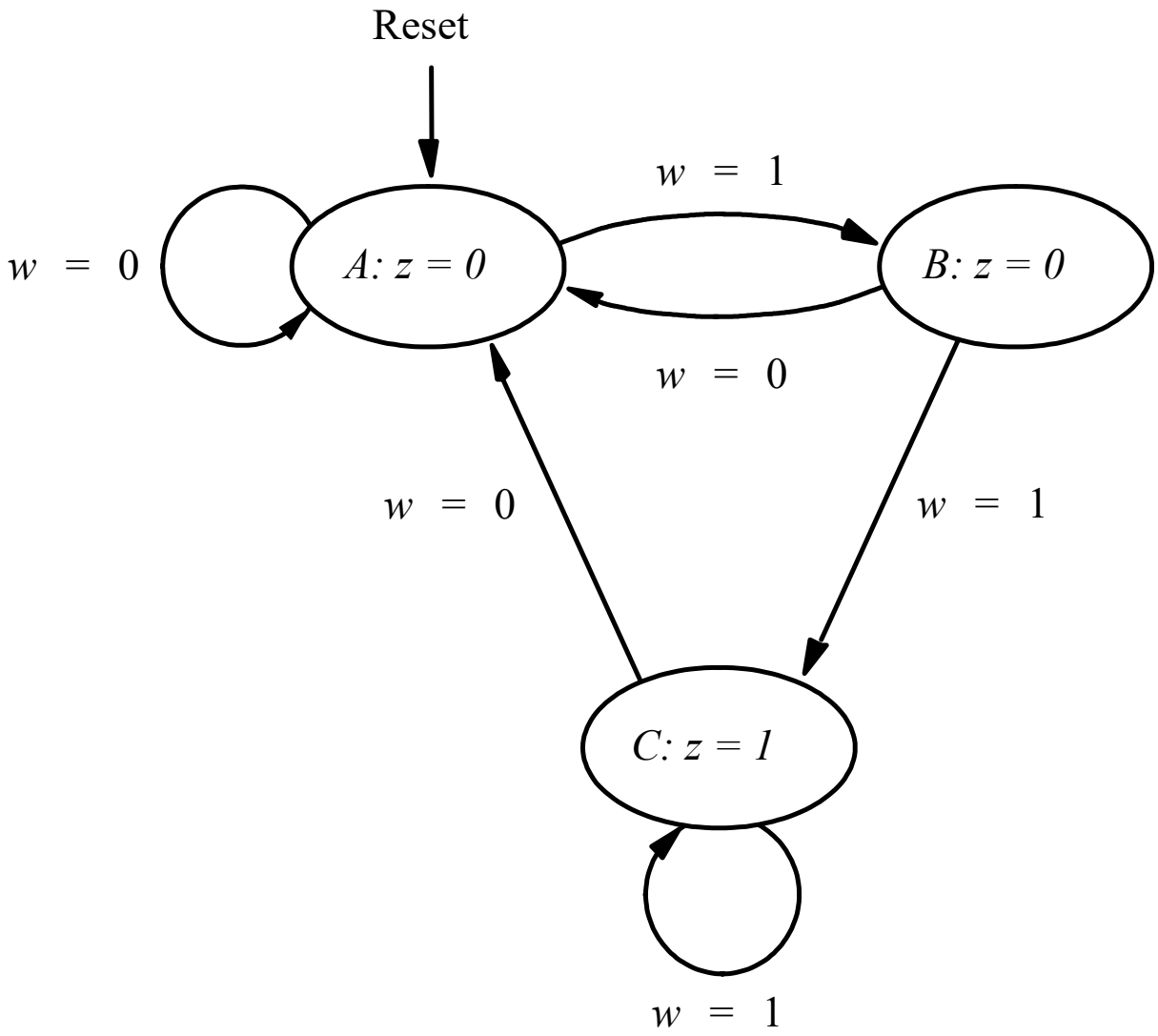
There is a whole other class of sequential circuits that are asynchronous, but we will not study them in this course.

Sequential Circuits: Key Ideas

The current output depends on something about the preceding sequence of inputs (and maybe the current output).

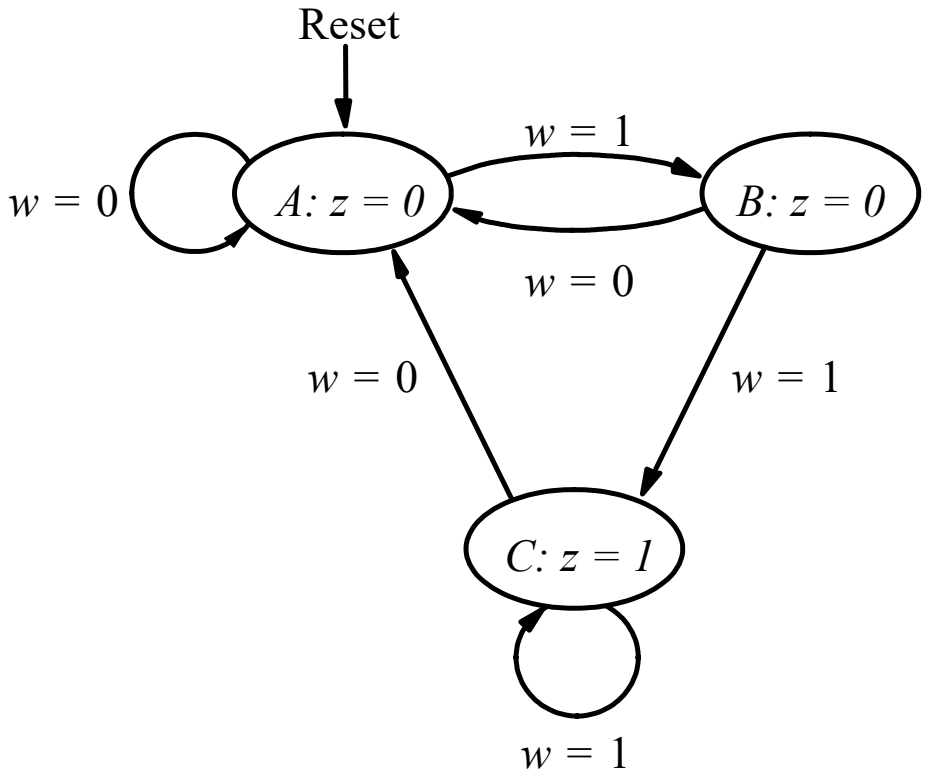
Using *memory elements* (i.e., flip-flops), we design the circuit to remember some *relevant* information about the prior inputs.

Example

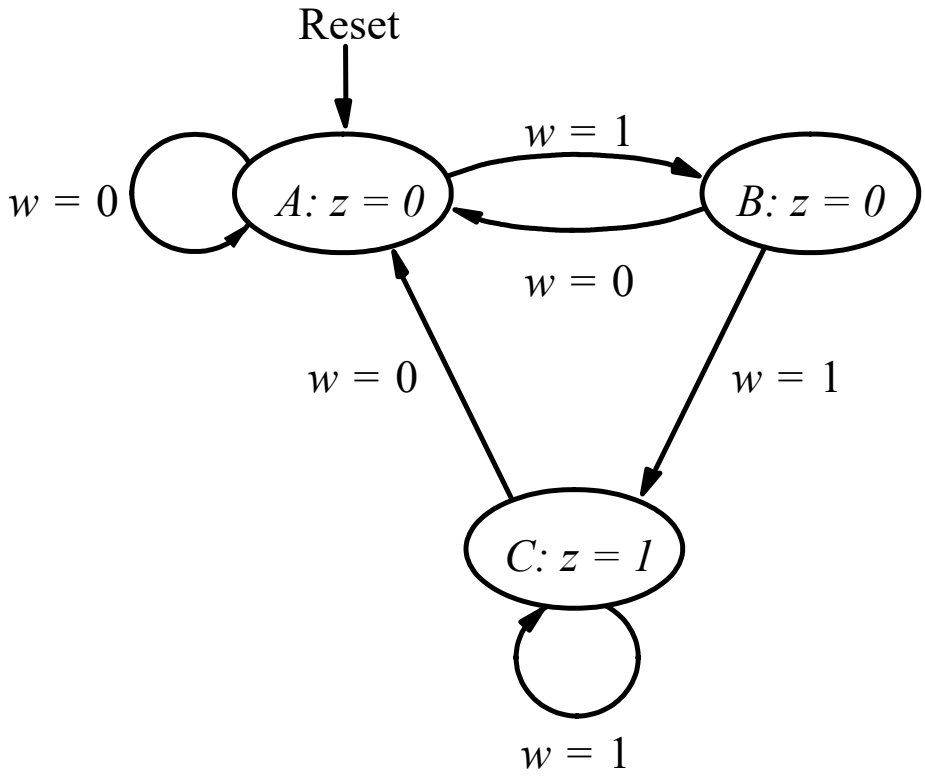


We need to find both the *next state logic* and the *output logic* implied by this machine.

[Figure 6.3 from the textbook]



Present state	Next state		Output
	$w = 0$	$w = 1$	z
A			
B			
C			



Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

[Figure 6.4 from the textbook]

How to represent the States?

One way is to encode each state with a 2-bit binary number

A ~ 00

B ~ 01

C ~ 10

How to represent the states?

One way is to encode each state with a 2-bit binary number

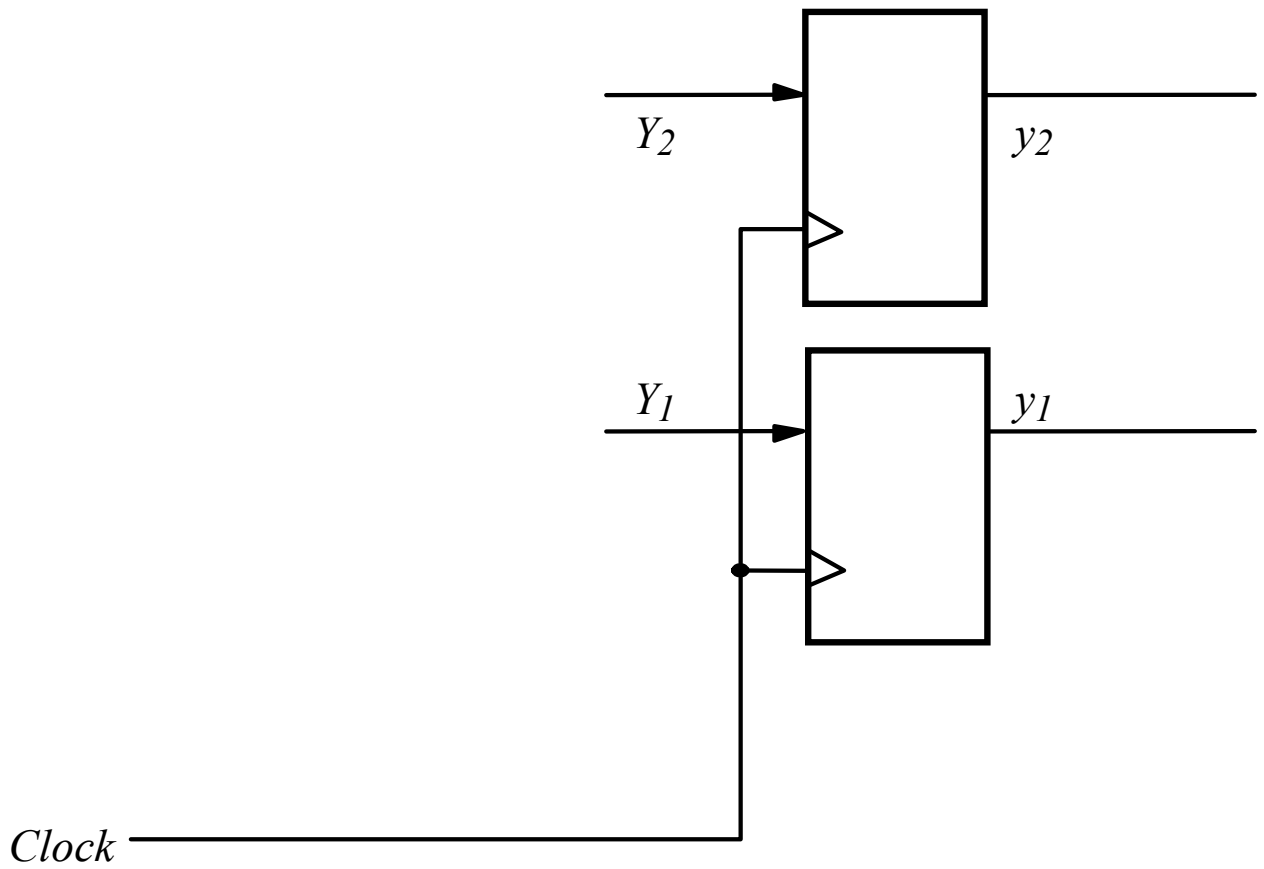
A ~ 00

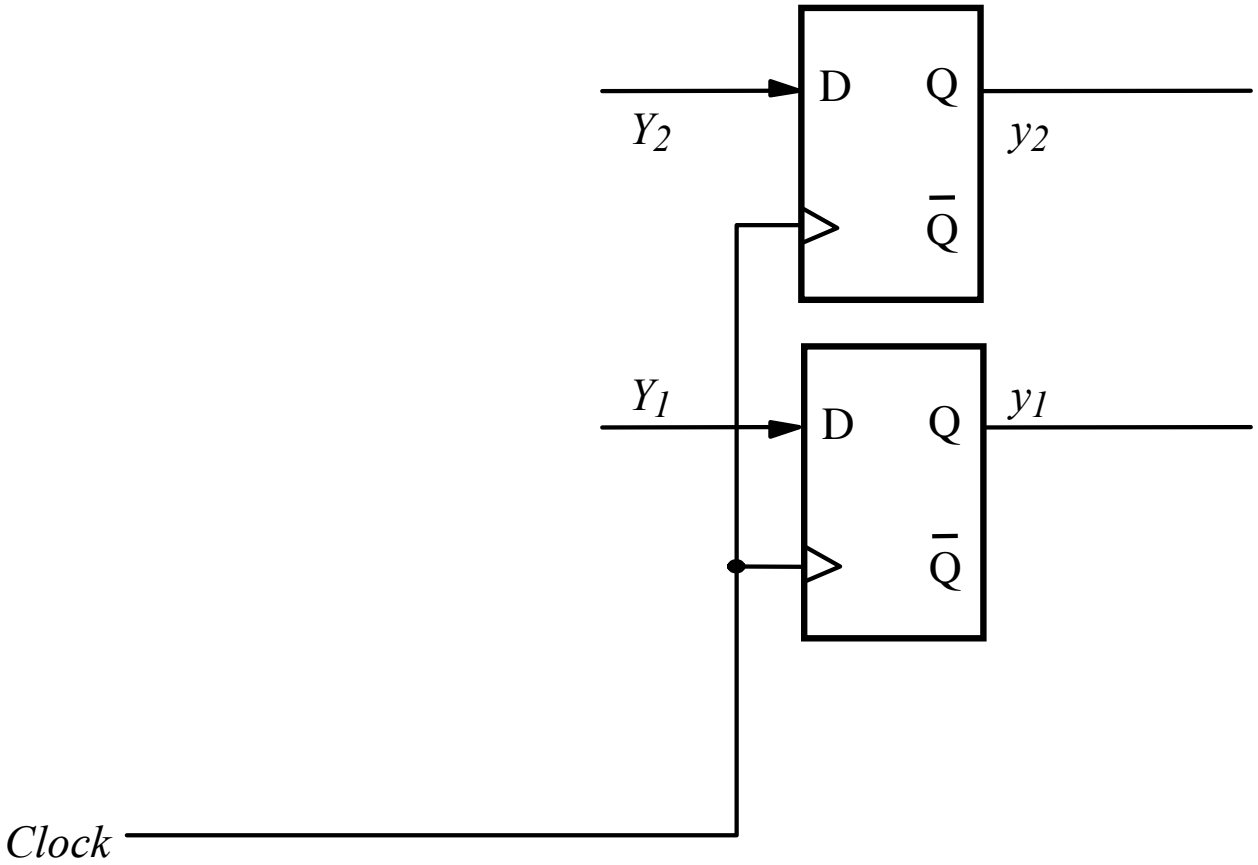
B ~ 01

C ~ 10

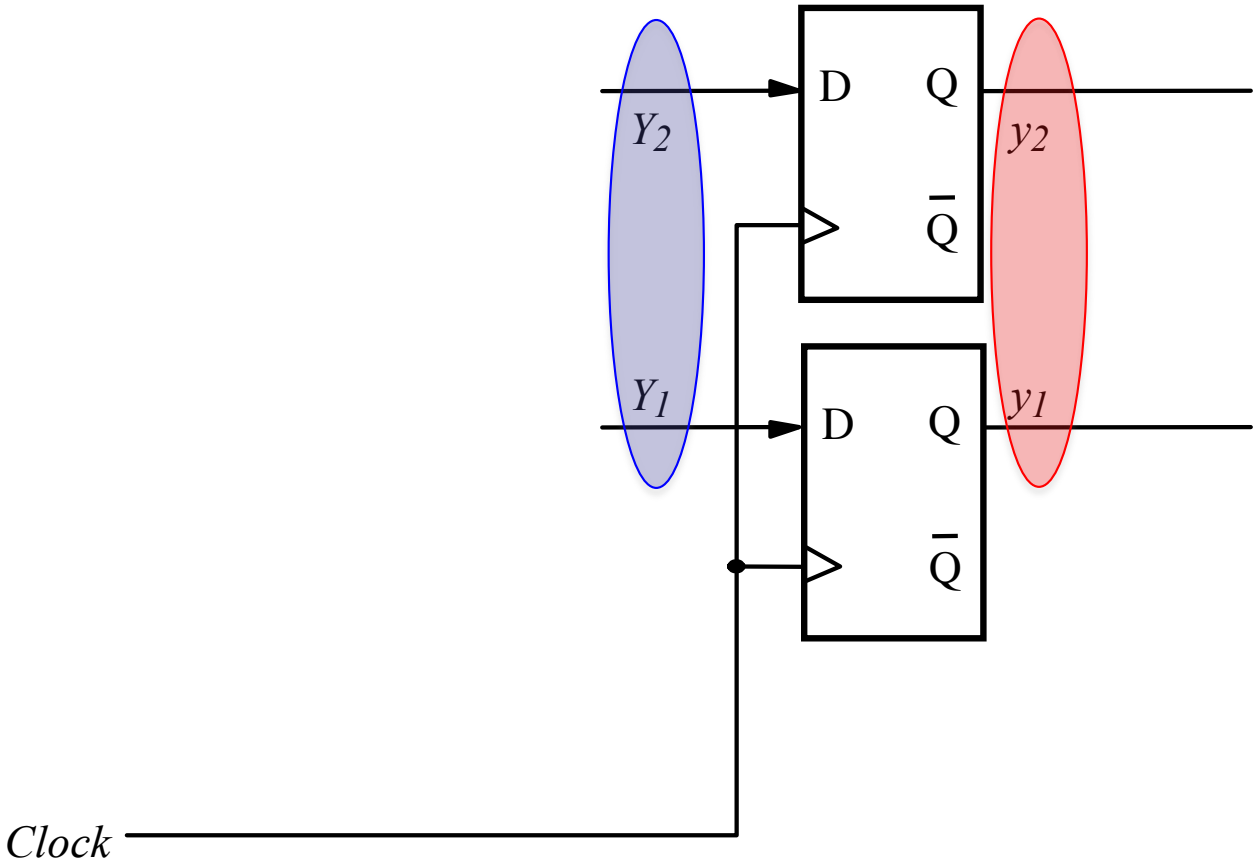
How many flip-flops do we need?

Let's use two flip-flops to hold the machine's state


Y_2
y_2
Y_1
y_1
Clock



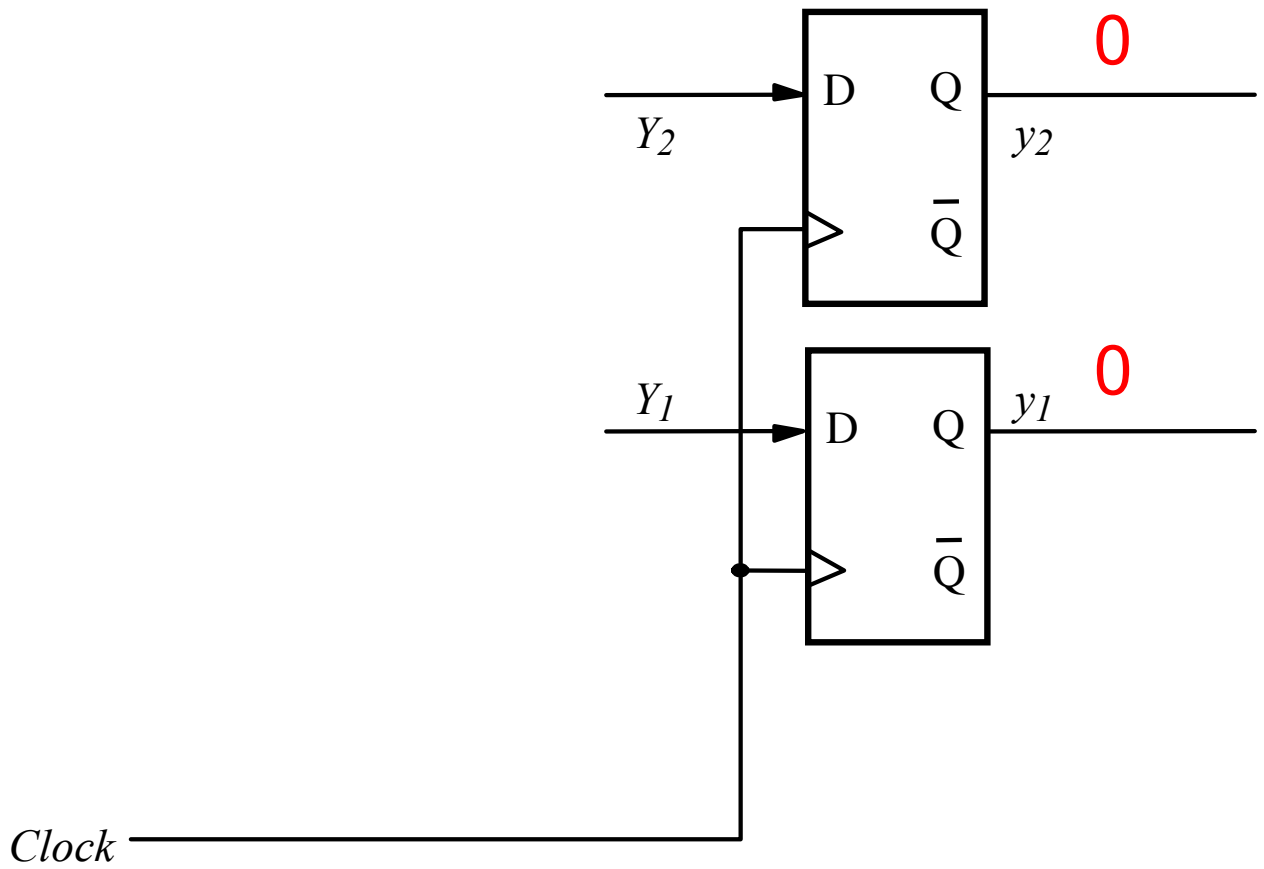
Let’s pick D Flip-Flops.



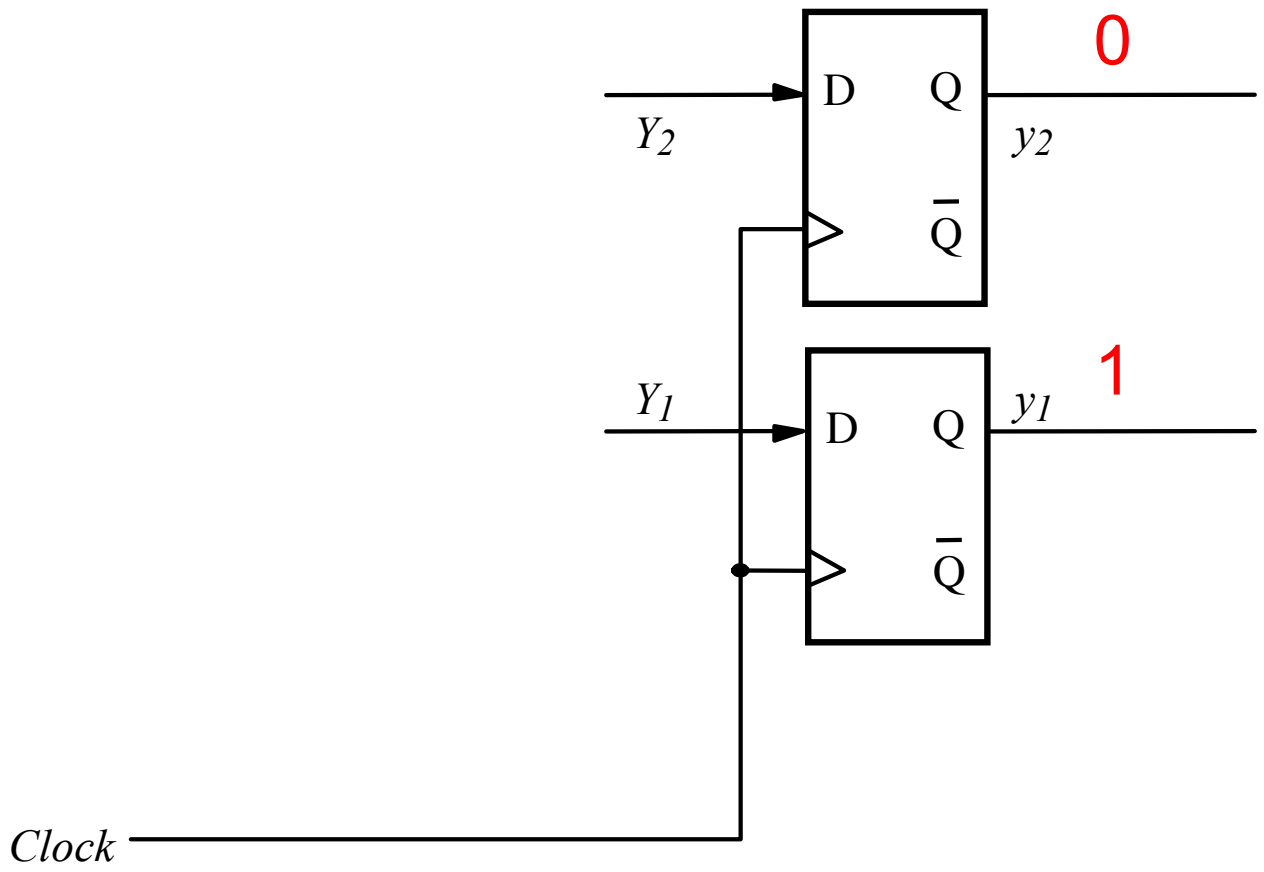
- **We will call y_1 and y_2 the *present state variables*.**

- **We will call Y_1 and Y_2 the *next state variables*.**

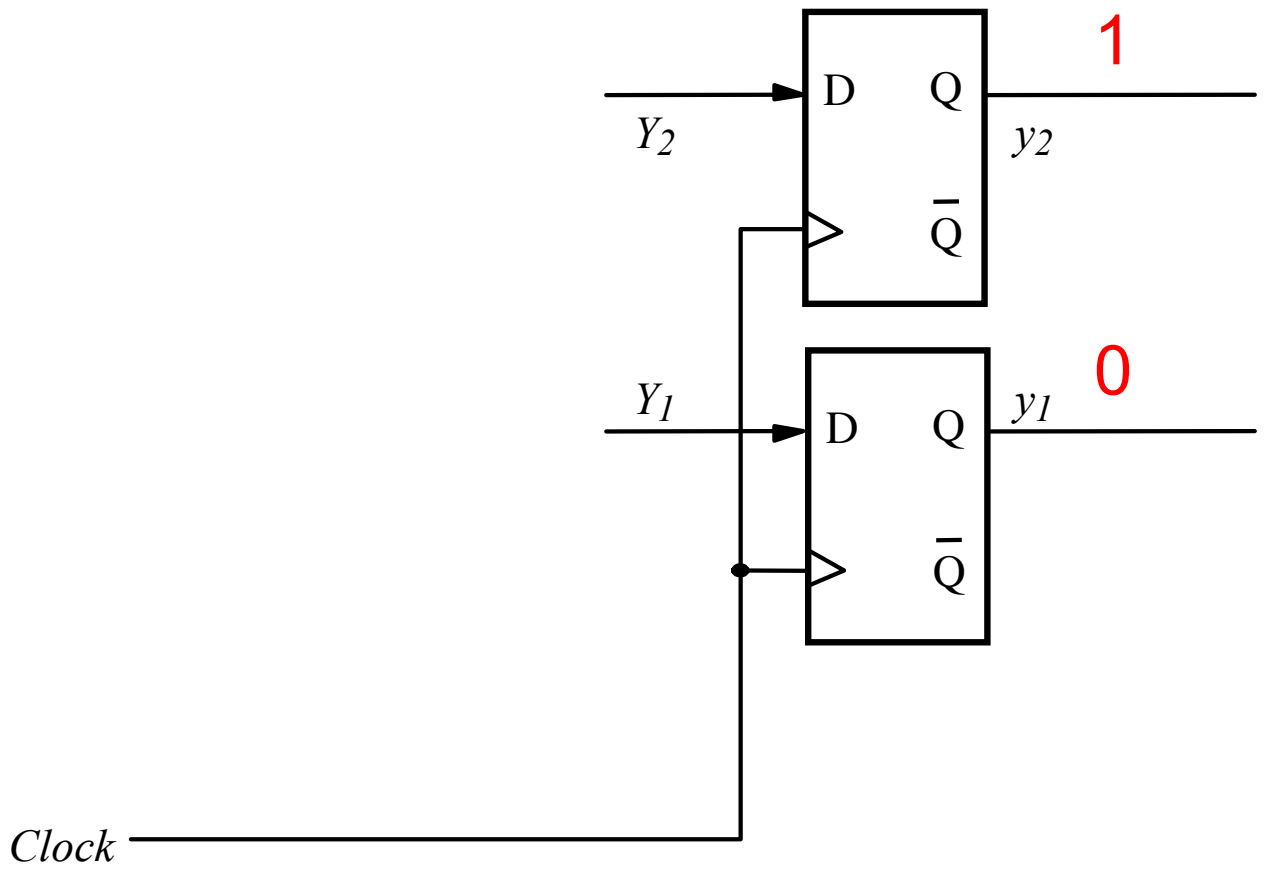
[Figure 6.5 from the textbook]



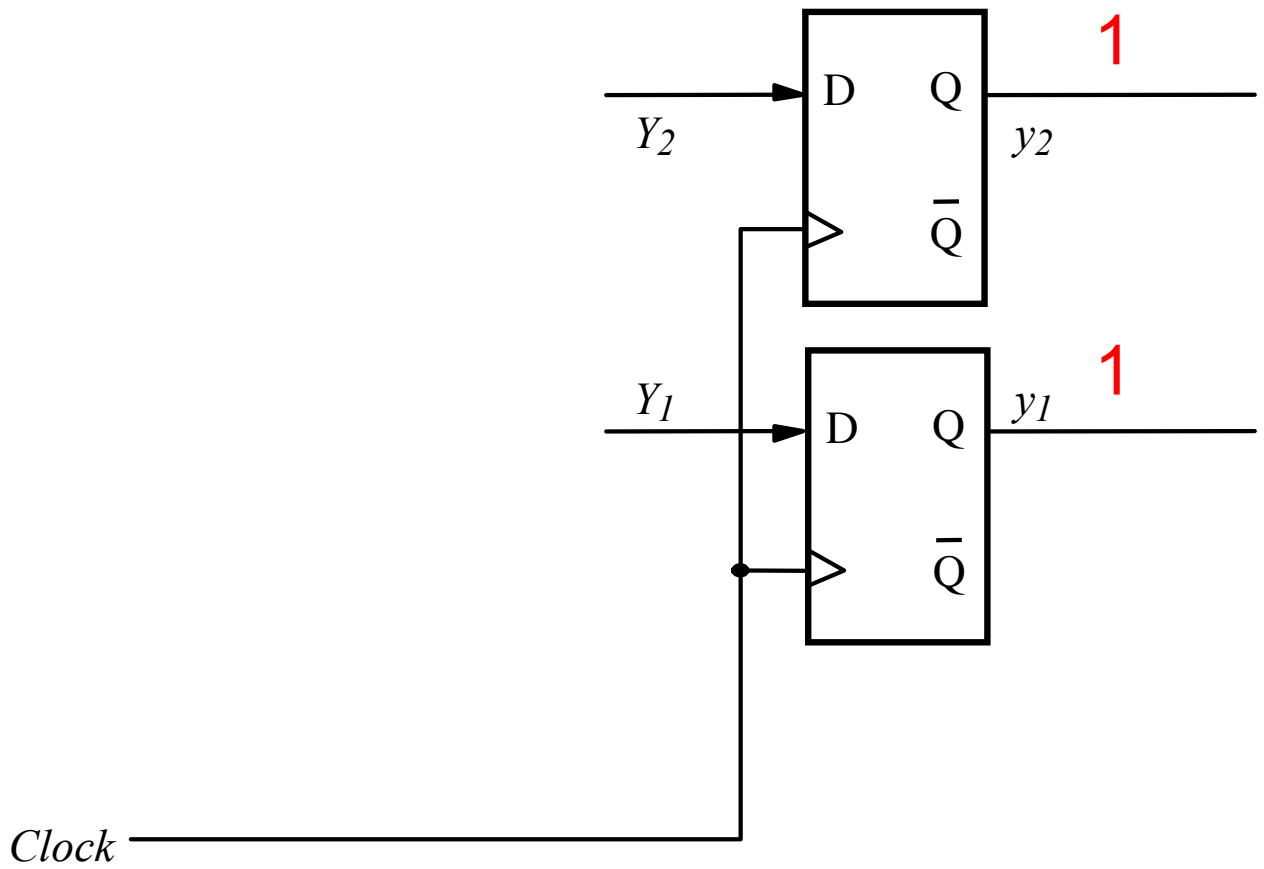
- **Two zeros on the output JOINTLY represent state A.**



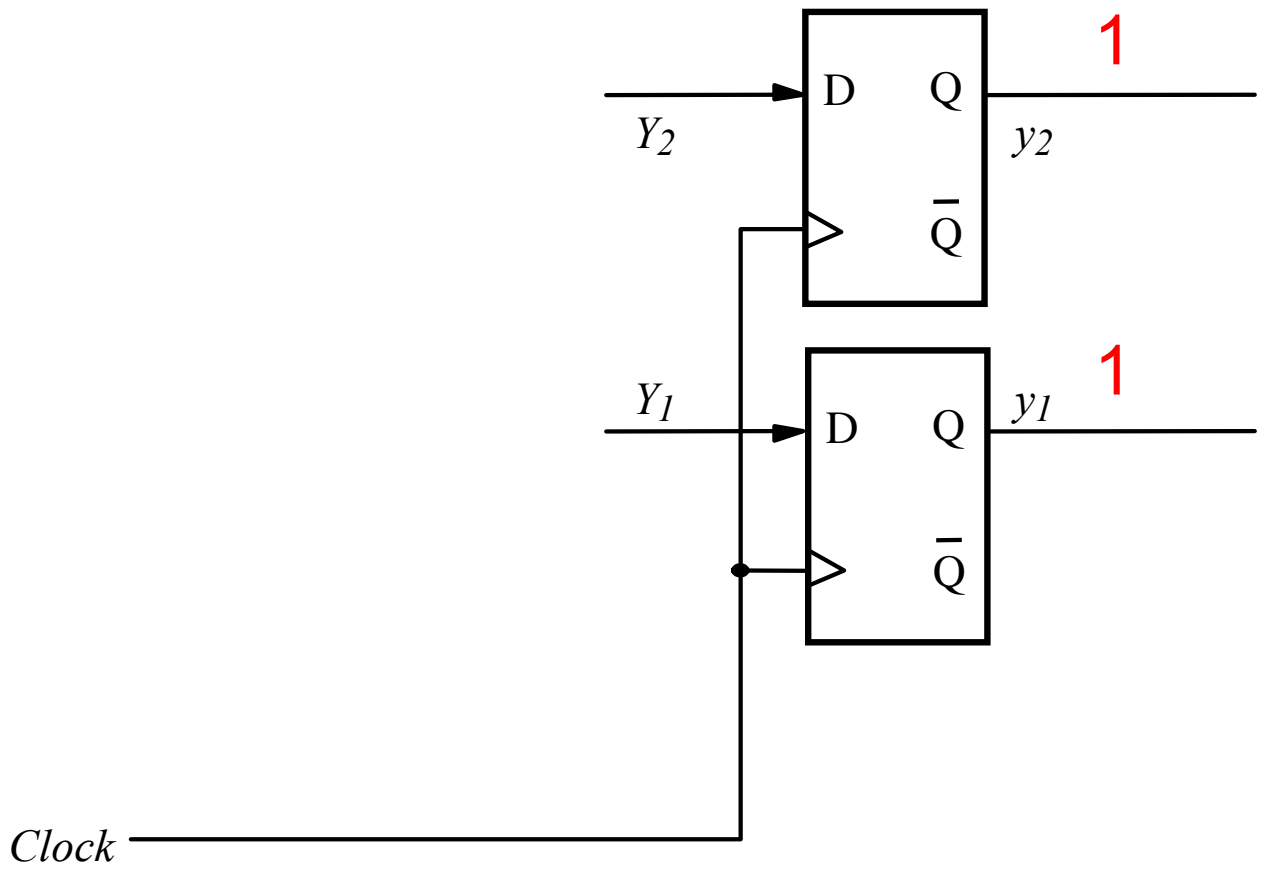
- **This flip-flop output pattern represents state B.**



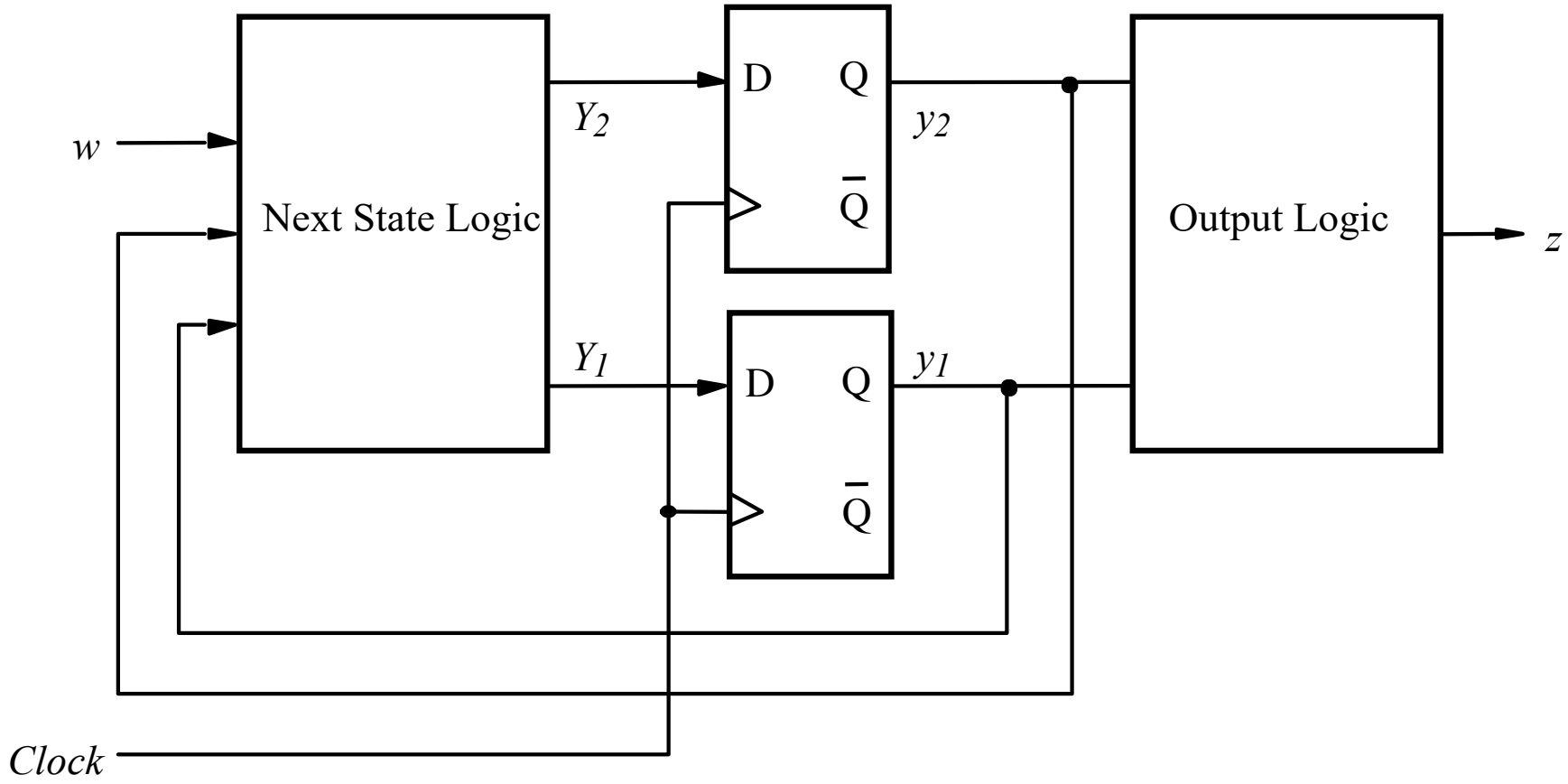
- **This flip-flop output pattern represents state C.**



- **What does this flip-flop output pattern represent?**



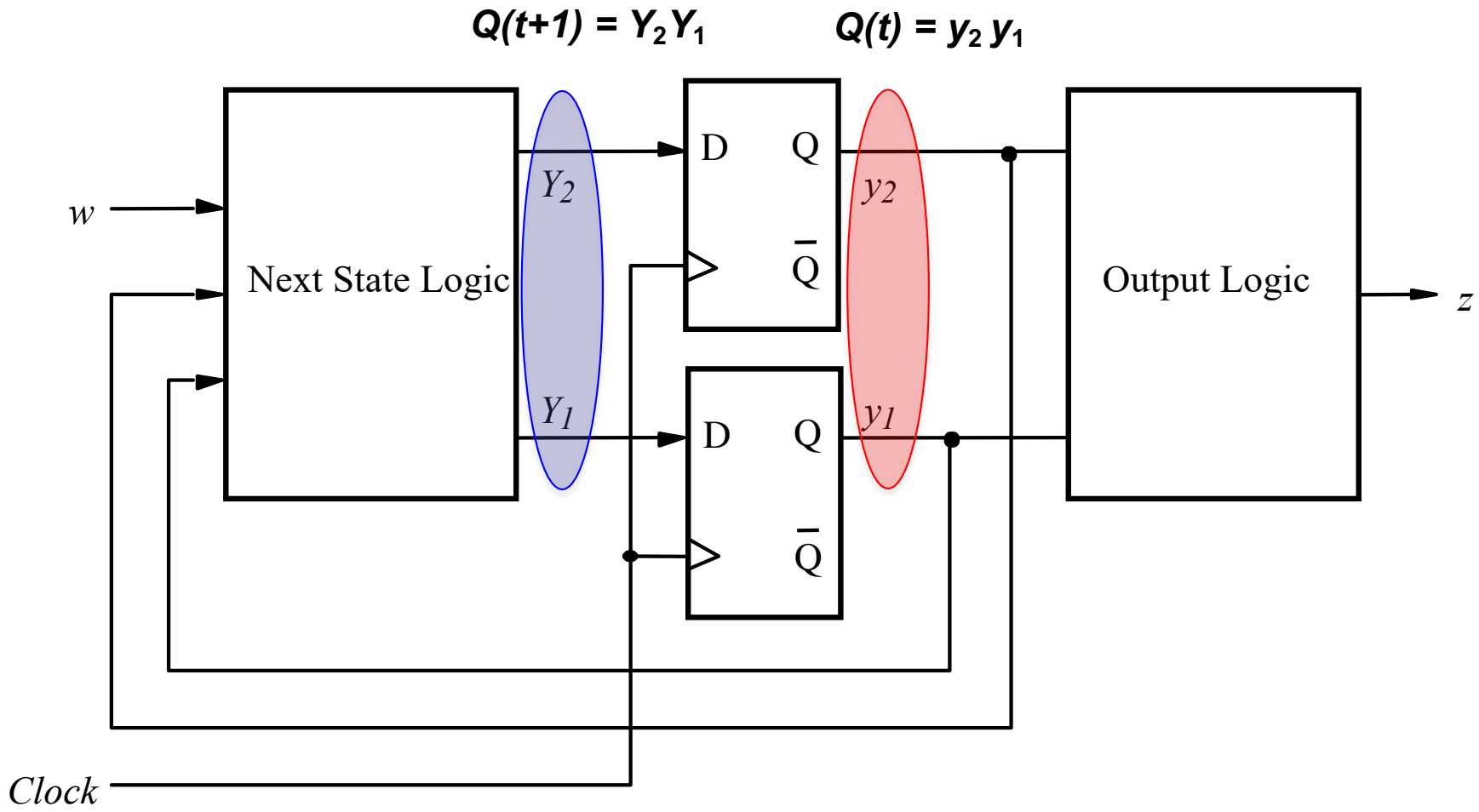
- **This would be state D, but we don't have one**
- **in this example. So this is an impossible state.**



- **We will call y_1 and y_2 the *present state variables*.**

- **We will call Y_1 and Y_2 the *next state variables*.**

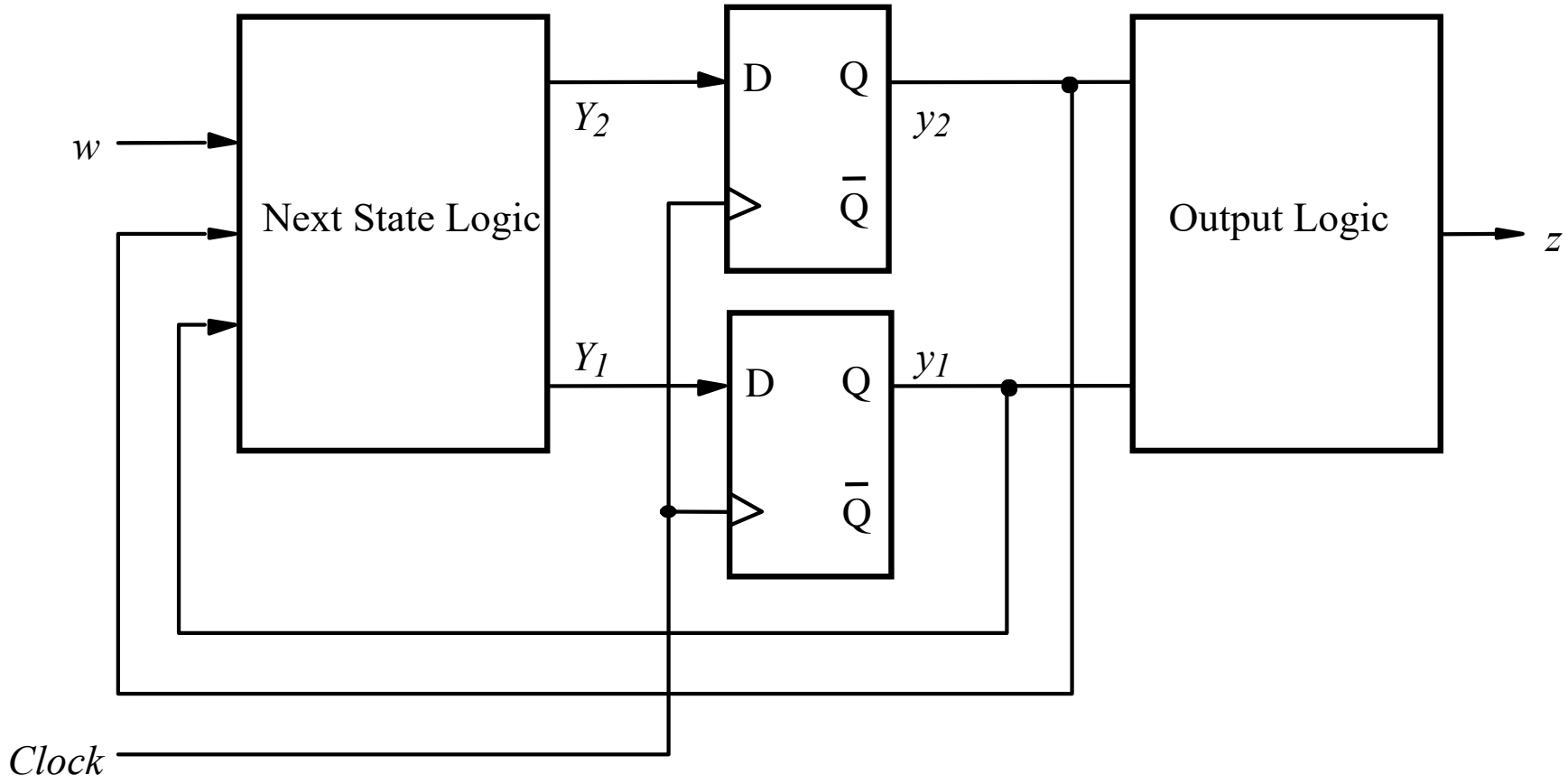
[Figure 6.5 from the textbook]



- **We will call y_1 and y_2 the *present state variables*.**

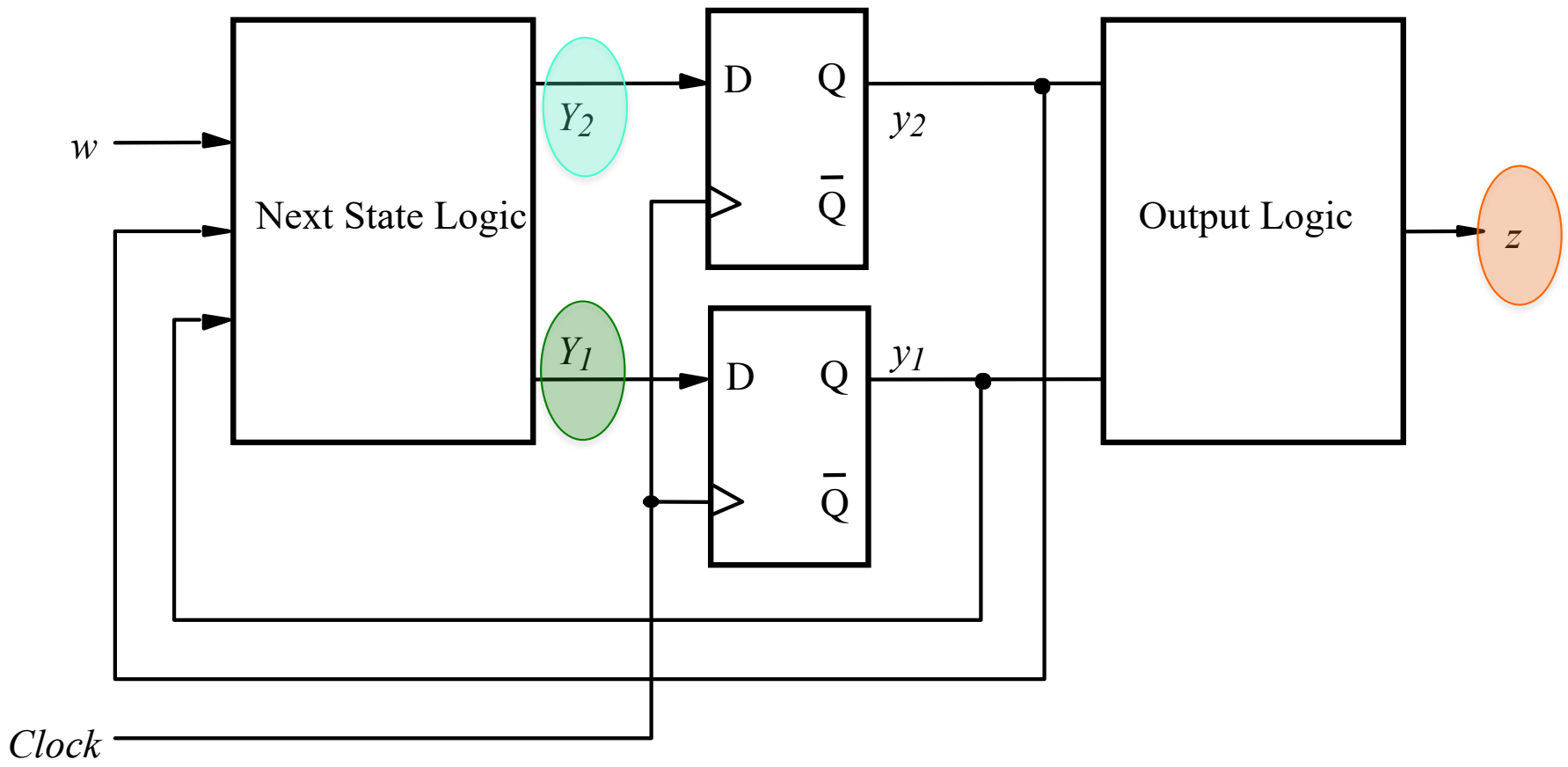
- **We will call Y_1 and Y_2 the *next state variables*.**

[Figure 6.5 from the textbook]



- **We need to find logic expressions for**
- **$Y_1(w, y_1, y_2)$, $Y_2(w, y_1, y_2)$, and $z(y_1, y_2)$.**

[Figure 6.5 from the textbook]



- We need to find logic expressions for
- $Y_1(w, y_1, y_2)$, $Y_2(w, y_1, y_2)$, and $z(y_1, y_2)$.

[Figure 6.5 from the textbook]

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	A	B	0
B	A	C	0
C	A	C	1

- **Suppose we encoded our states in the same order in which they were labeled:**

- **A ~ 00**
- **B ~ 01**
- **C ~ 10**

[Figure 6.4 from the textbook]

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	A	B	0
B	A	C	0
C	A	C	1

	Present state	Next state		Output z
		$w = 0$	$w = 1$	
A	00			
B	01			
C	10			
	11			

The finite state machine will never reach a state encoded as 11.

[Figure 6.6 from the textbook]

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

	Present state	Next state		Output
		$w = 0$	$w = 1$	z
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

We arbitrarily chose these as our state encodings. We could have used others.

[Figure 6.6 from the textbook]

Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	
0	1	
1	0	
1	1	

Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2 y_1$ and $Q(t+1) = Y_2 Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	d	
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2 y_1$ and $Q(t+1) = Y_2 Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	d	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	d	

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2 y_1$ and $Q(t+1) = Y_2 Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	d	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	d	

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2 y_1$ and $Q(t+1) = Y_2 Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	d	

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	d	d

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

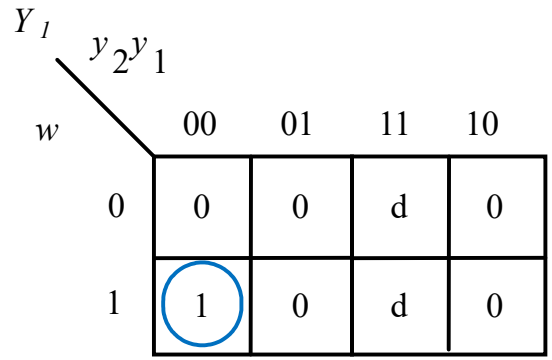
$Q(t) = y_2 y_1$ and $Q(t+1) = Y_2 Y_1$

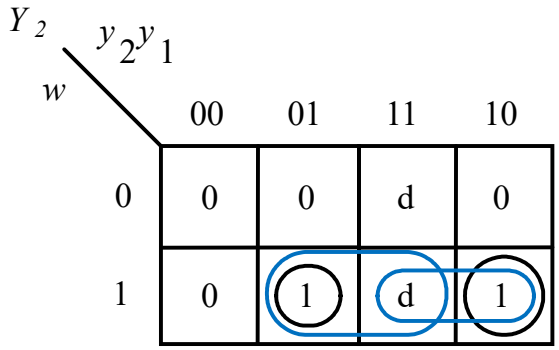
w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	d	d

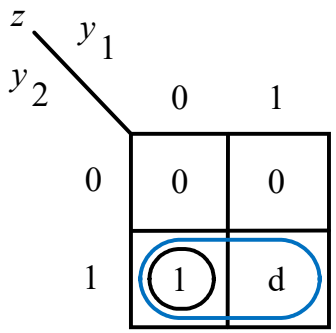
y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Note that the textbook draws these K-Maps differently from all previous K-maps (the most significant bit indexes the rows).

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$



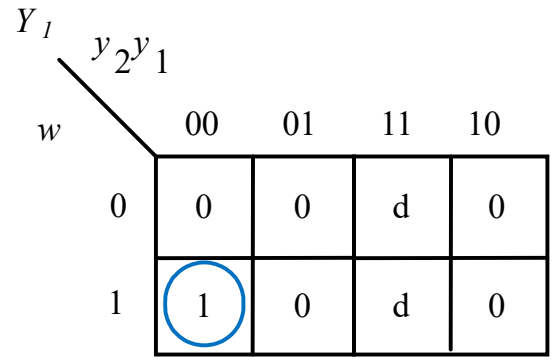


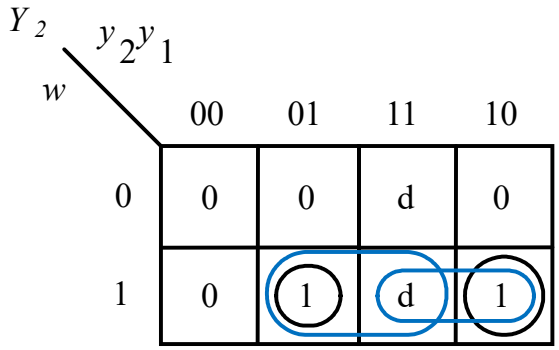


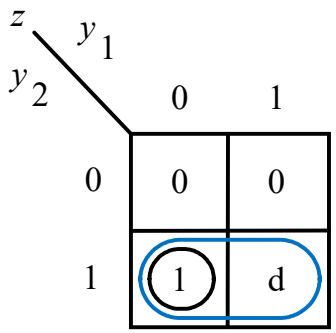
w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	d	d

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

•Don’t care conditions simplify the combinatorial logic







Ignoring don't cares

$Y_1 = w\bar{y}_1\bar{y}_2$

$Y_2 = wy_1\bar{y}_2 + \bar{w}y_1y_2$

$z = \bar{y}_1y_2$

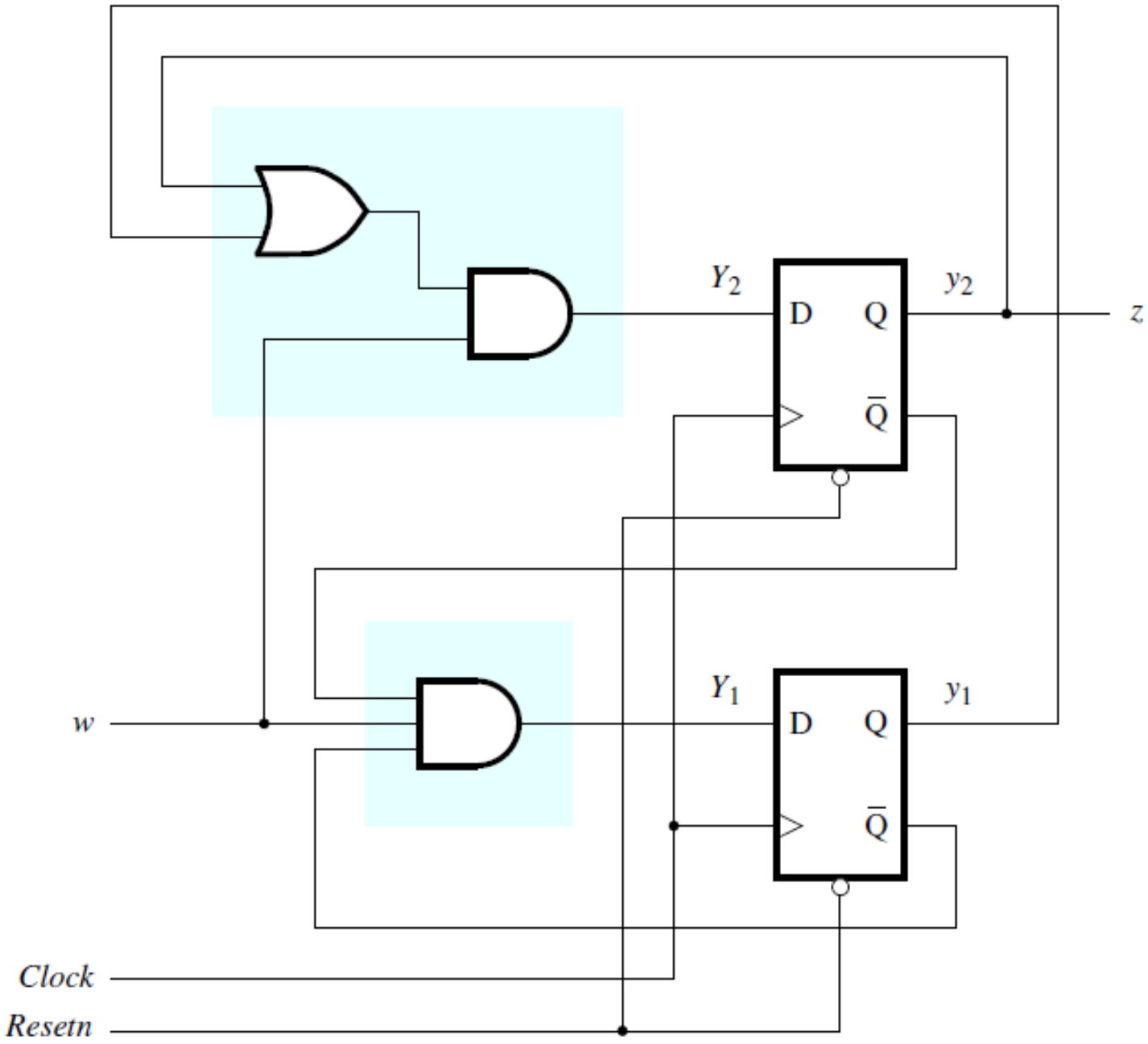
Using don't cares

$Y_1 = w\bar{y}_1\bar{y}_2$

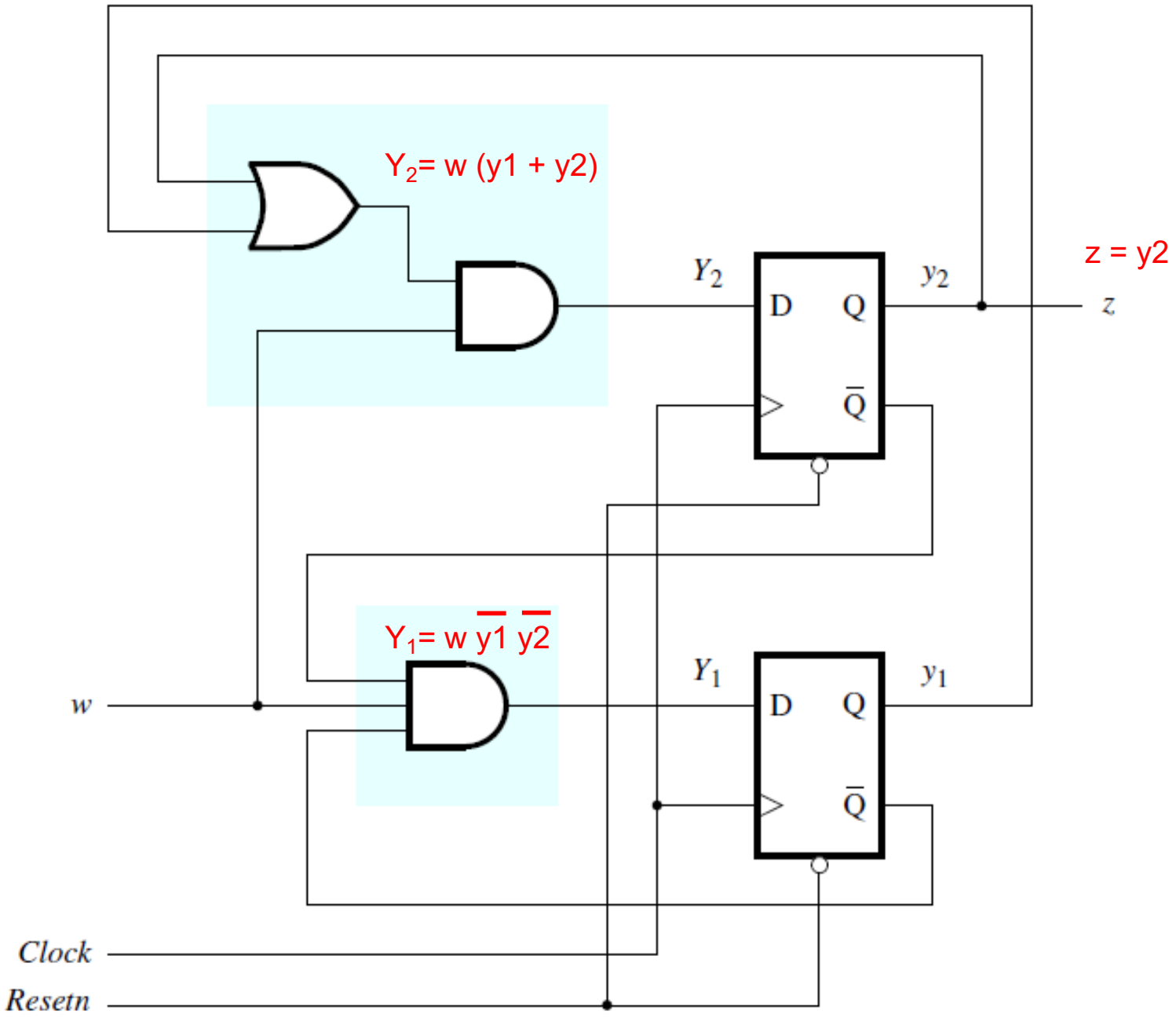
$Y_2 = wy_1 + wy_2$
$= w(y_1 + y_2)$

$z = y_2$

[Figure 6.7 from the textbook]

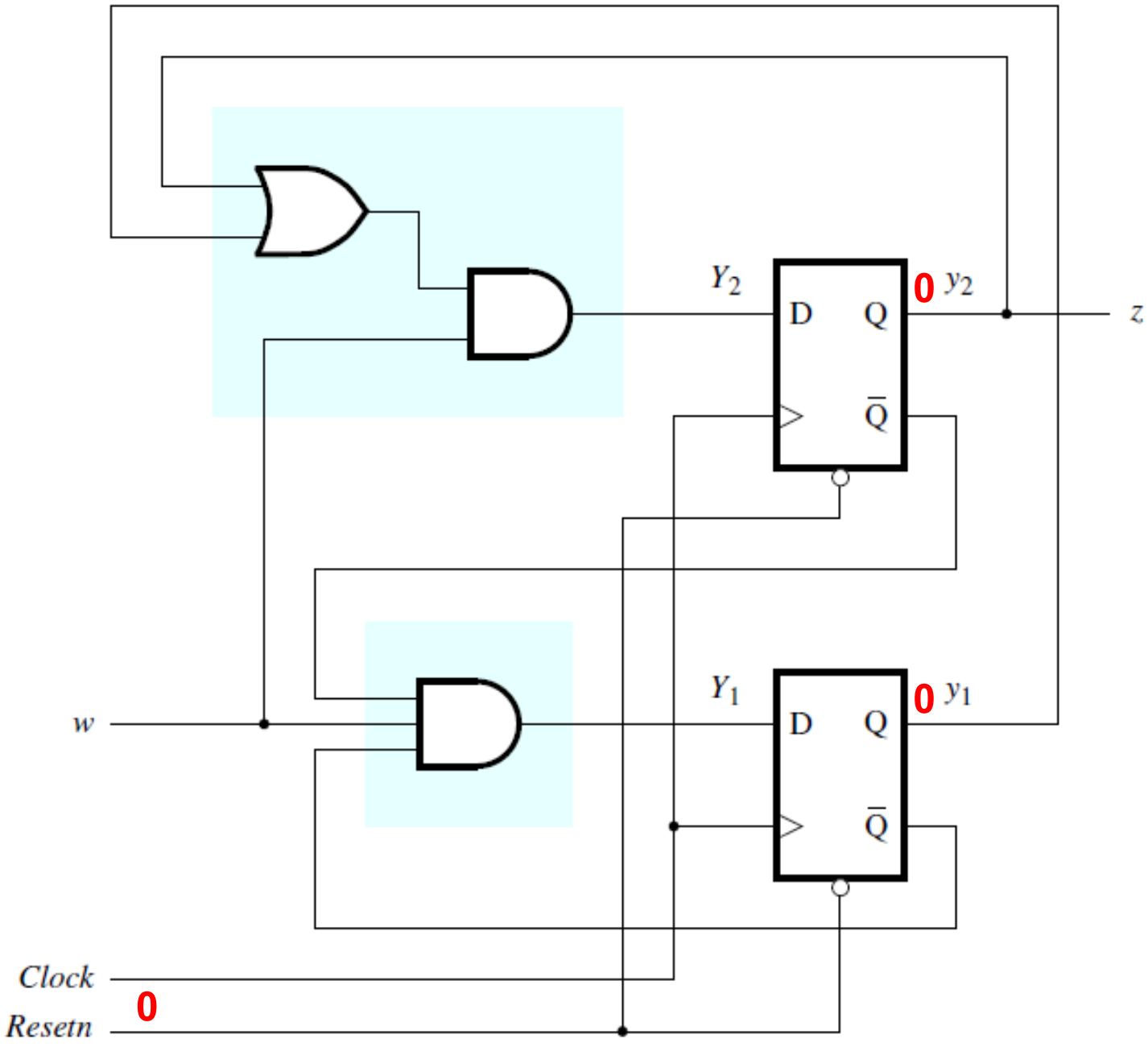


[Figure 6.8 from the textbook]



[Figure 6.8 from the textbook]

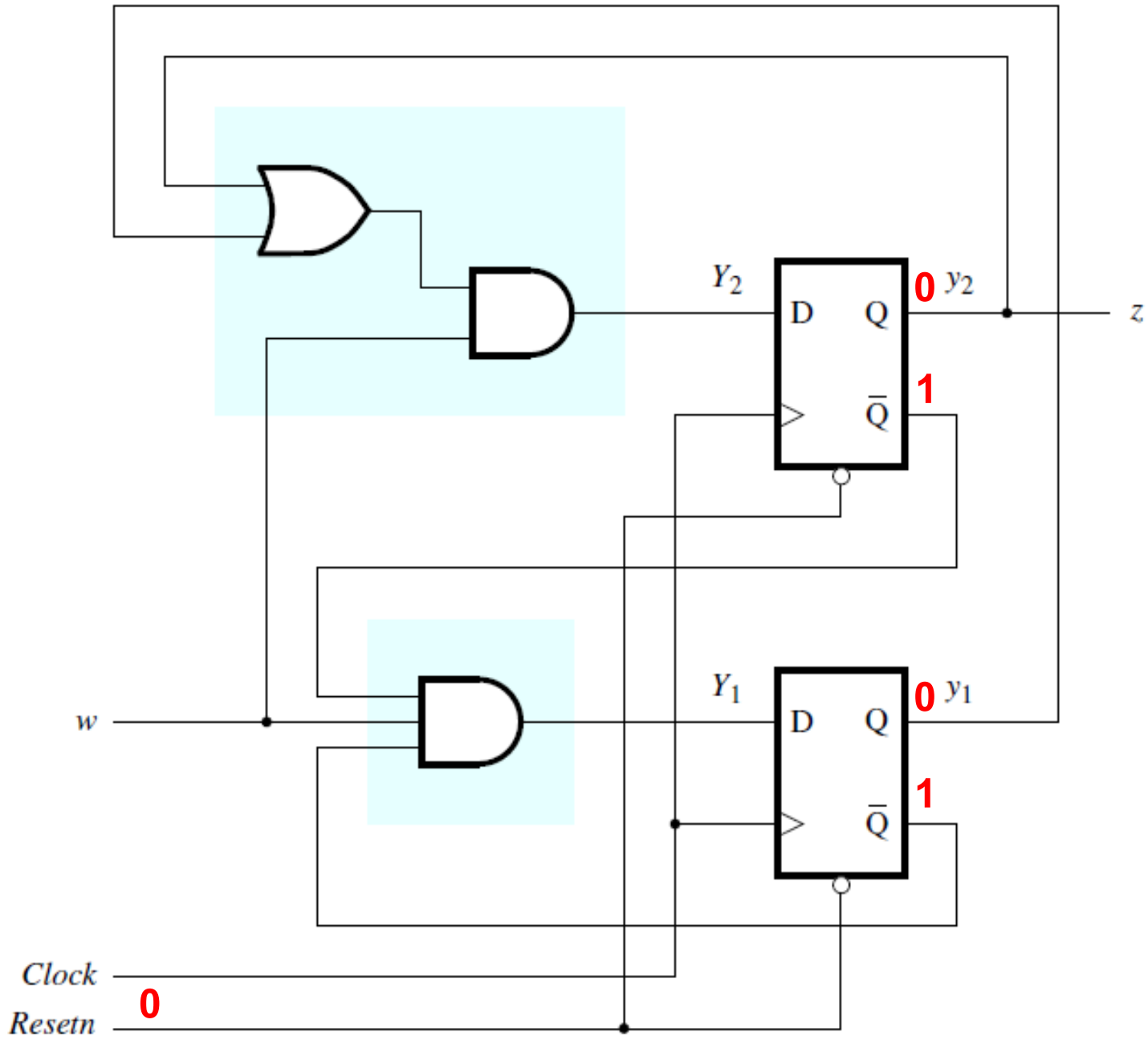
State A=00



Finally, we add a reset signal. When it is equal to zero it puts the machine back to its start state, which is state *00* in this case.

[Figure 6.8 from the textbook]

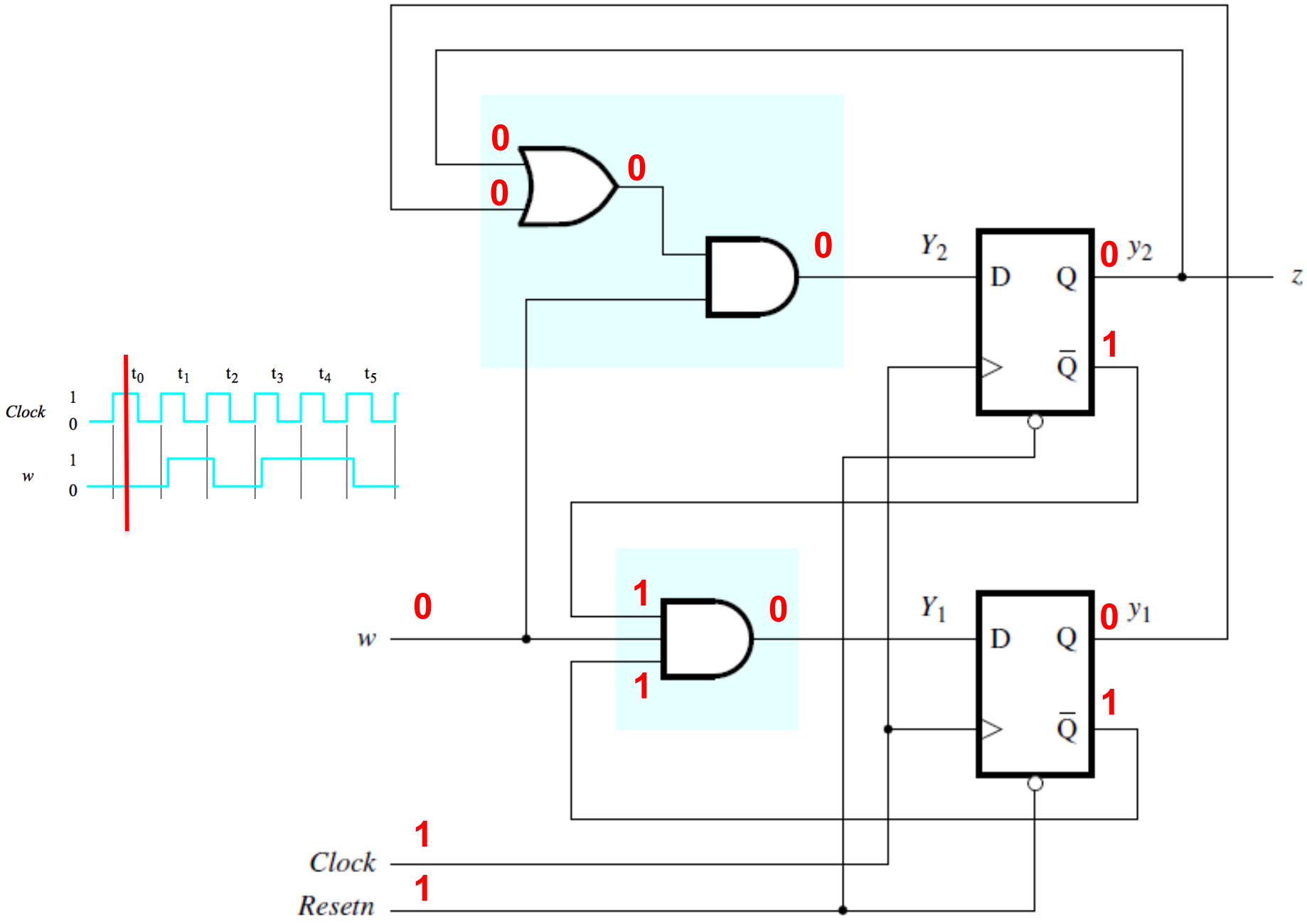
State A=00



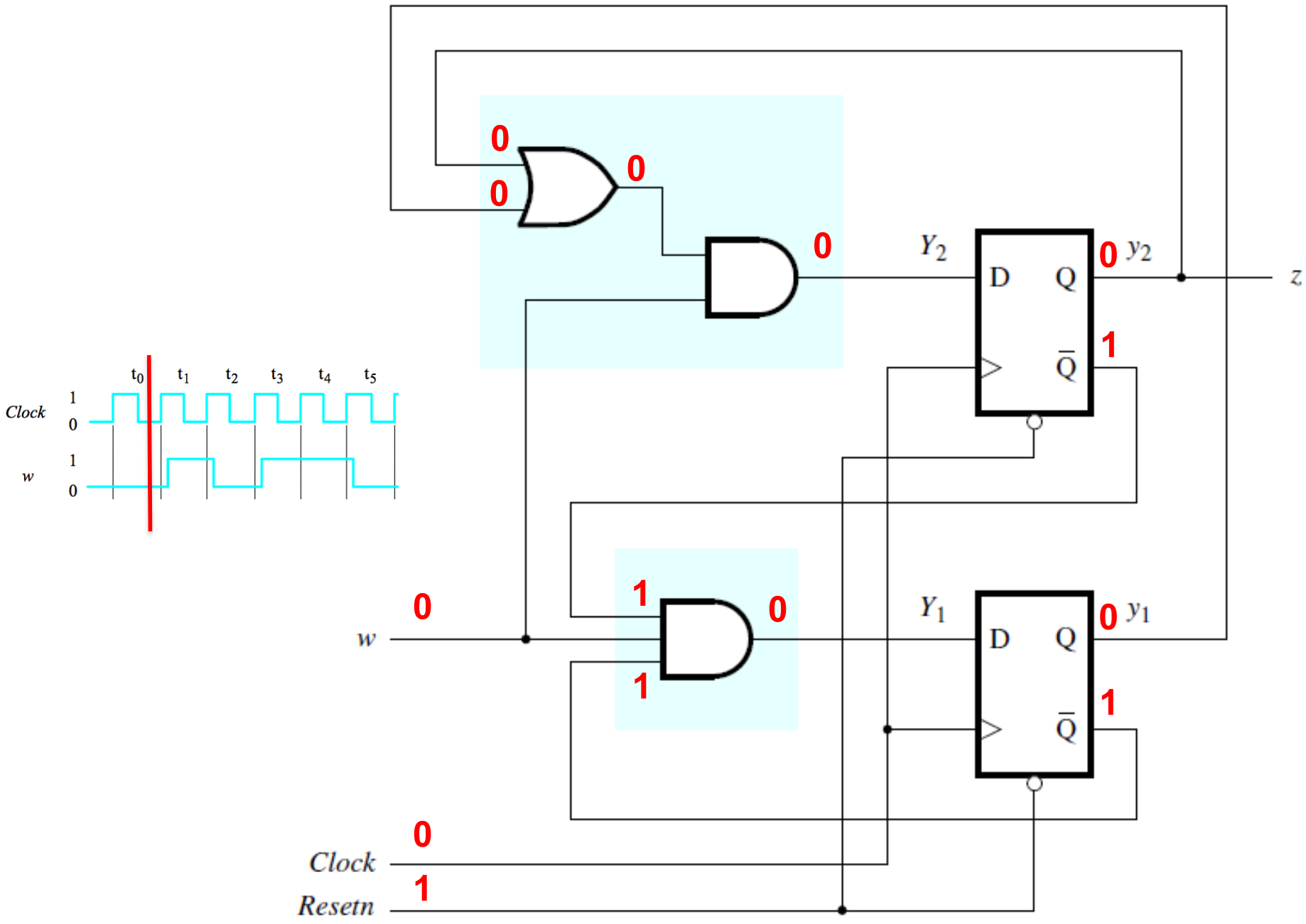
Finally, we add a reset signal. When it is equal to zero it puts the machine back to its start state, which is state *00* in this case.

[Figure 6.8 from the textbook]

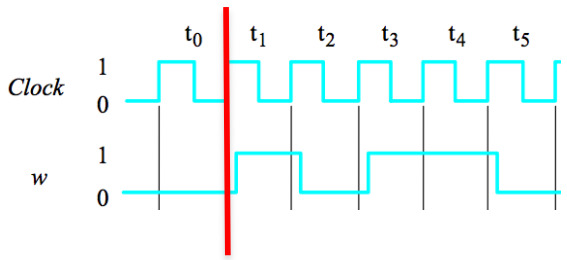
State A=00

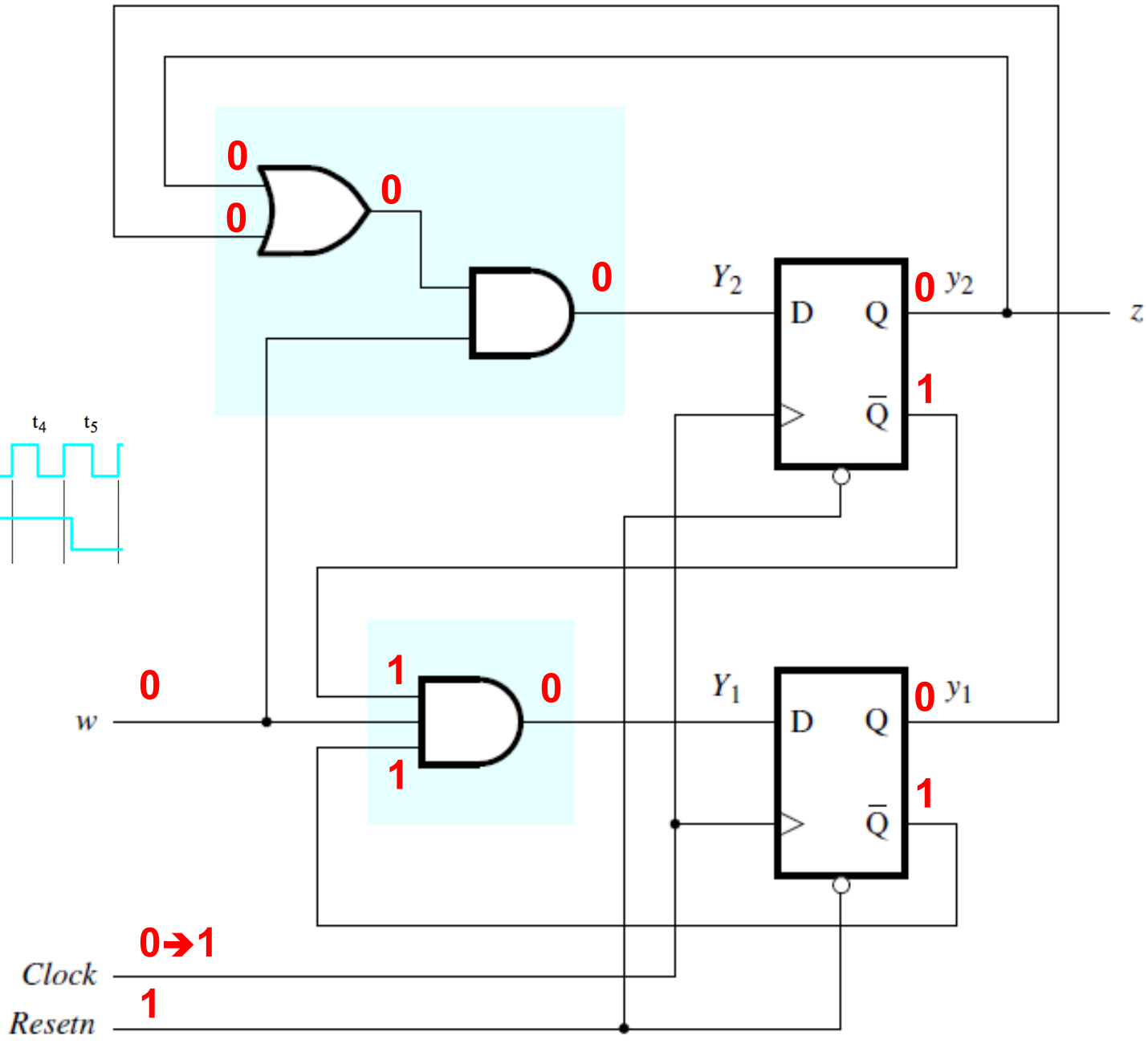

0
0
0
0
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
w
1
0
1
0
0
1
1
0
Y_1
D
Q
$\bar{Q}$
0
y_1
1
w
1
Clock
1
Resetn

State A=00

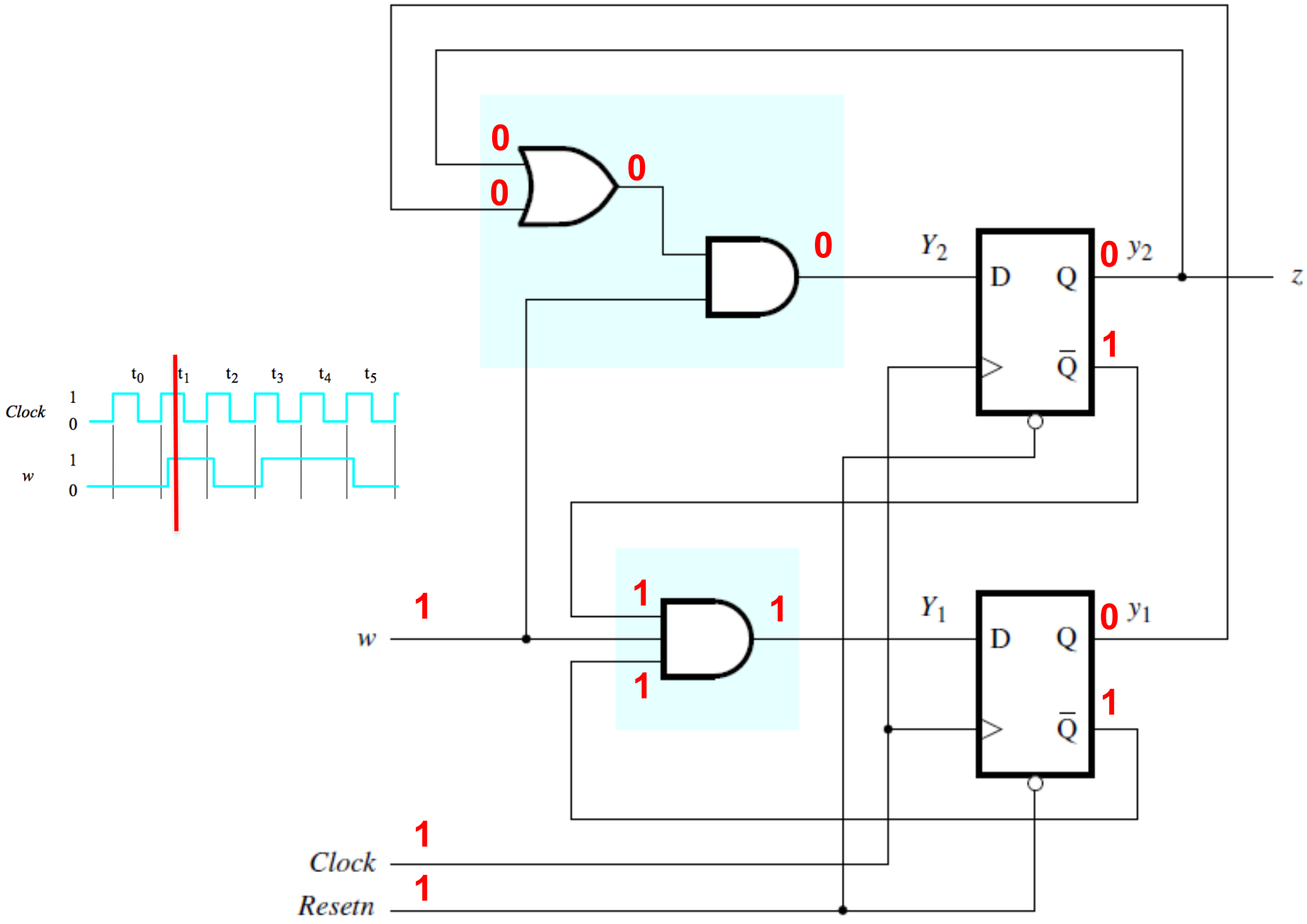

0
0
0
0
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t_0
t_1
t_2
t_3
t_4
t_5
Clock
1
0
w
1
0
1
0
0
1
w
Y_1
D
Q
$\bar{Q}$
0
y_1
1
0
Clock
1
Resetn

State A=00

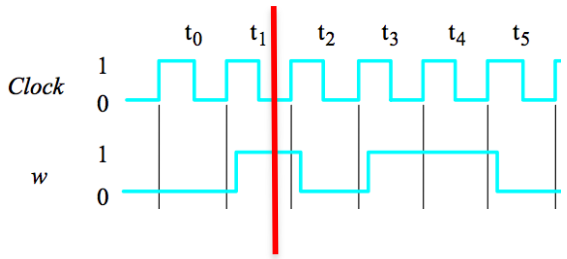

t0
t1
t2
t3
t4
t5
Clock
1
0
w
1
0

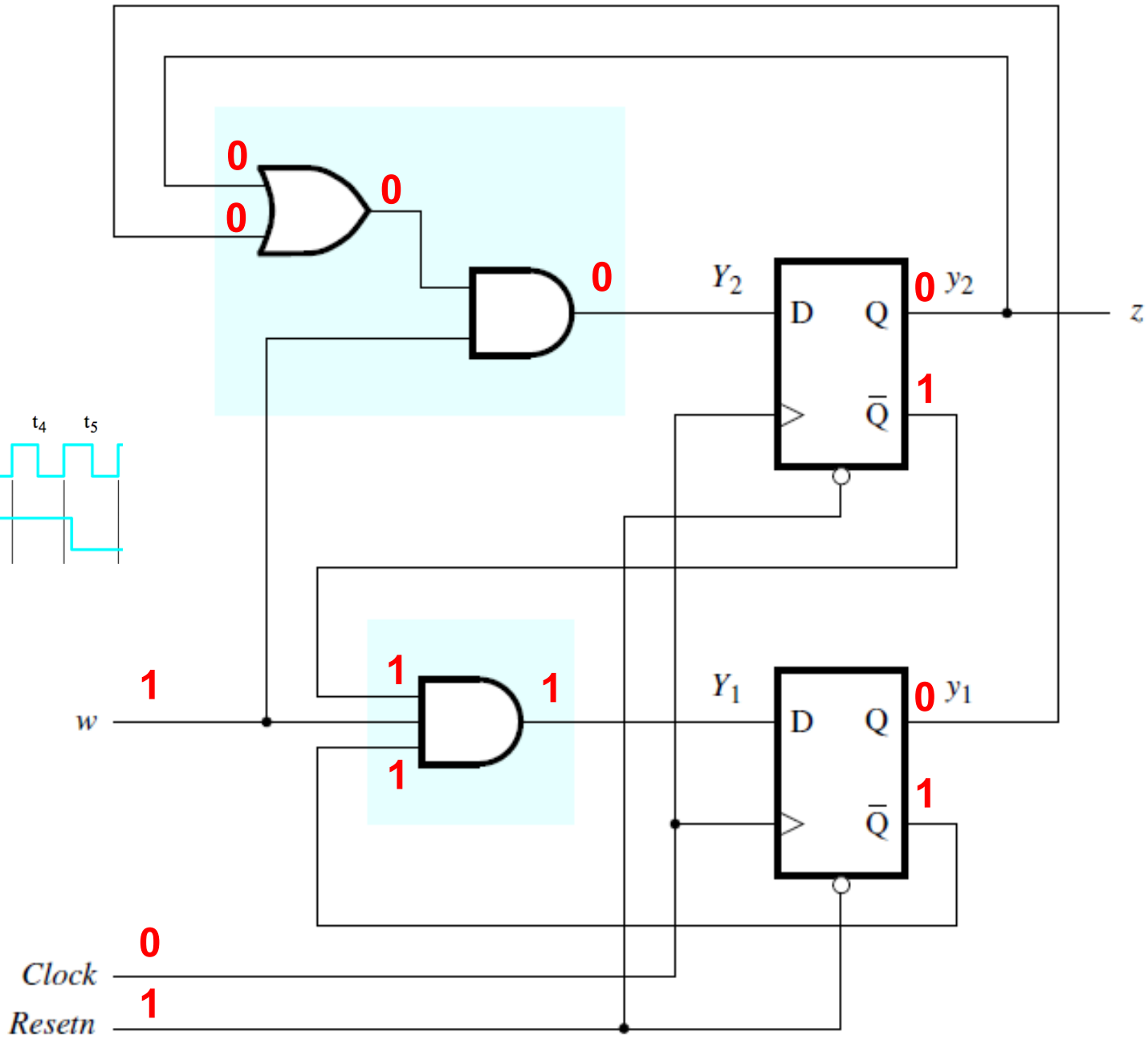

0
0
0
0
Y2
D
Q
0
y2
z
1
Q
1
0
w
1
0
1
Y1
D
Q
0
y1
1
Q
0→1
Clock
1
Resetn

State A=00

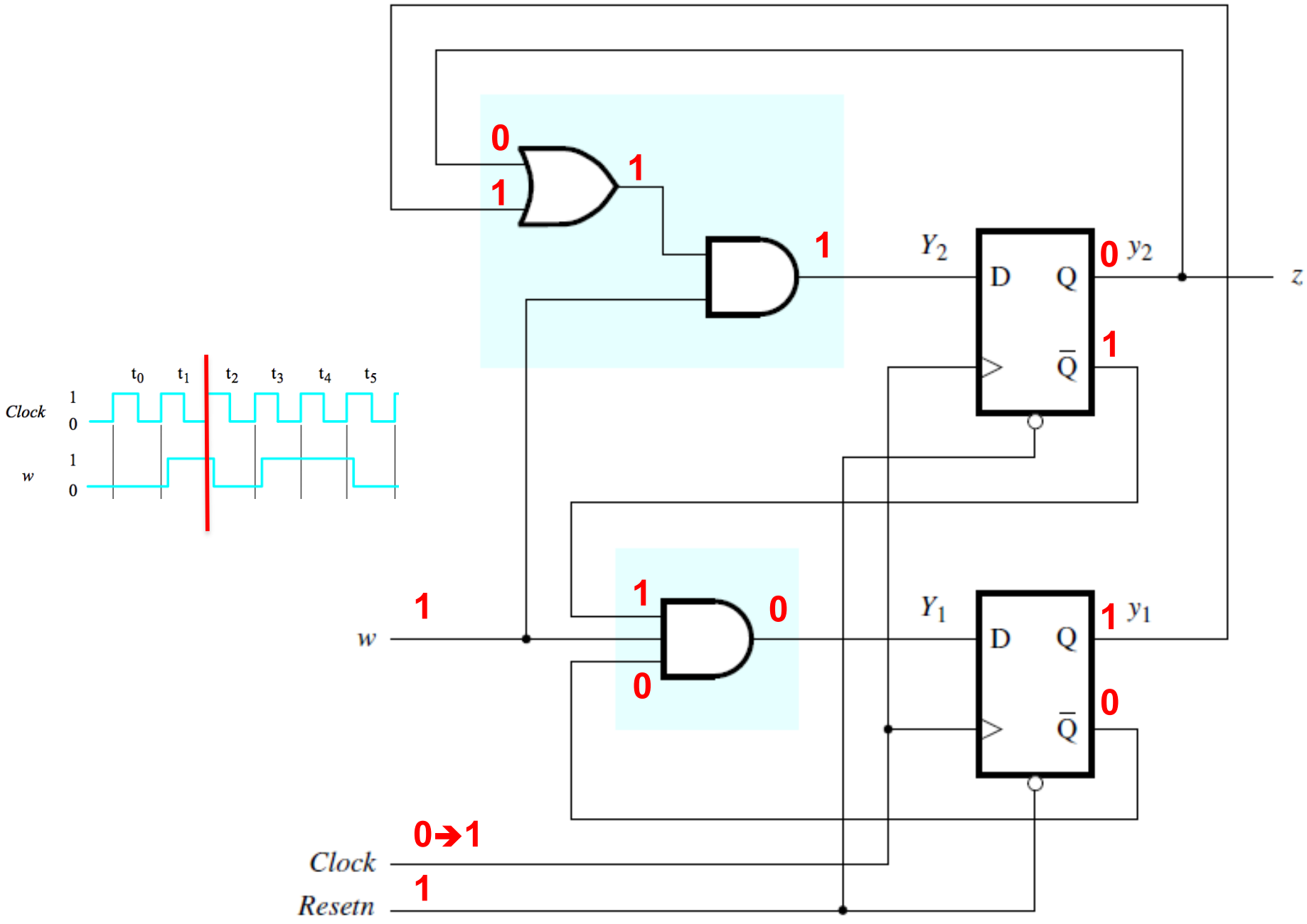

0
0
0
0
Y_2
D
Q
0
y_2
$\bar{Q}$
1
z
t0
t1
t2
t3
t4
t5
Clock
1
0
w
1
0
1
1
1
1
w
Y_1
D
Q
0
y_1
$\bar{Q}$
1
1
Clock
1
Resetn

State A=00

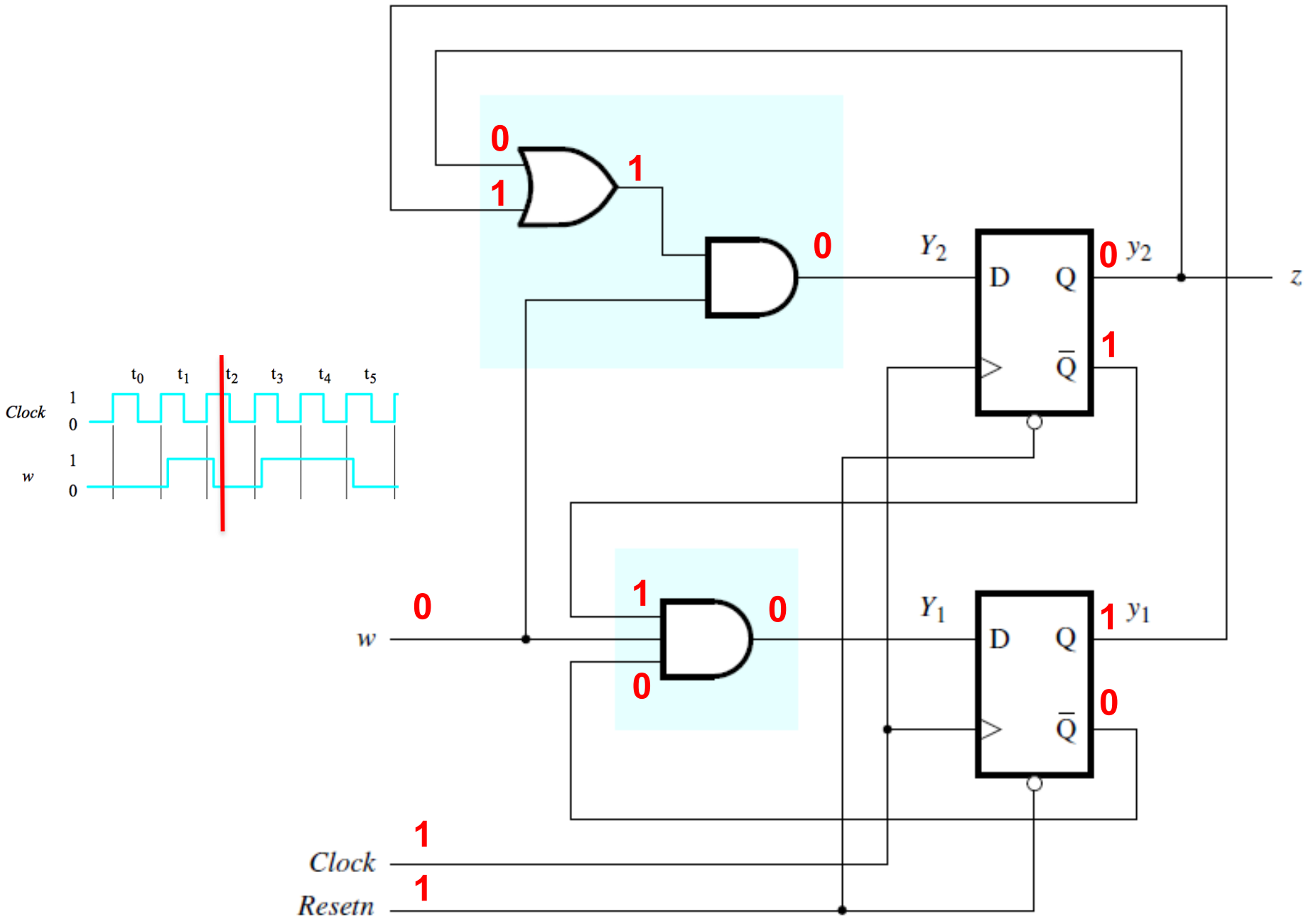

t0
t1
t2
t3
t4
t5
Clock
1
0
w
1
0


0
0
0
0
Y2
D
Q
0
y2
1
z
1
1
1
1
w
1
Y1
D
Q
0
y1
1
0
Clock
1
Resetn

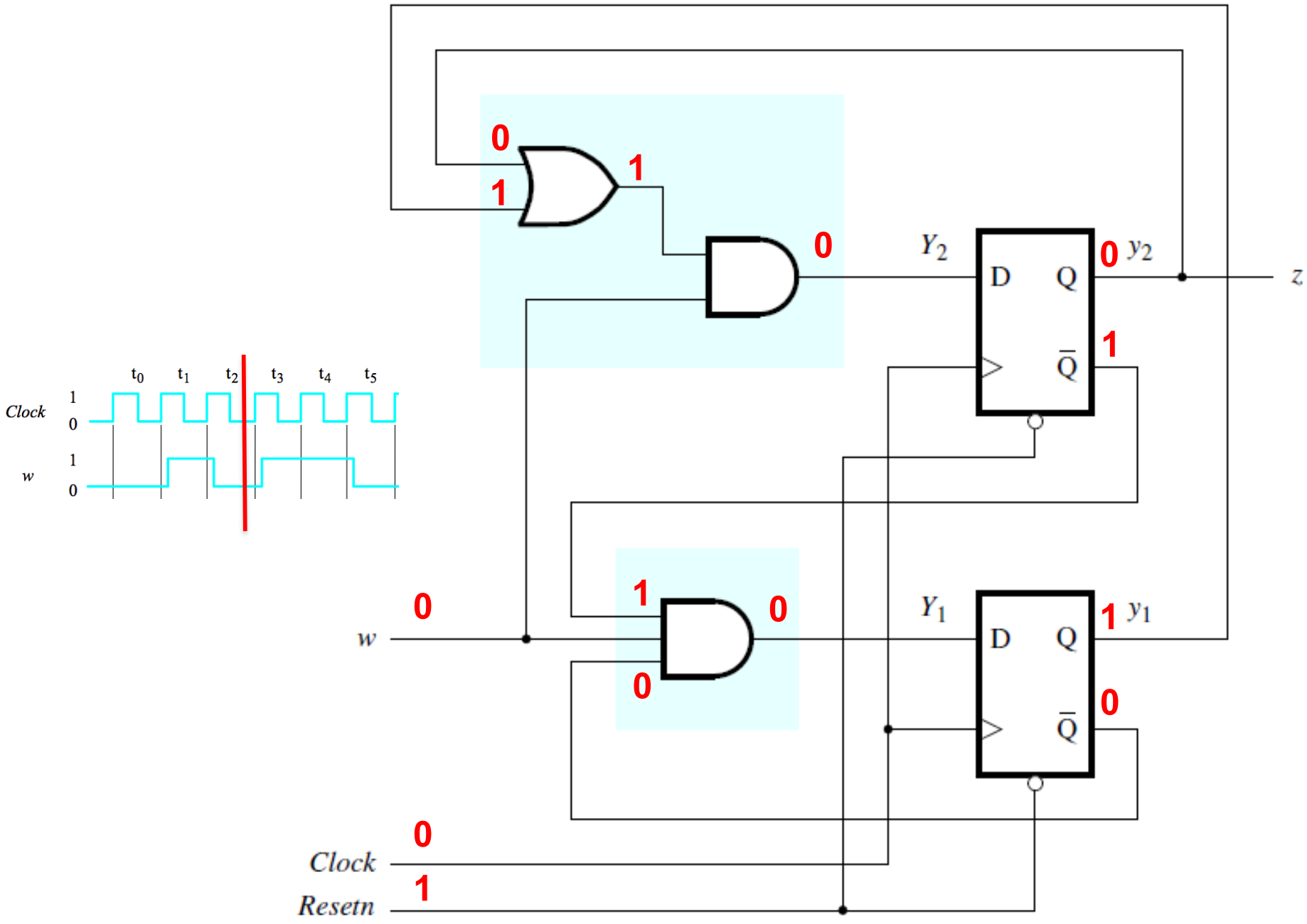
State B=01


0
1
1
1
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
1
0
w
1
0
1
0
1
0
w
Y_1
D
Q
$\bar{Q}$
1
y_1
0
0→1
Clock
1
Resetn

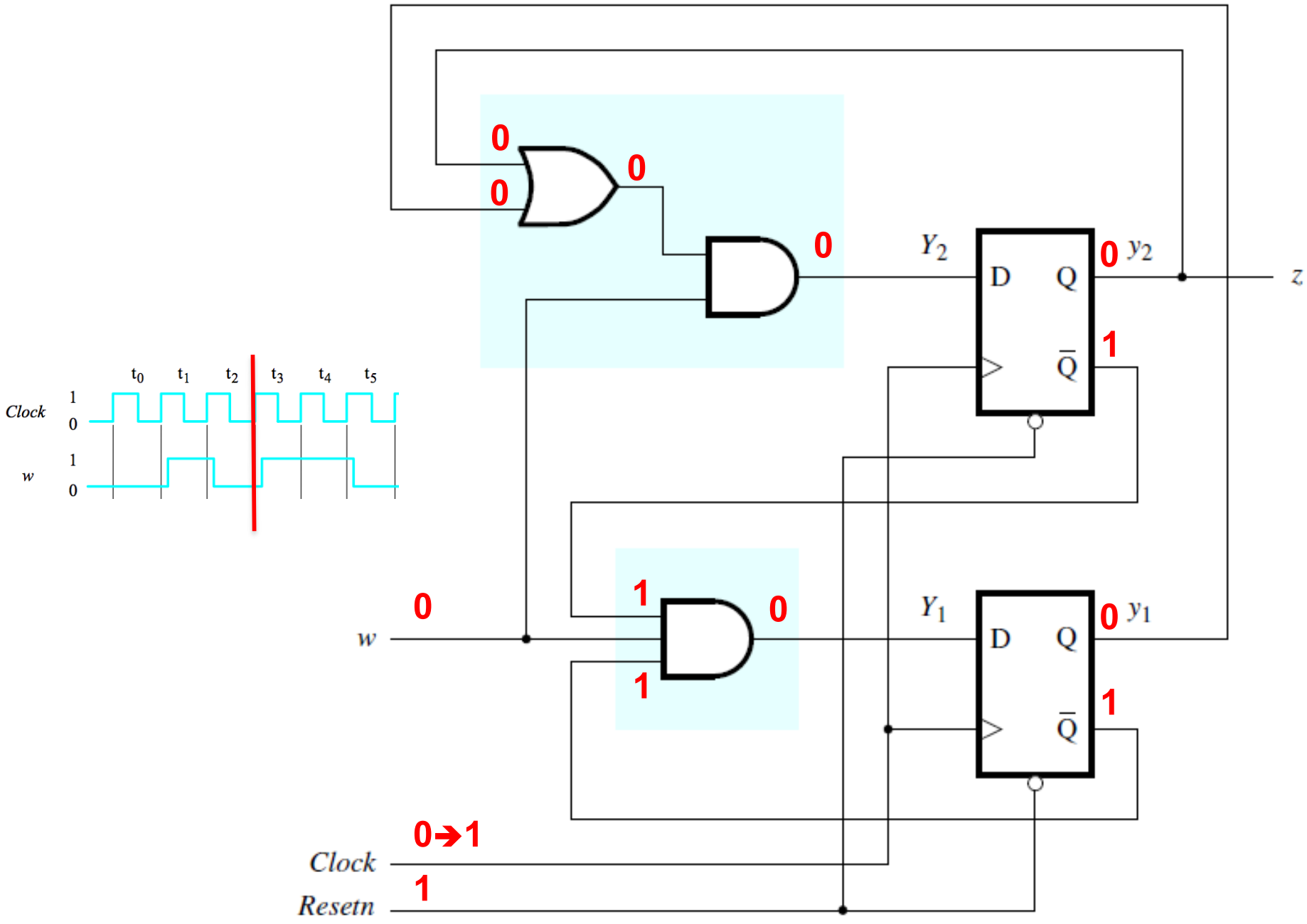
State B=01


0
1
1
0
Y_2
D
Q
0
y_2
z
$\bar{Q}$
1
t_0 t_1 t_2 t_3 t_4 t_5
Clock
1
0
w
1
0
1
0
0
w
0
Y_1
D
Q
1
y_1
$\bar{Q}$
0
1
Clock
1
Resetn

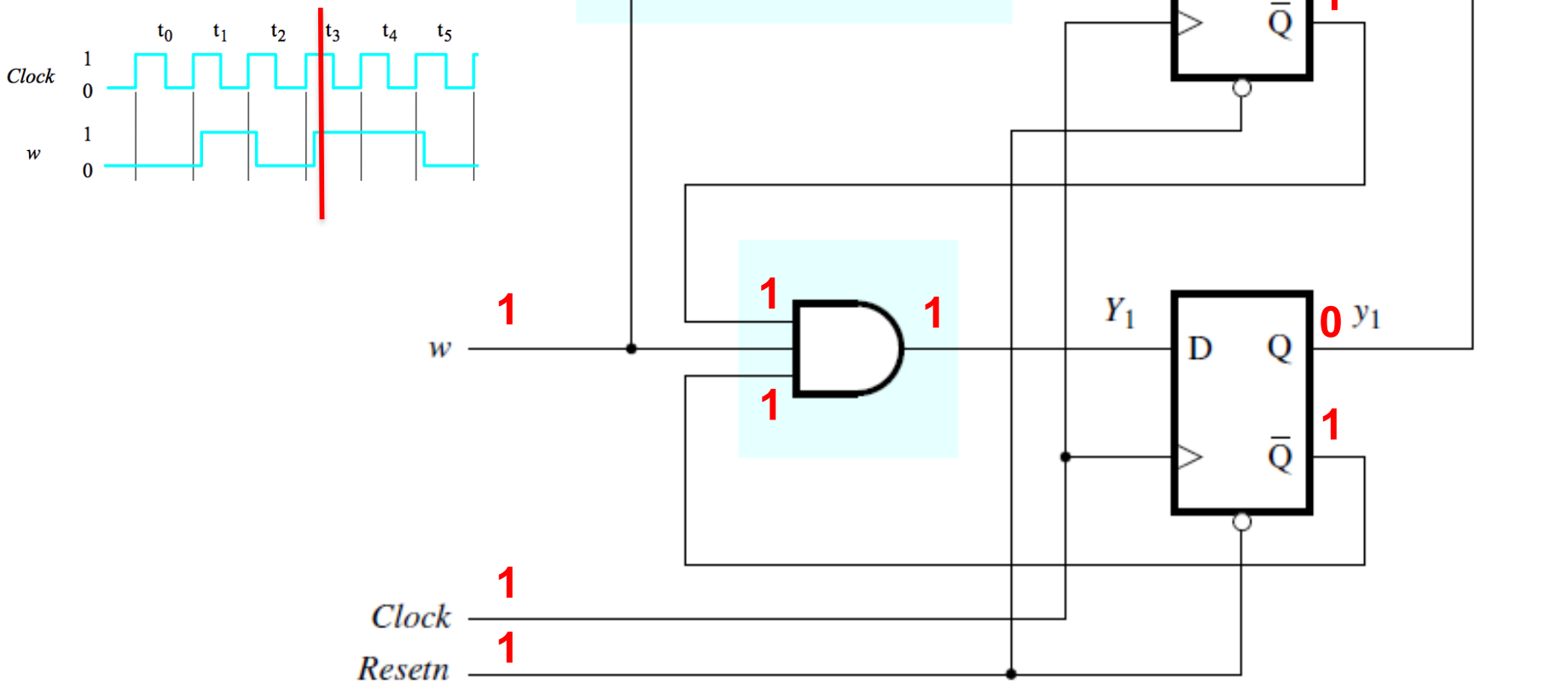
State B=01


0
1
1
0
Y_2
D
Q
0
y_2
z
$\bar{Q}$
1
t_0 t_1 t_2 t_3 t_4 t_5
Clock
1
0
w
1
0
1
0
0
w
Y_1
D
Q
1
y_1
0
$\bar{Q}$
0
Clock
1
Resetn

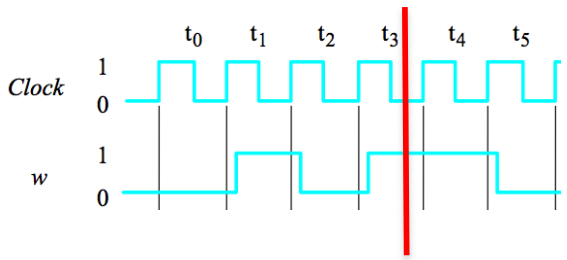
State A=00

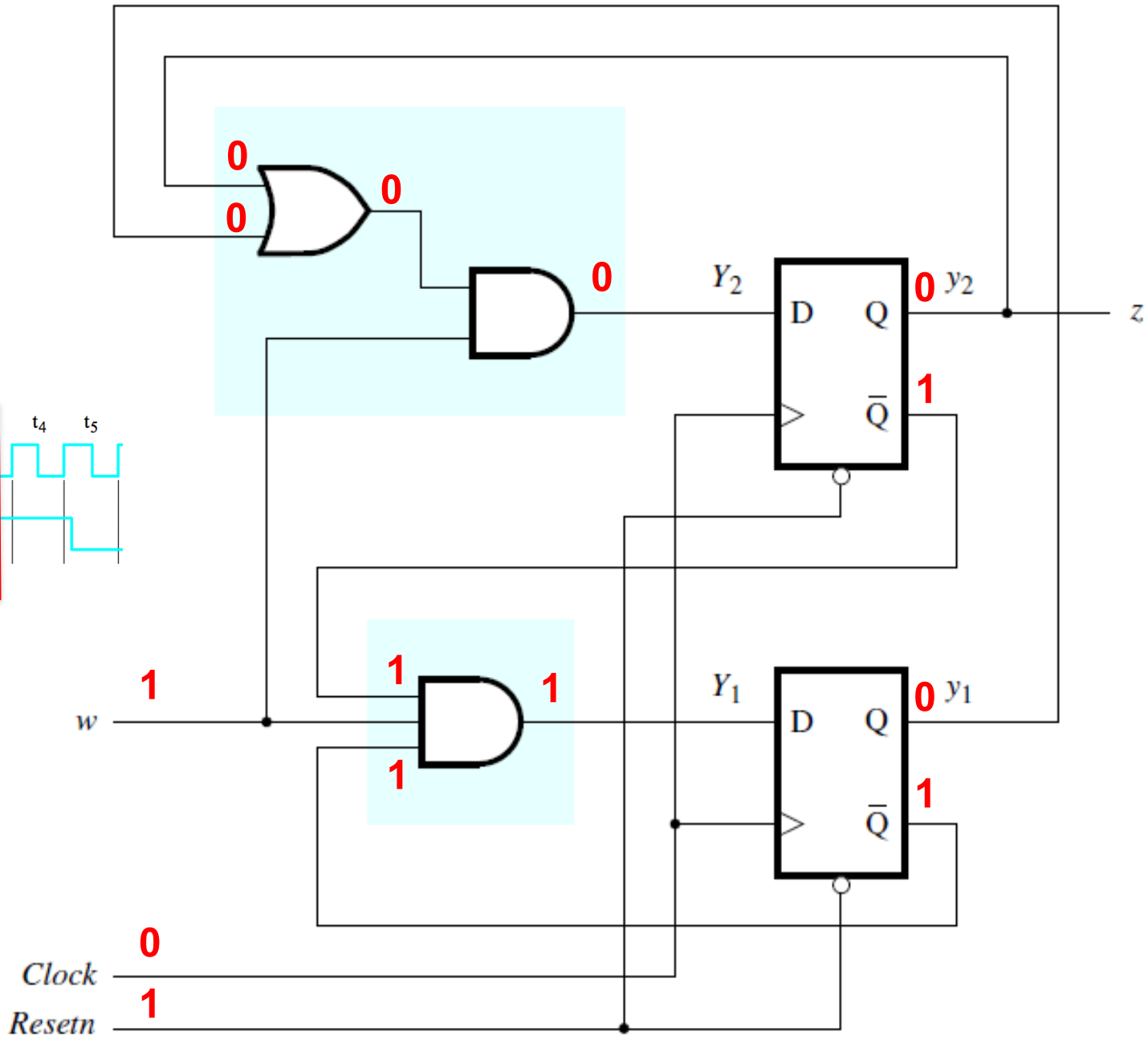

0
0
0
0
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
w
1
0
1
0
0
1
1
0
w
Y_1
D
Q
$\bar{Q}$
0
y_1
1
0→1
Clock
1
Resetn

State A=00

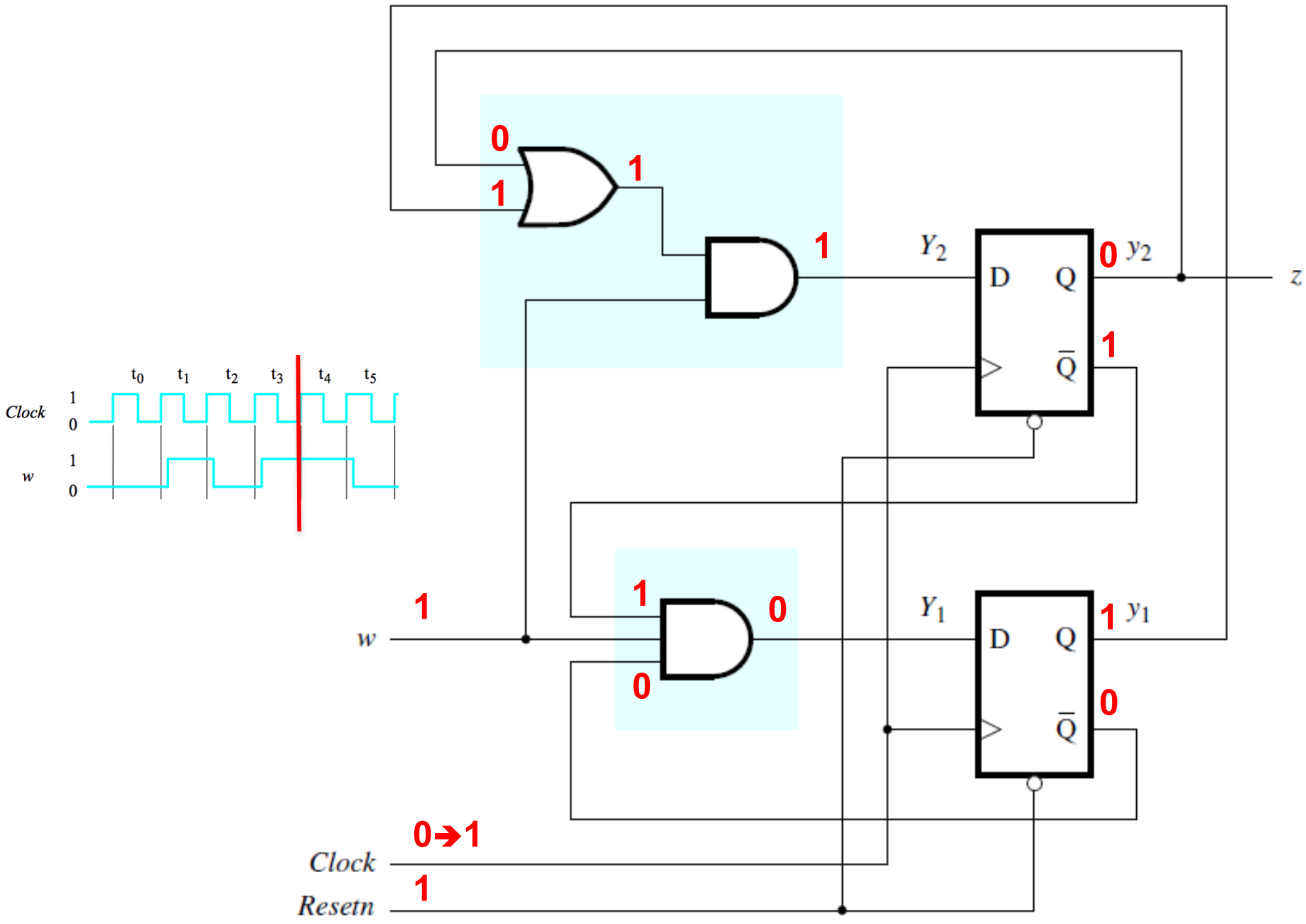

t0 t1 t2 t3 t4 t5
Clock
w
1
0
0
0
0
0
Y2
D
Q
Q
0
y2
1
z
1
1
1
1
w
Y1
D
Q
Q
0
y1
1
1
Clock
Resetn

State A=00

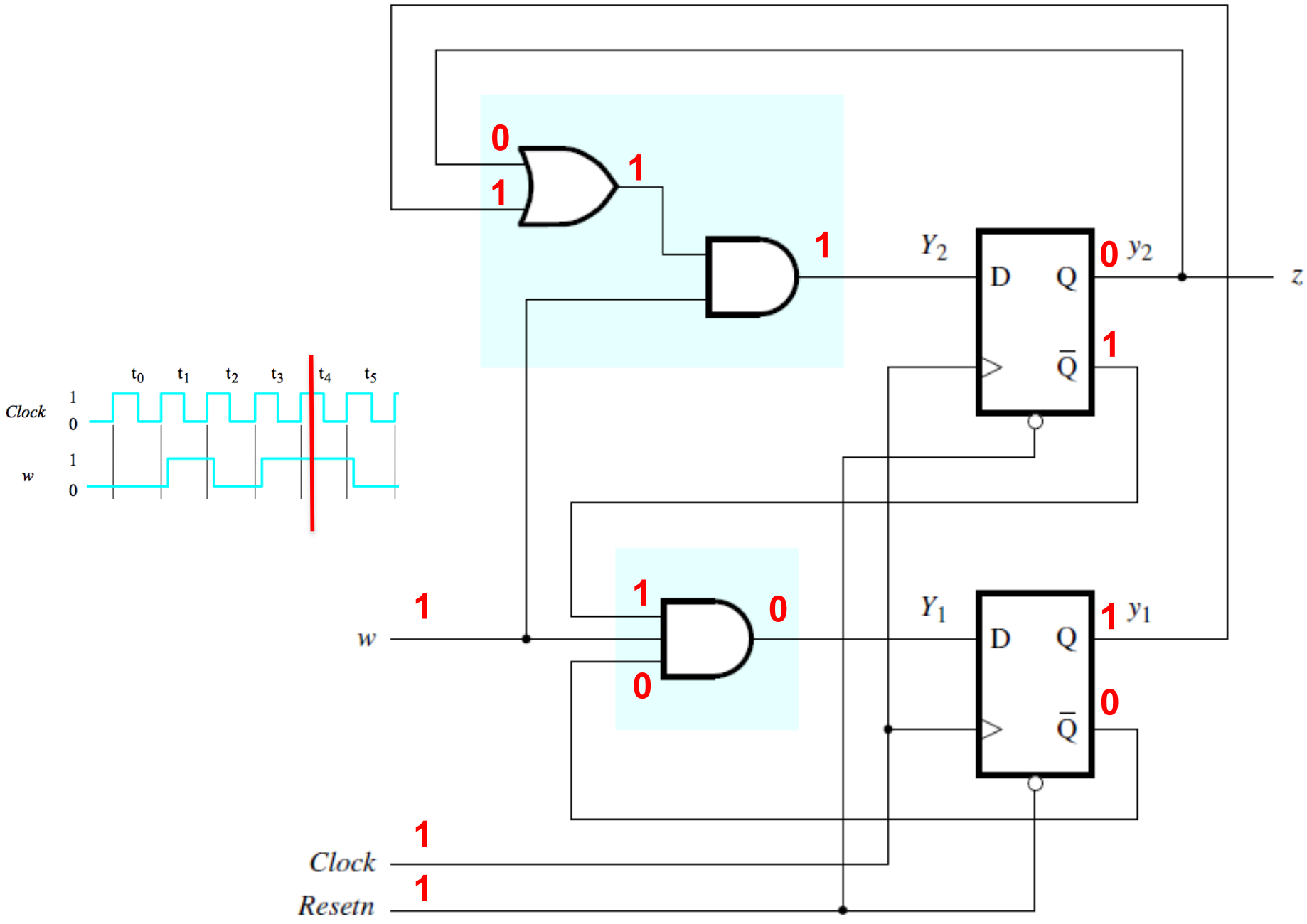

t0 t1 t2 t3 t4 t5
Clock
1
0
w
1
0


0
0
0
0
Y2
D
Q
0
y2
1
z
1
1
1
1
w
Y1
D
Q
0
y1
1
0
Clock
1
Resetn

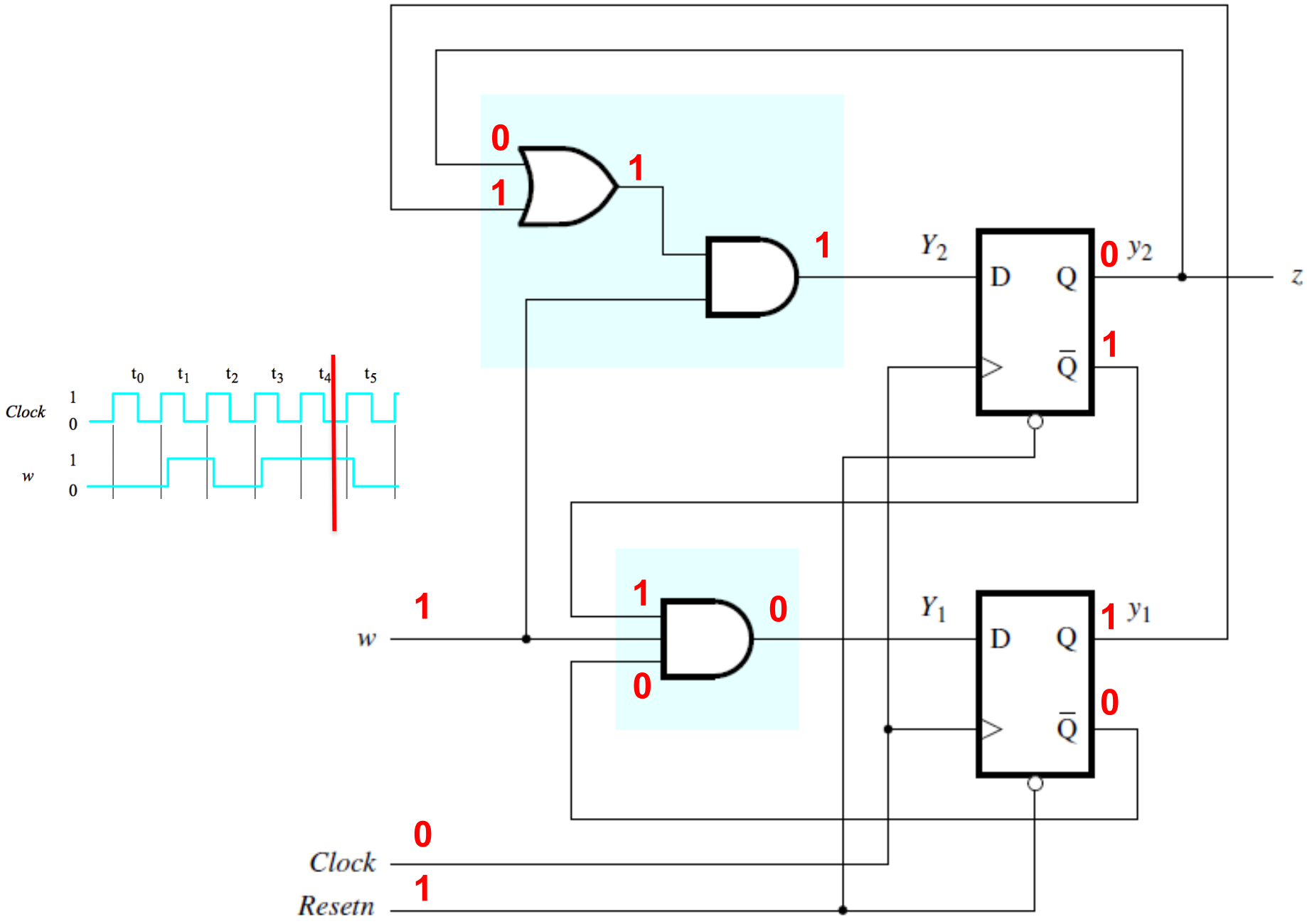
State B=01


0
1
1
1
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
w
1
0
1
0
1
0
1
w
Y_1
D
Q
$\bar{Q}$
1
y_1
0
0→1
Clock
1
Resetn

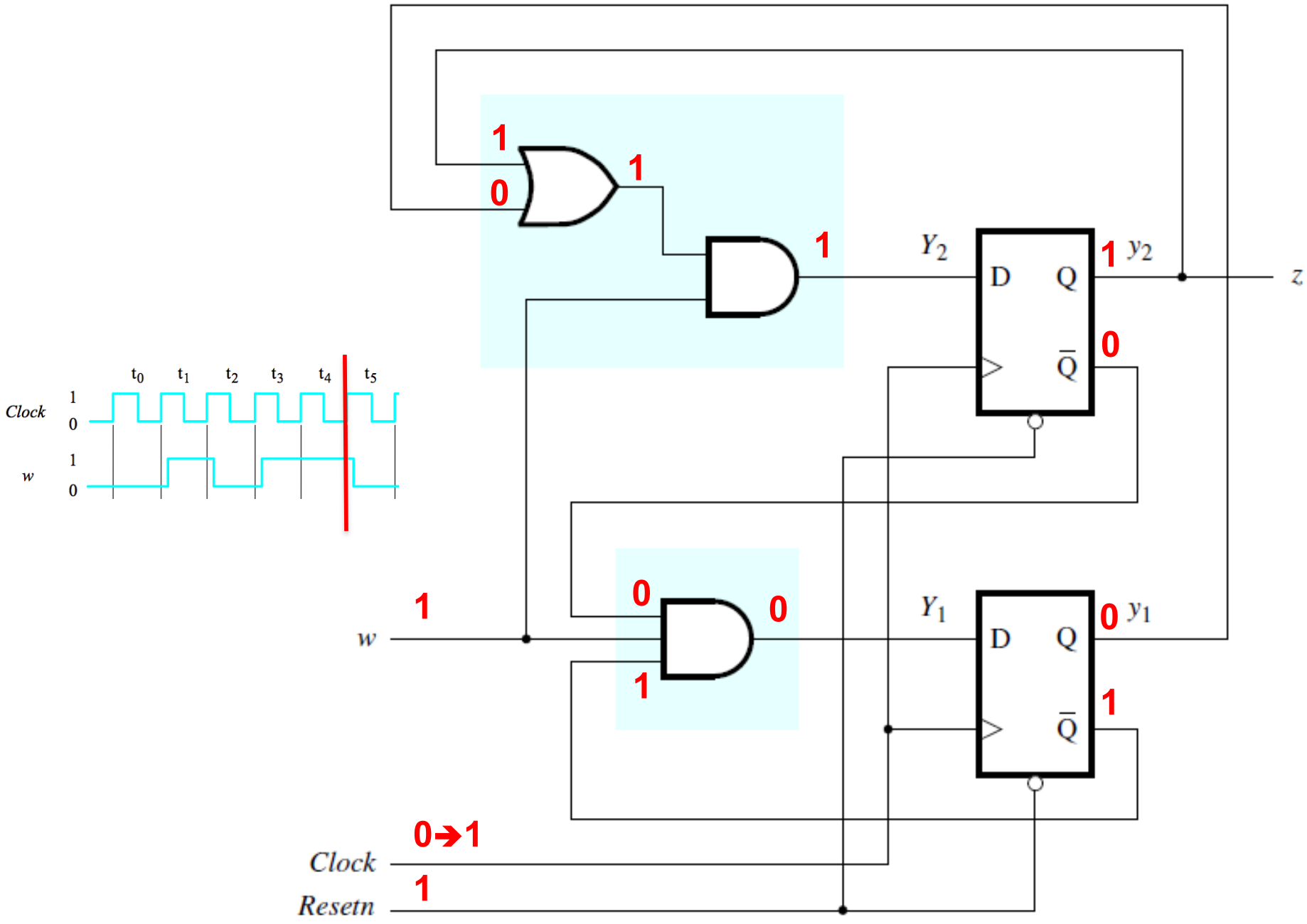
State B=01


0
1
1
1
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t0
t1
t2
t3
t4
t5
Clock
1
0
w
1
0
1
1
0
0
w
Y_1
D
Q
$\bar{Q}$
1
y_1
0
1
Clock
1
Resetn

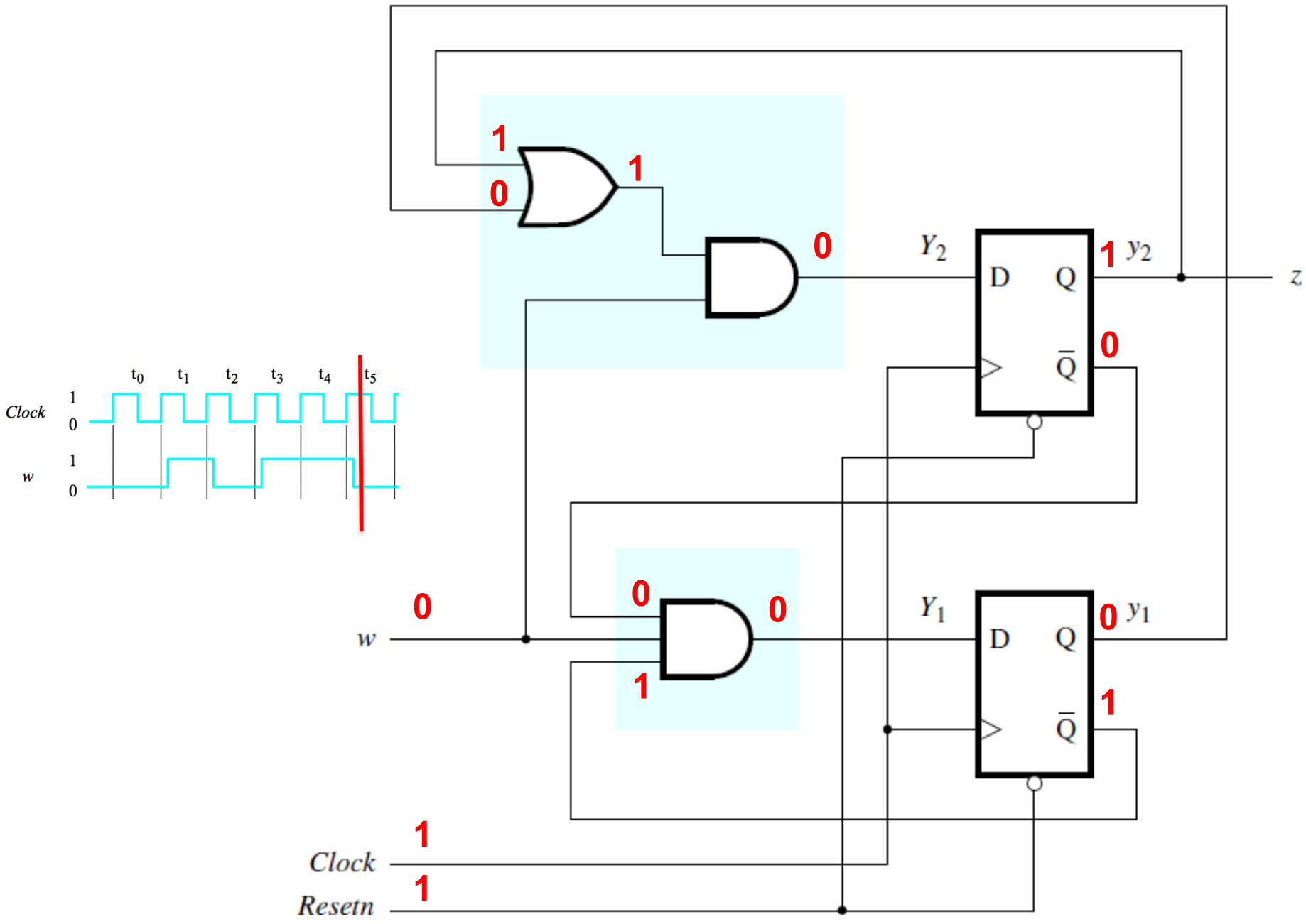
State B=01


0
1
1
1
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
w
1
0
1
0
1
0
1
w
Y_1
D
Q
$\bar{Q}$
1
y_1
0
0
Clock
Resetn

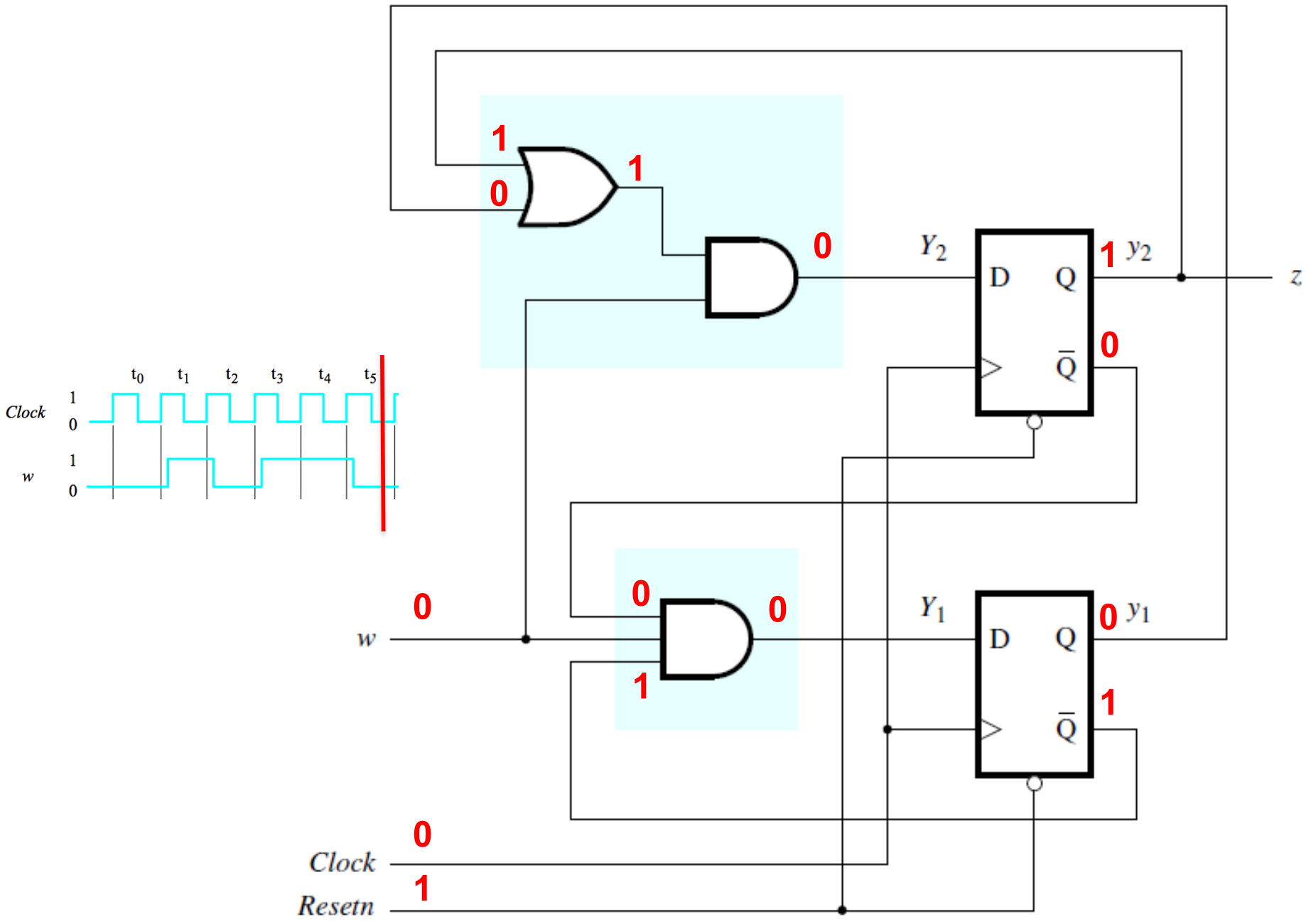
State C=10


1
0
1
1
Y_2
D
Q
$\bar{Q}$
1
y_2
0
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
w
1
0
1
0
0
1
w
Y_1
D
Q
$\bar{Q}$
0
y_1
1
0→1
Clock
1
Resetn

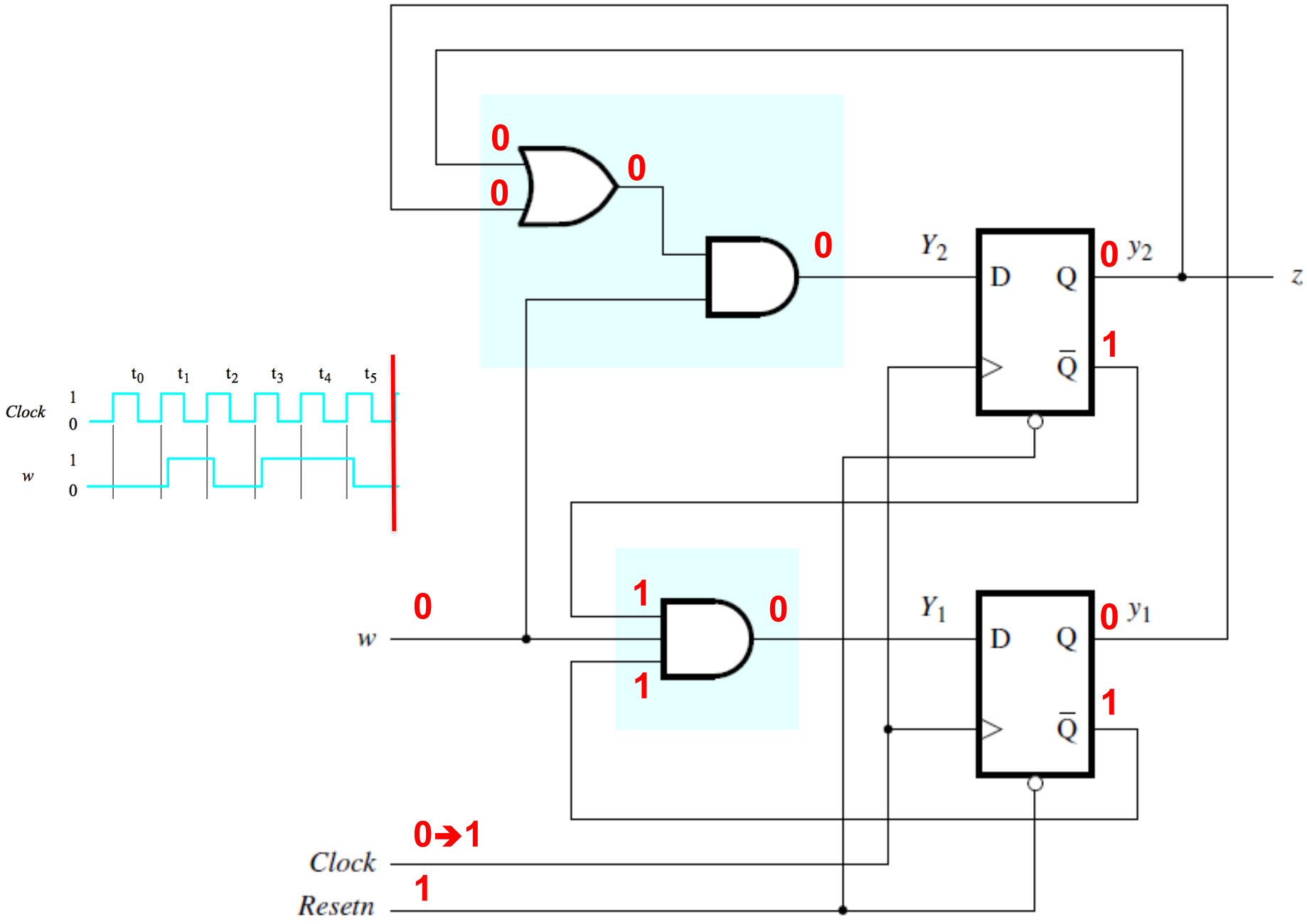
State C=10


1
0
1
0
Y_2
D
Q
$\bar{Q}$
1
y_2
0
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
w
0
0
0
1
Y_1
D
Q
$\bar{Q}$
0
y_1
1
w
1
Clock
1
Resetn

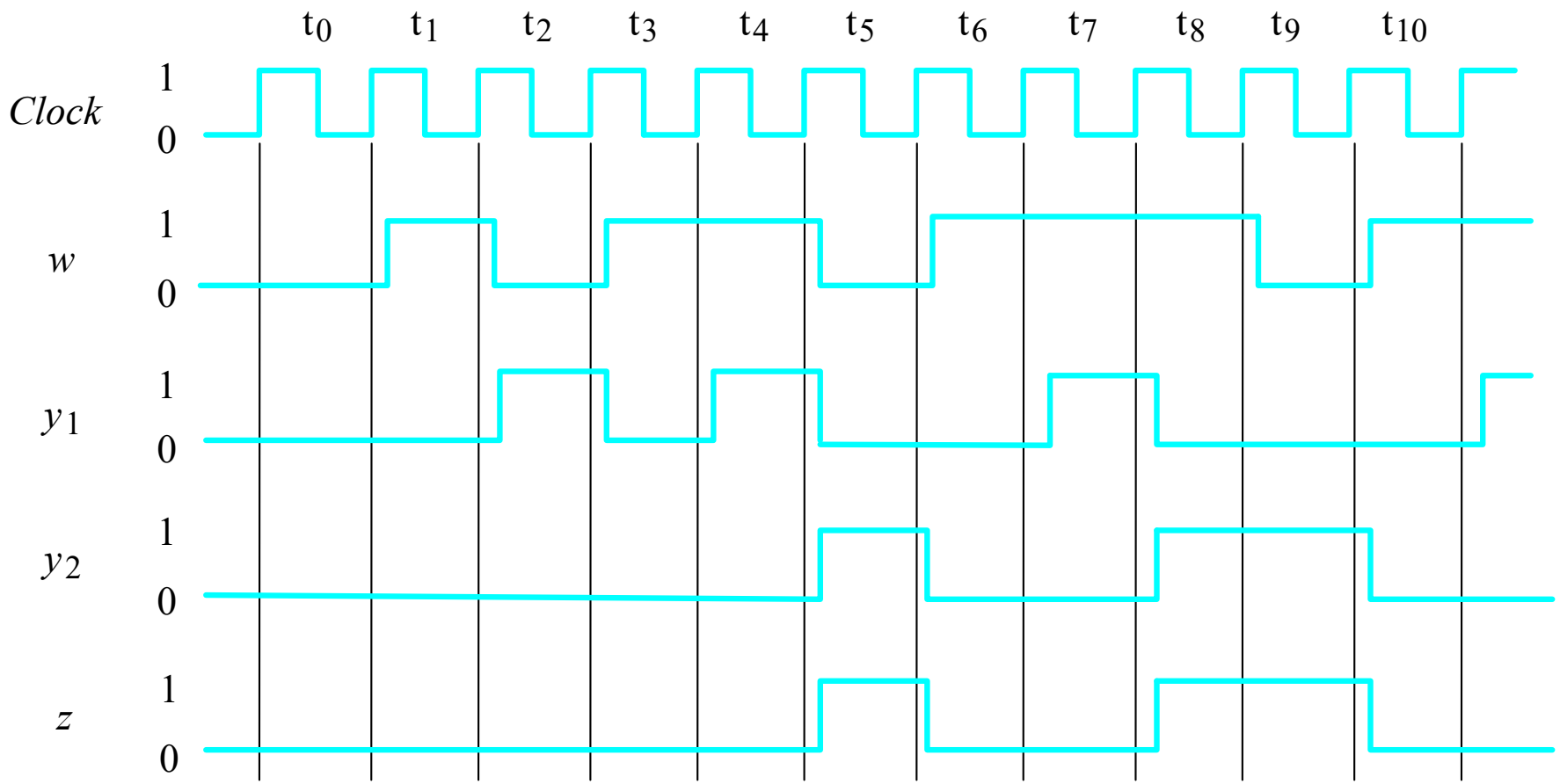
State C=10


1
0
1
0
Y_2
D
Q
$\bar{Q}$
1
y_2
0
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
w
0
0
0
1
0
Y_1
D
Q
$\bar{Q}$
0
y_1
1
w
0
Clock
1
Resetn

State A=00

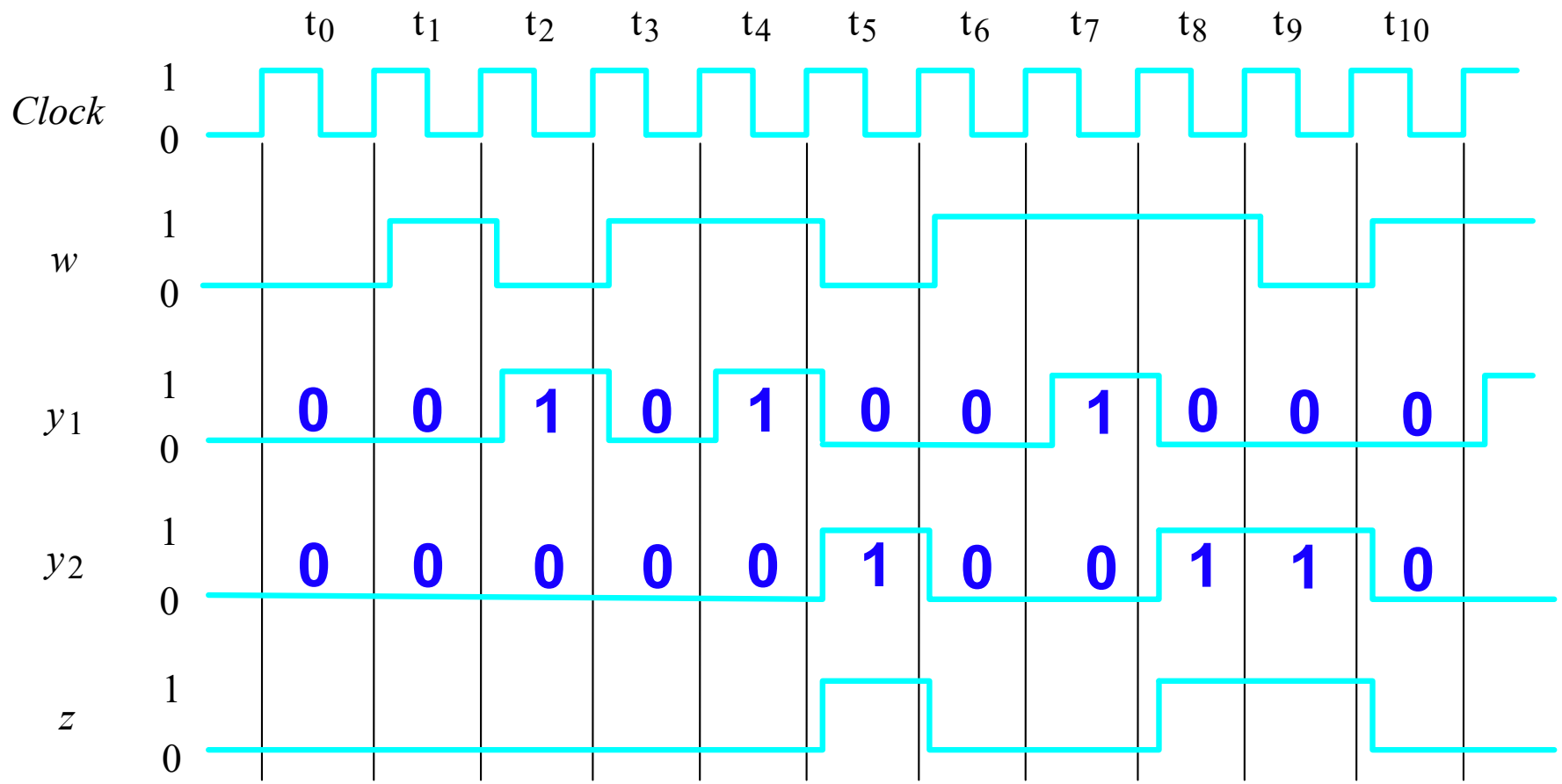

0
0
0
0
Y_2
D
Q
$\bar{Q}$
0
y_2
1
z
t_0 t_1 t_2 t_3 t_4 t_5
Clock
1
0
w
1
0
0
1
0
1
Y_1
D
Q
$\bar{Q}$
0
y_1
1
w
0→1
Clock
1
Resetn

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

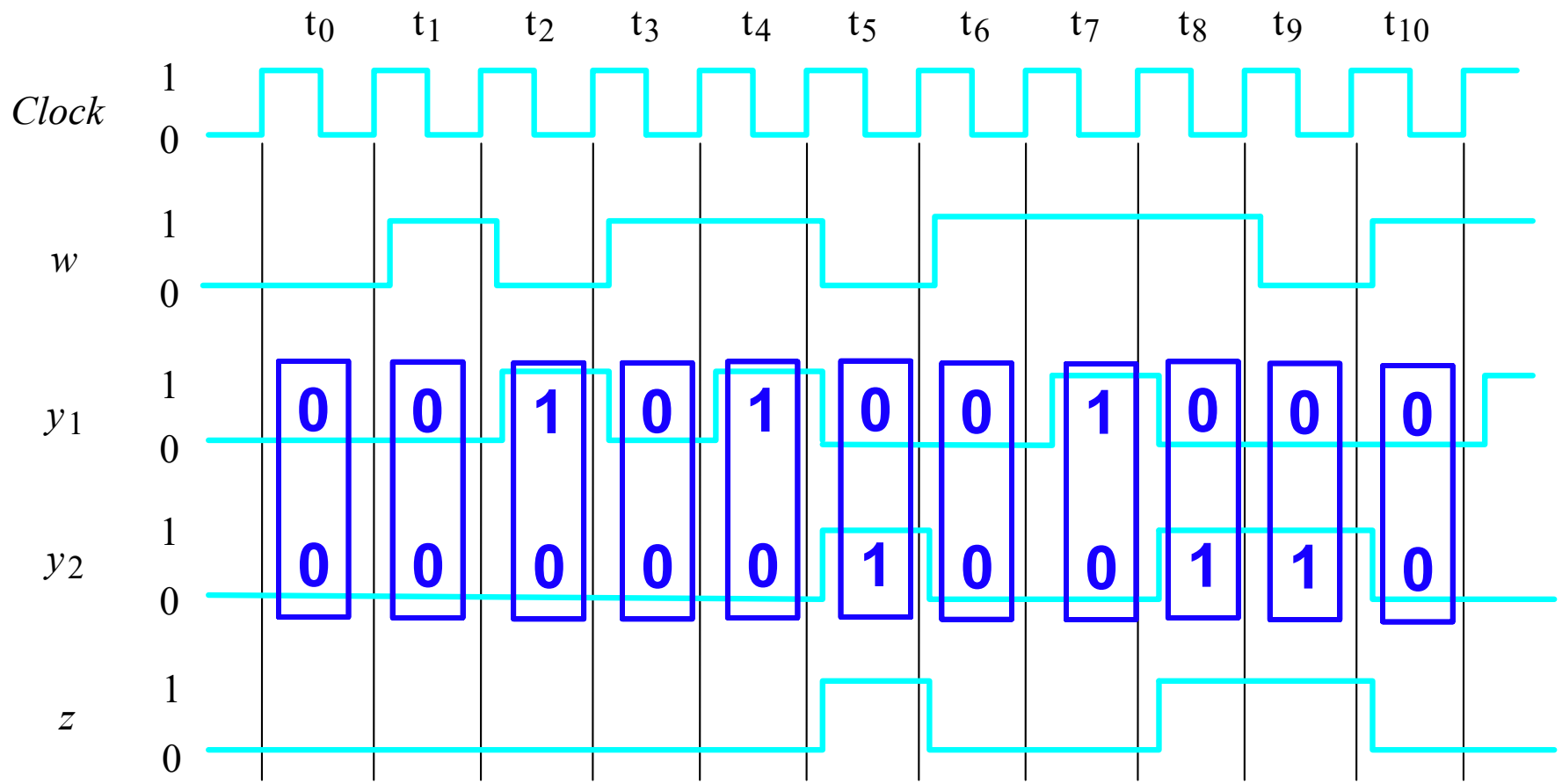


[Figure 6.9 from the textbook]

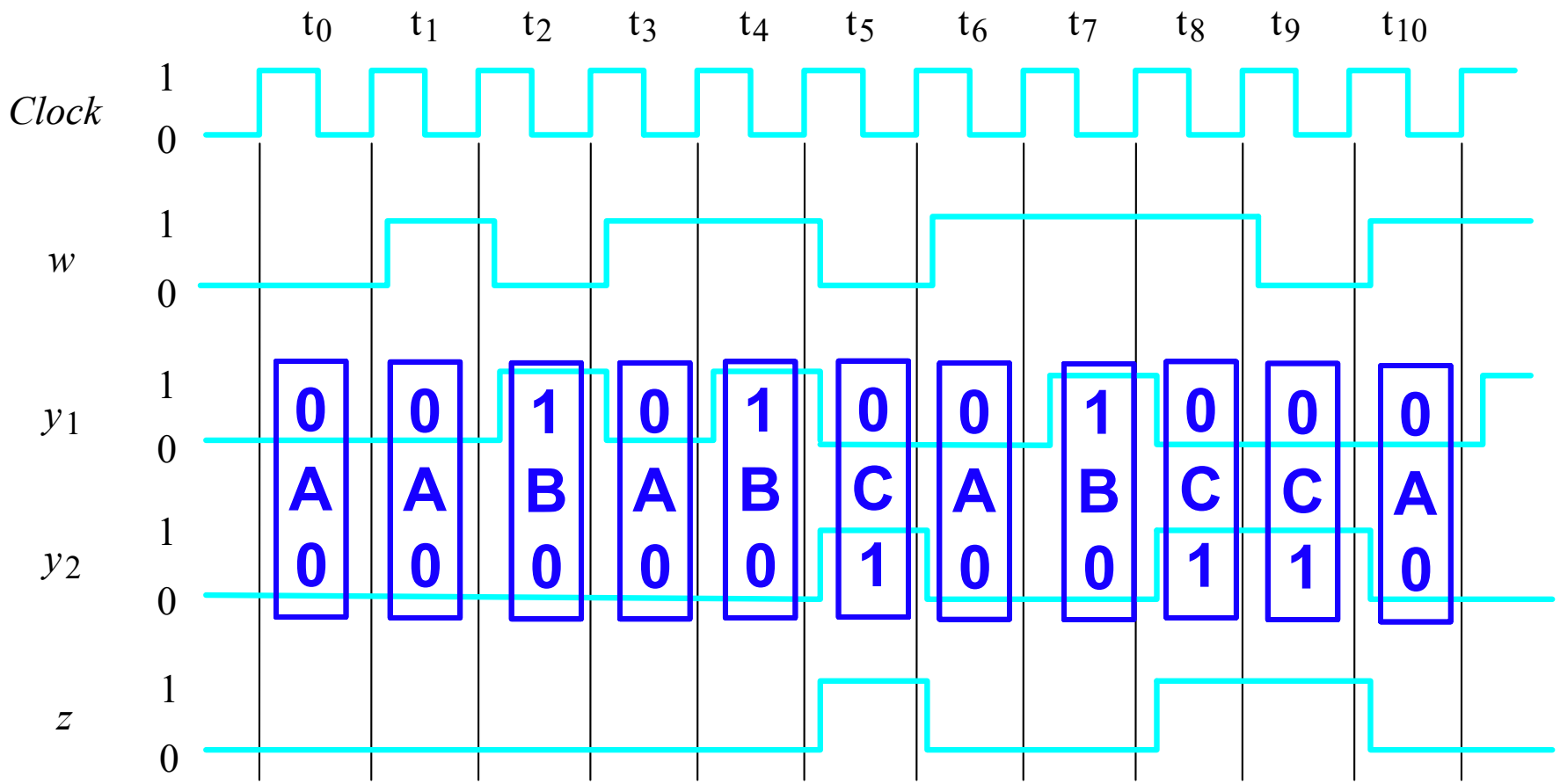
Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



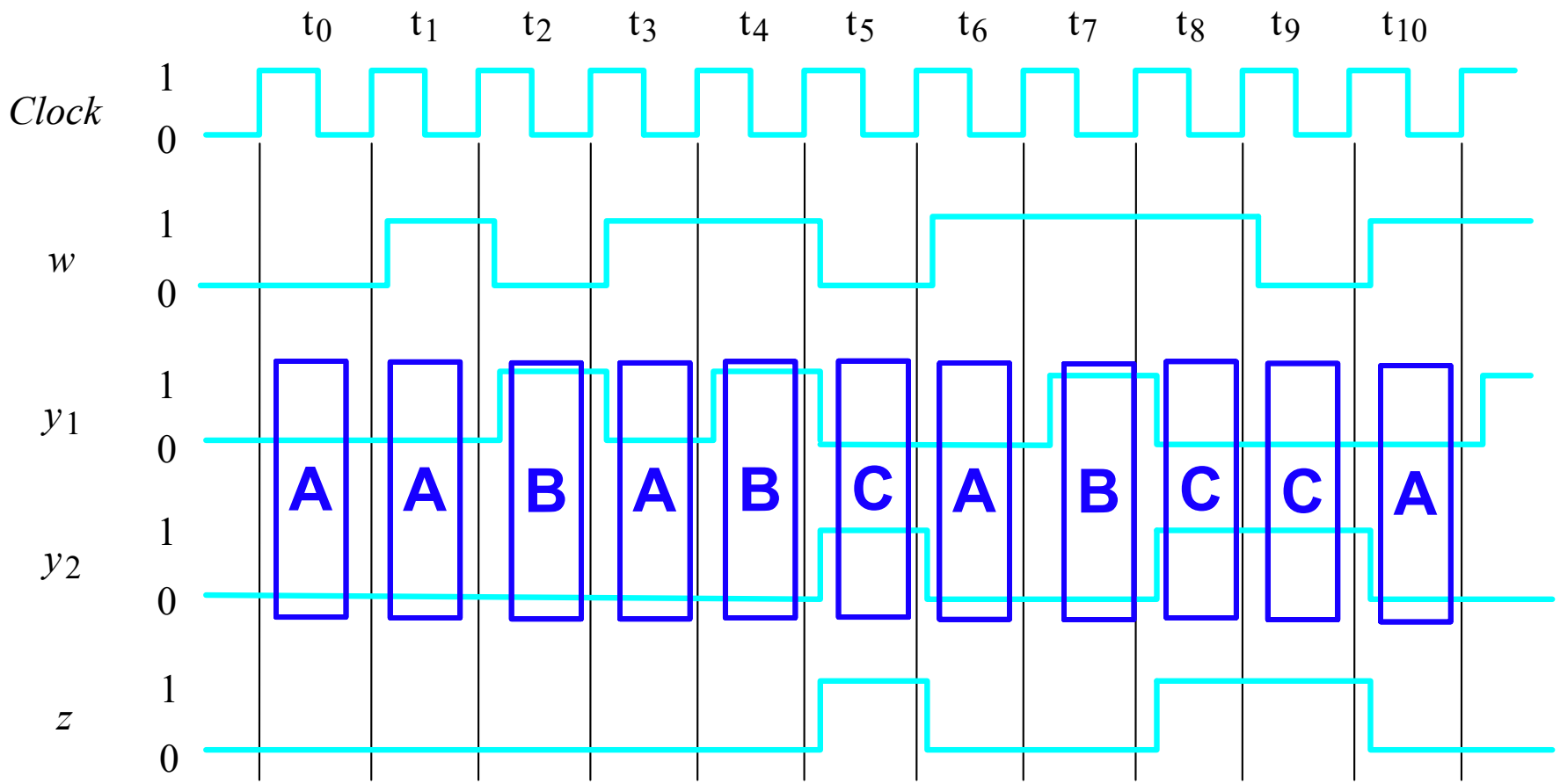
Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

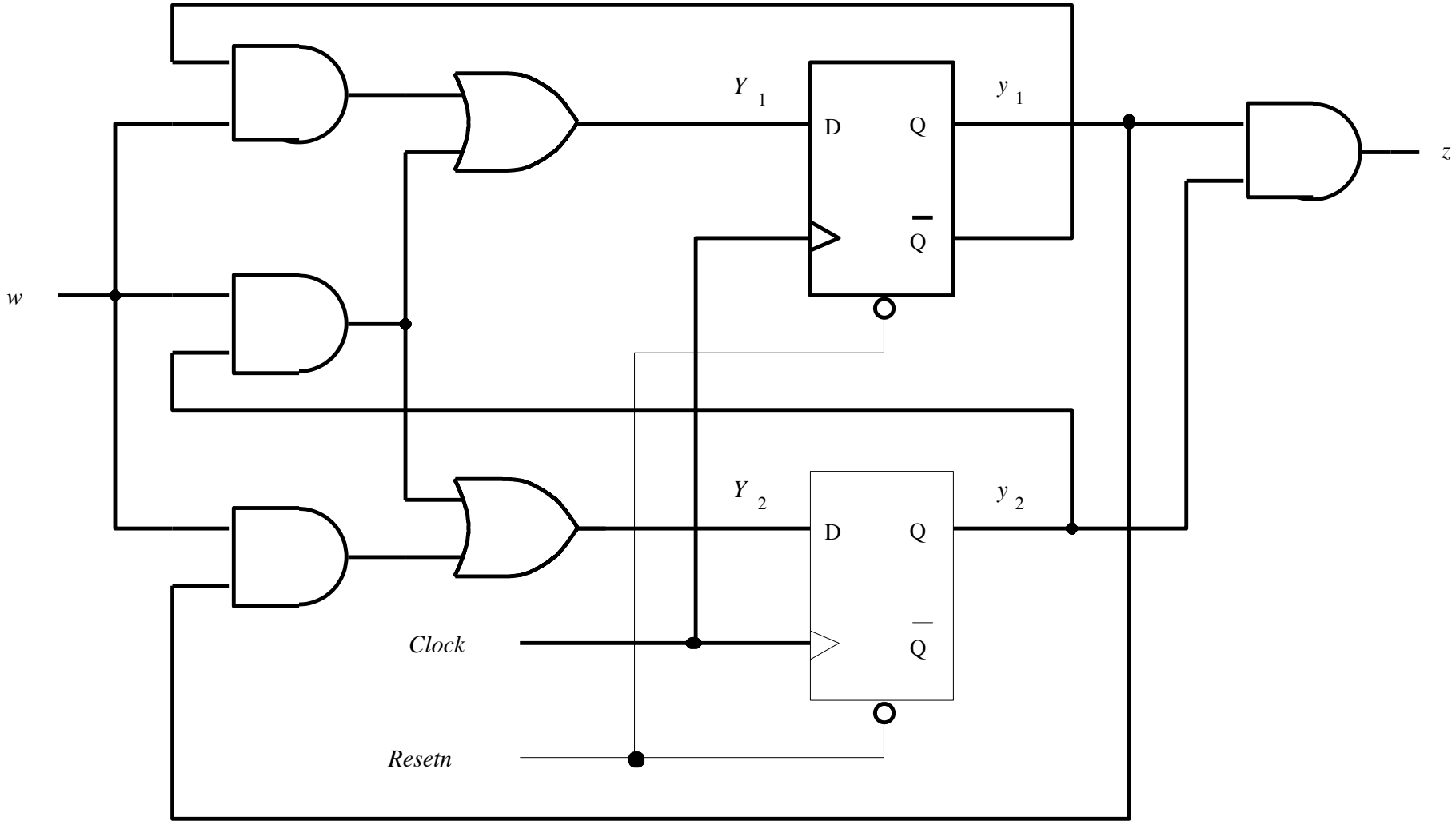


Summary: Designing a Moore Machine

- Obtain the circuit specification.
- Derive a state diagram.
- Derive the state table.
- Decide on a state encoding.
- Encode the state table.
- Derive the output logic and next-state logic.
- Add a reset signal.

Reverse Engineering

What does this circuit do?



[Figure 6.75 from the textbook]

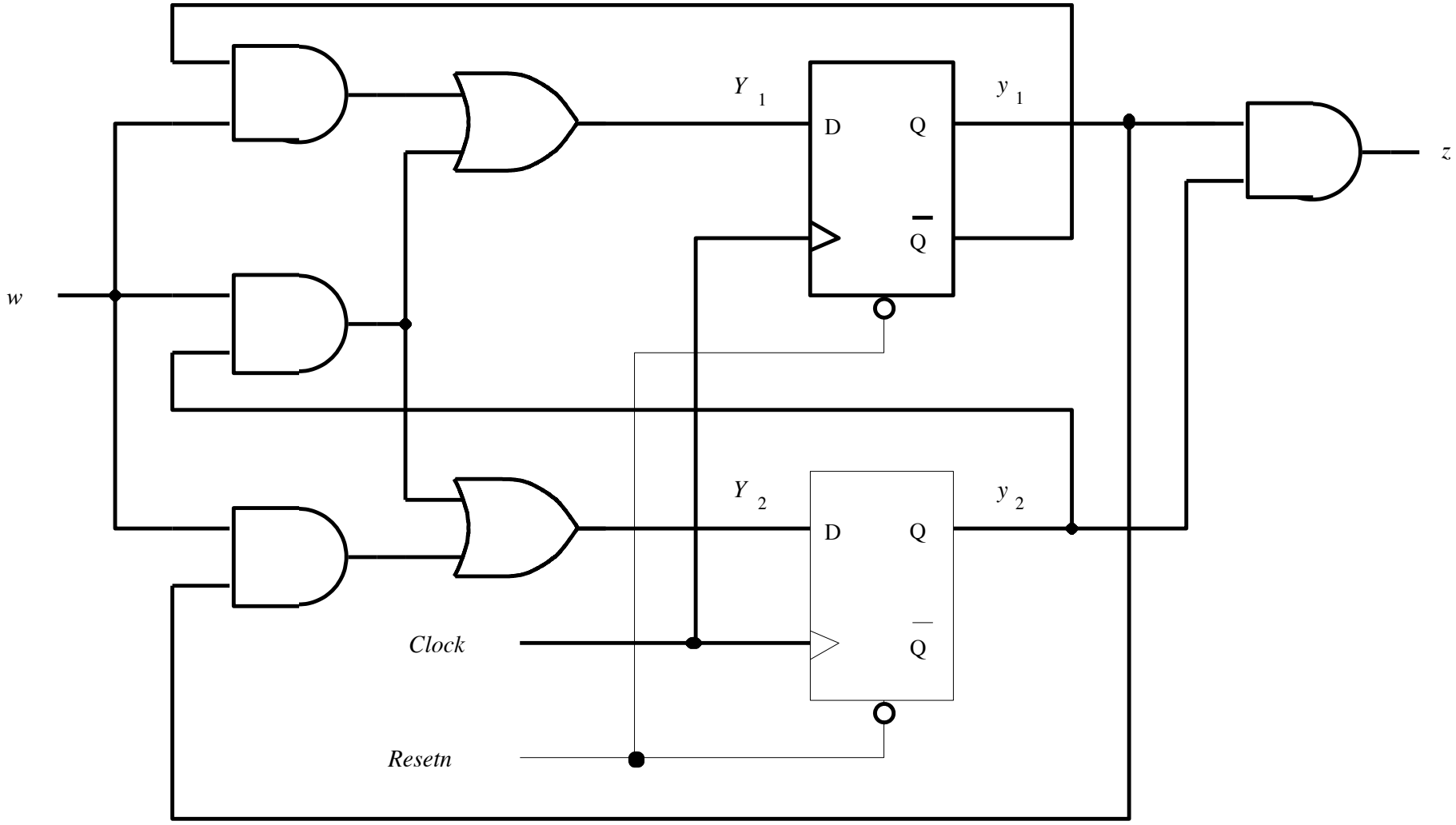
Approach

- Find the flip-flops
- Outputs of the flip-flops = present state variables
- Inputs of the flip-flops determine the next state variables
- Determine the logical expressions for the outputs
- Given this info it is easy to do the state-assigned table
- Next do the state table
- Finally, draw the state diagram.

Goal

- **Given a circuit diagram for a synchronous sequential circuit, the goal is to figure out the FSM**
- **Figure out the present state variables, the next state variables, the state-assigned table, the state table, and finally the state diagram.**
- **In other words, the goal is to reverse engineer the circuit.**

What does this circuit do?

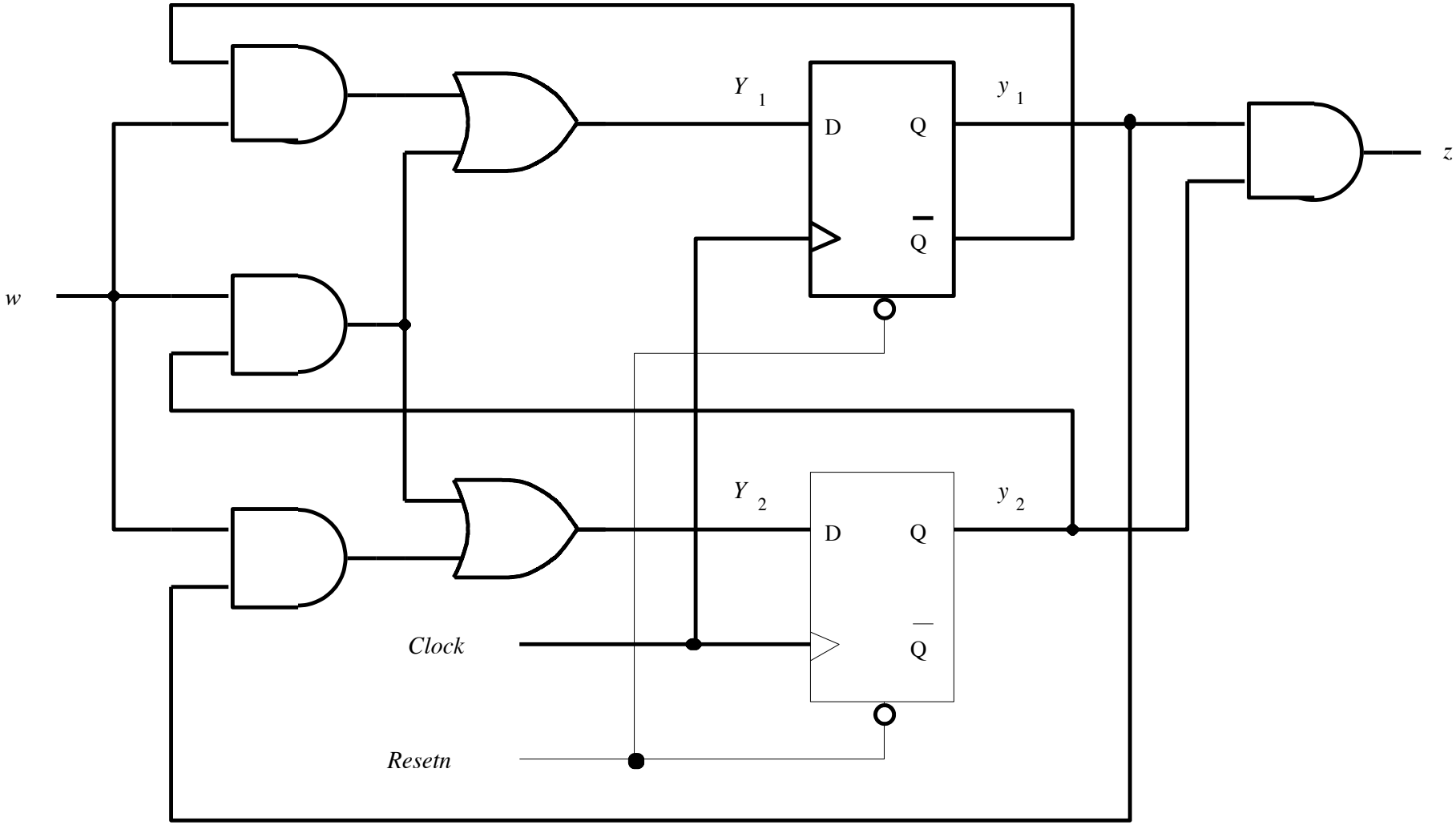


[Figure 6.75 from the textbook]

Approach

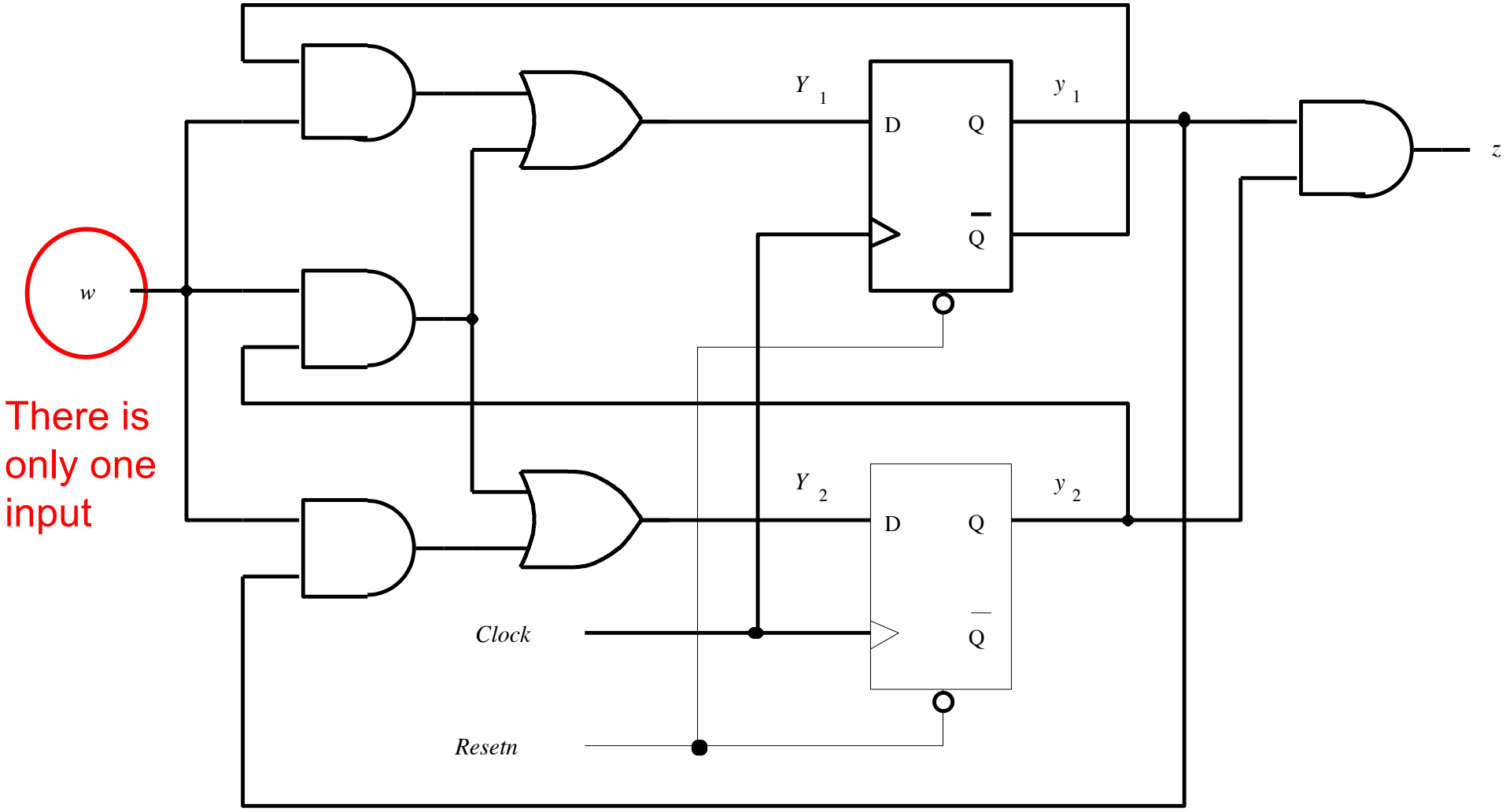
- Find the flip-flops
- Outputs of the flip-flops = present state variables
- Inputs of the flip-flops determine the next state variables
- Determine the logical expressions for the outputs
- Given this info it is easy to do the state-assigned table
- Next do the state table
- Finally, draw the state diagram.

Where are the inputs?



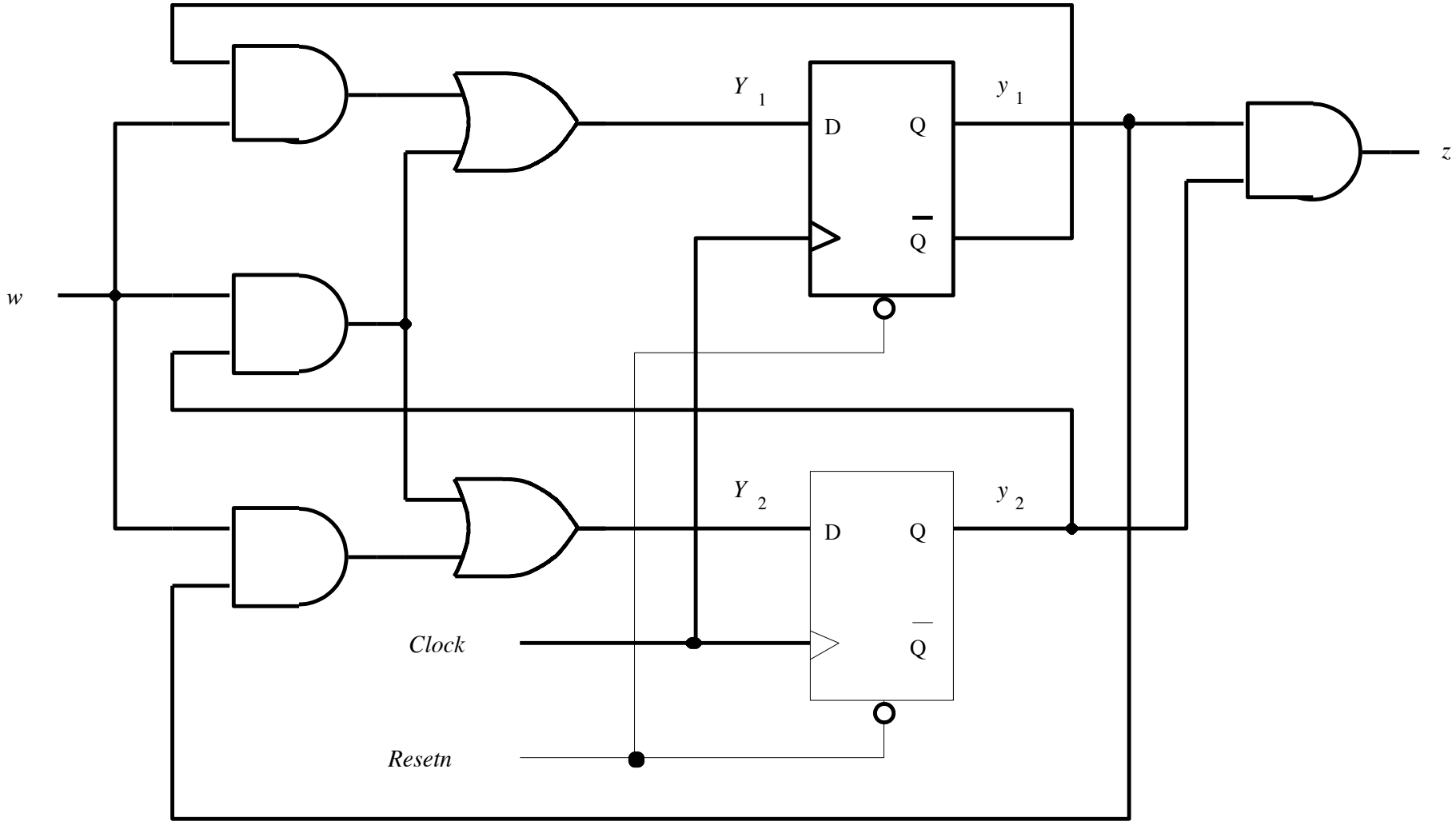
[Figure 6.75 from the textbook]

Where are the inputs?



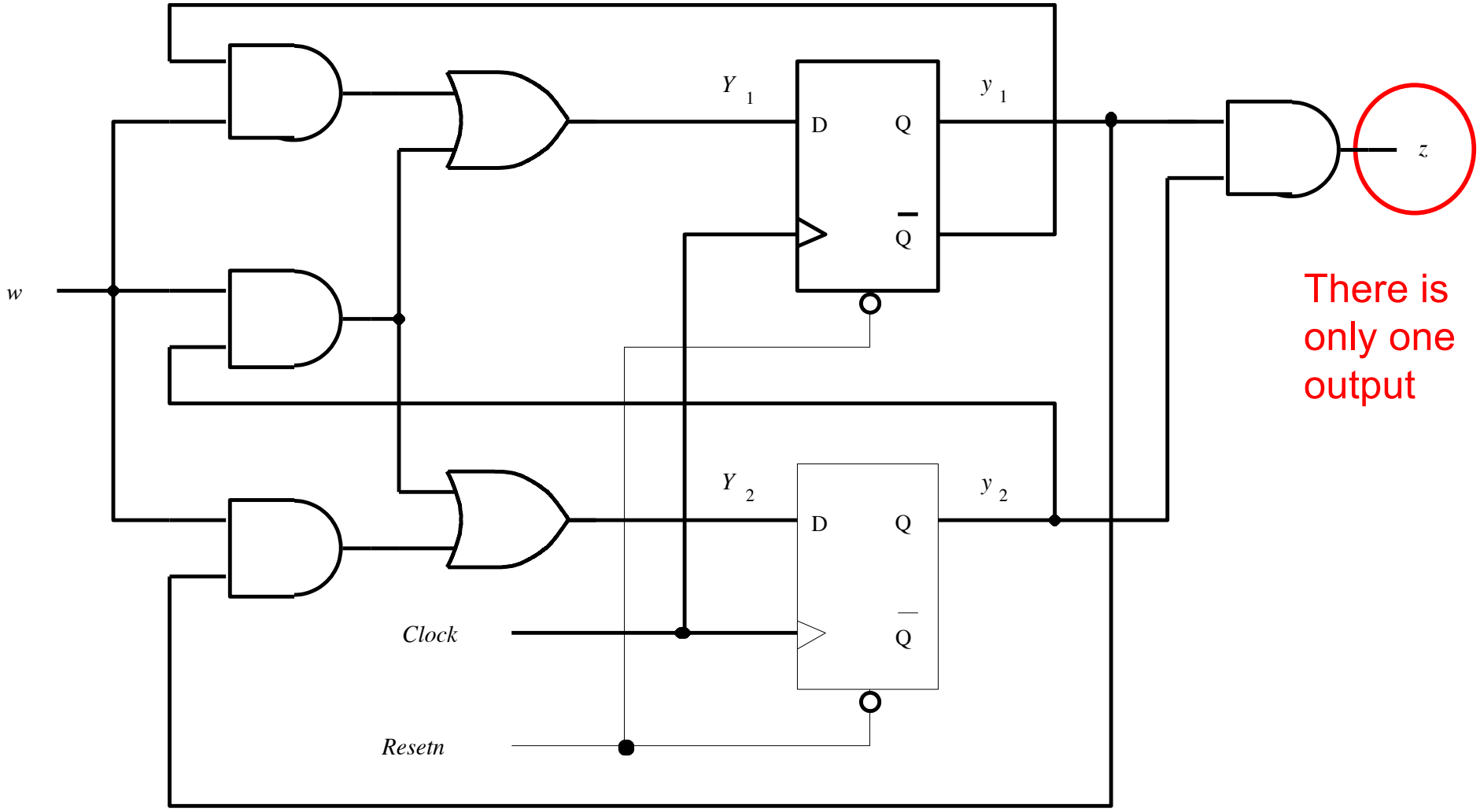
[Figure 6.75 from the textbook]

Where are the outputs?



[Figure 6.75 from the textbook]

Where are the outputs?

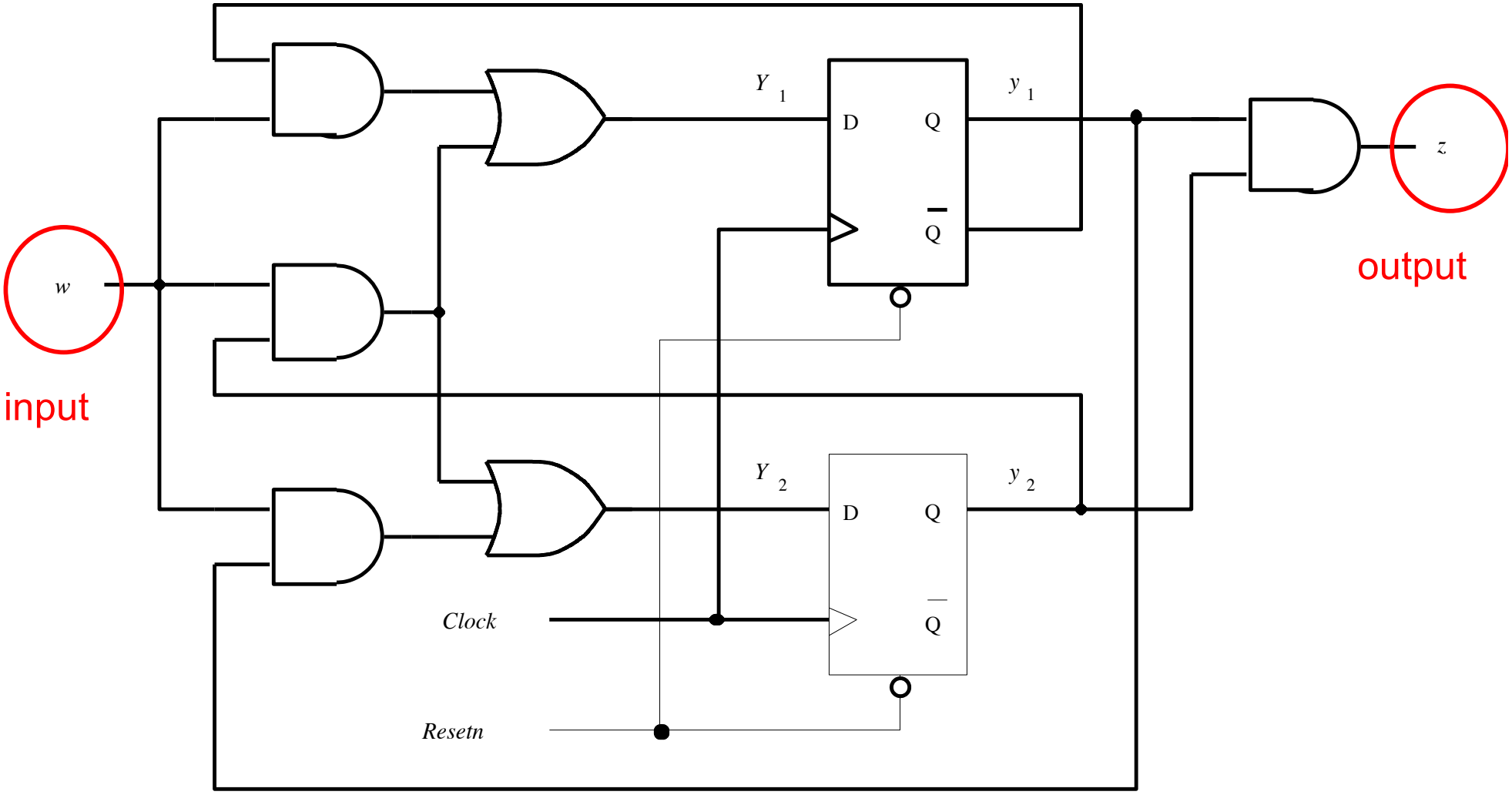


There is only one output

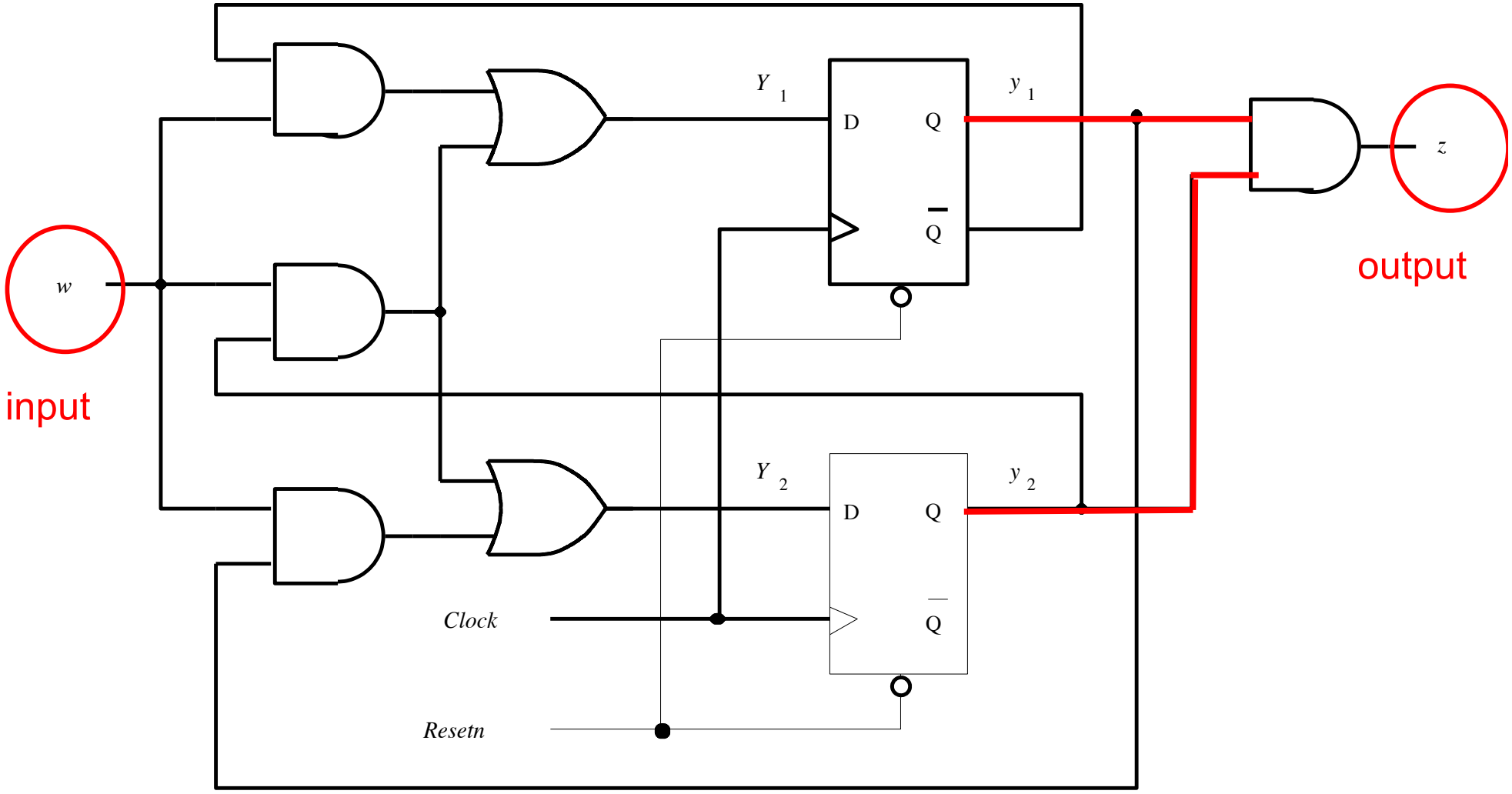
[Figure 6.75 from the textbook]

Where kind of machine is this?

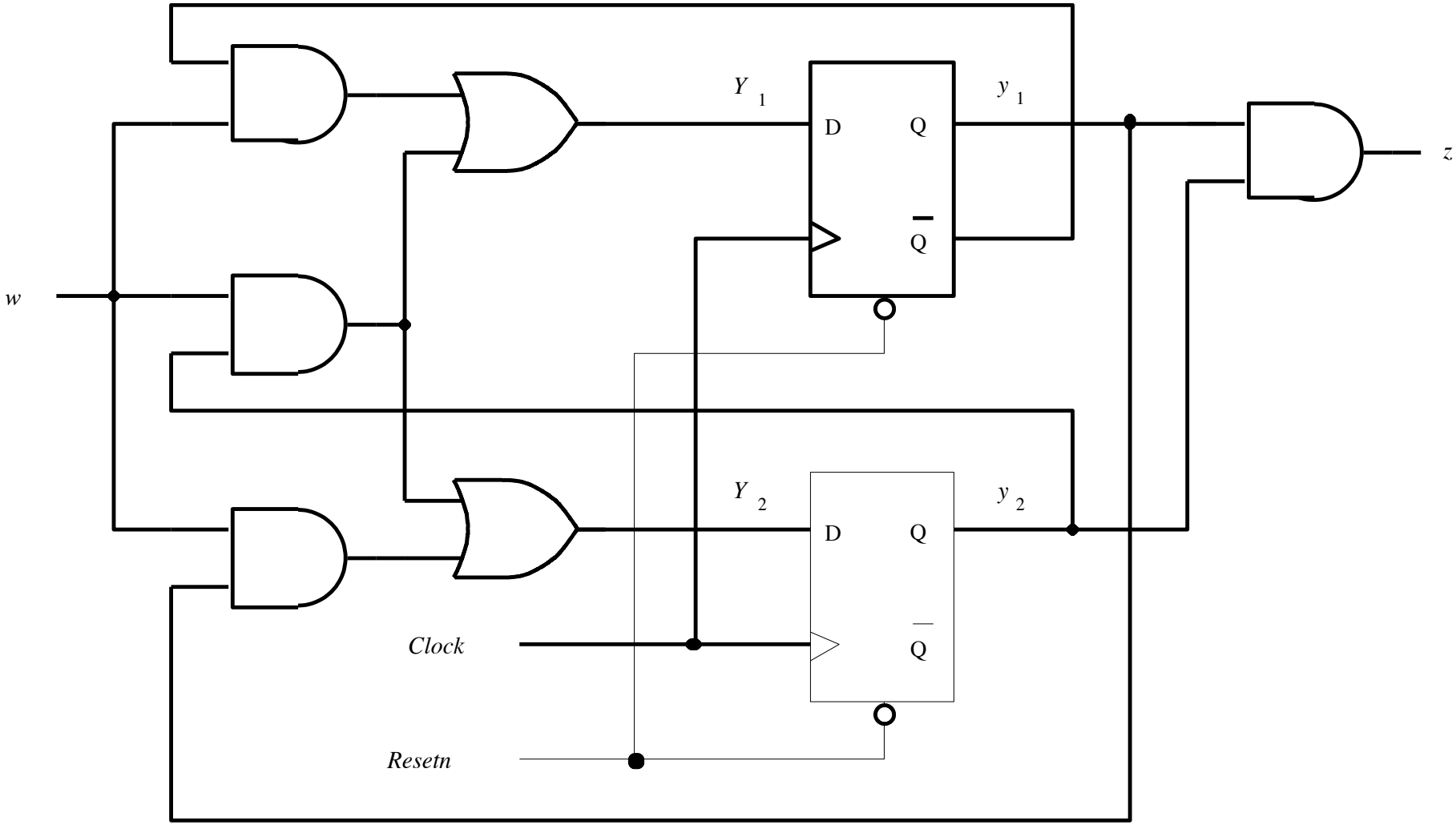
Moore or Mealy?



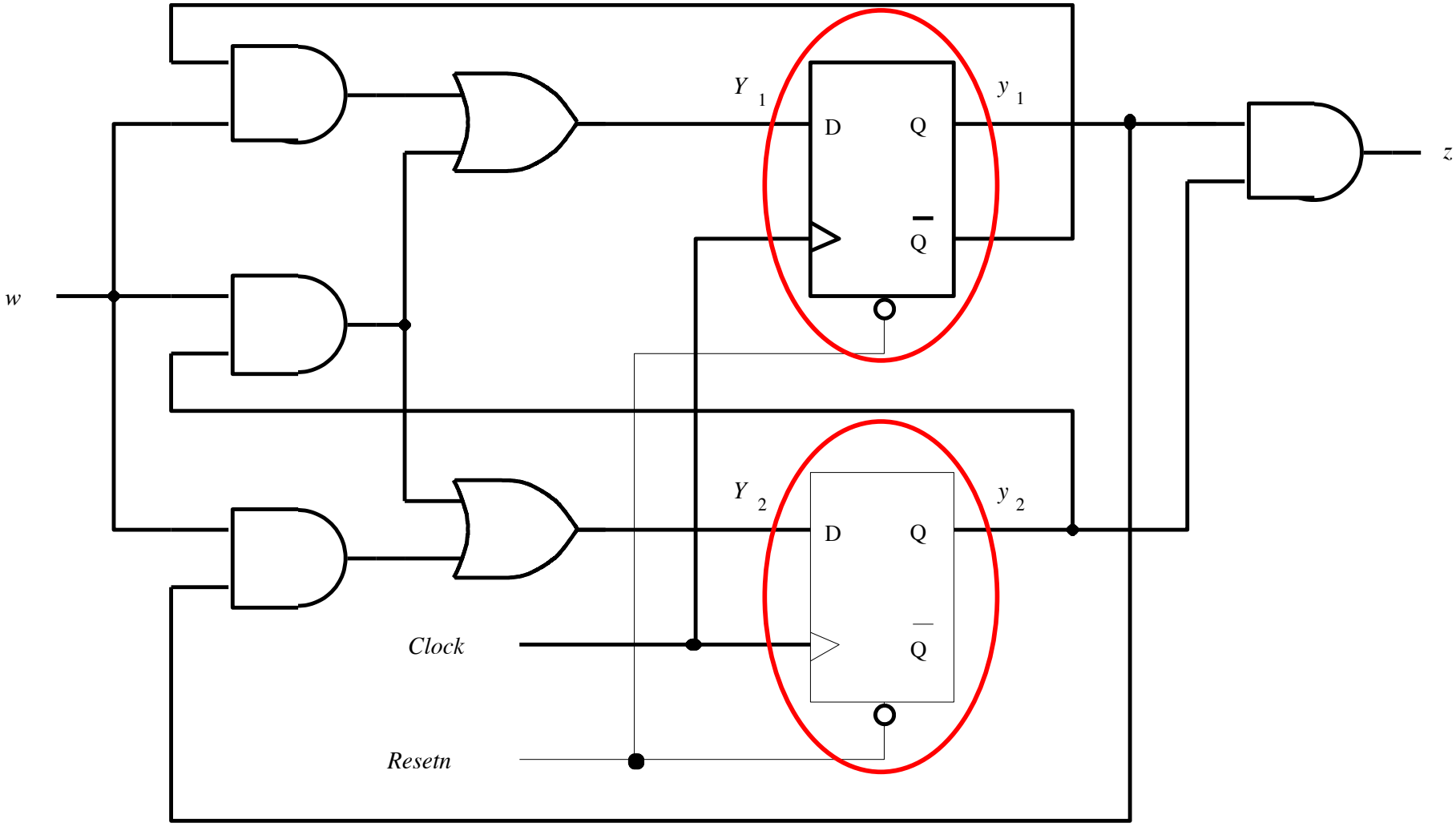
Moore: because the output does not depend directly on the primary input



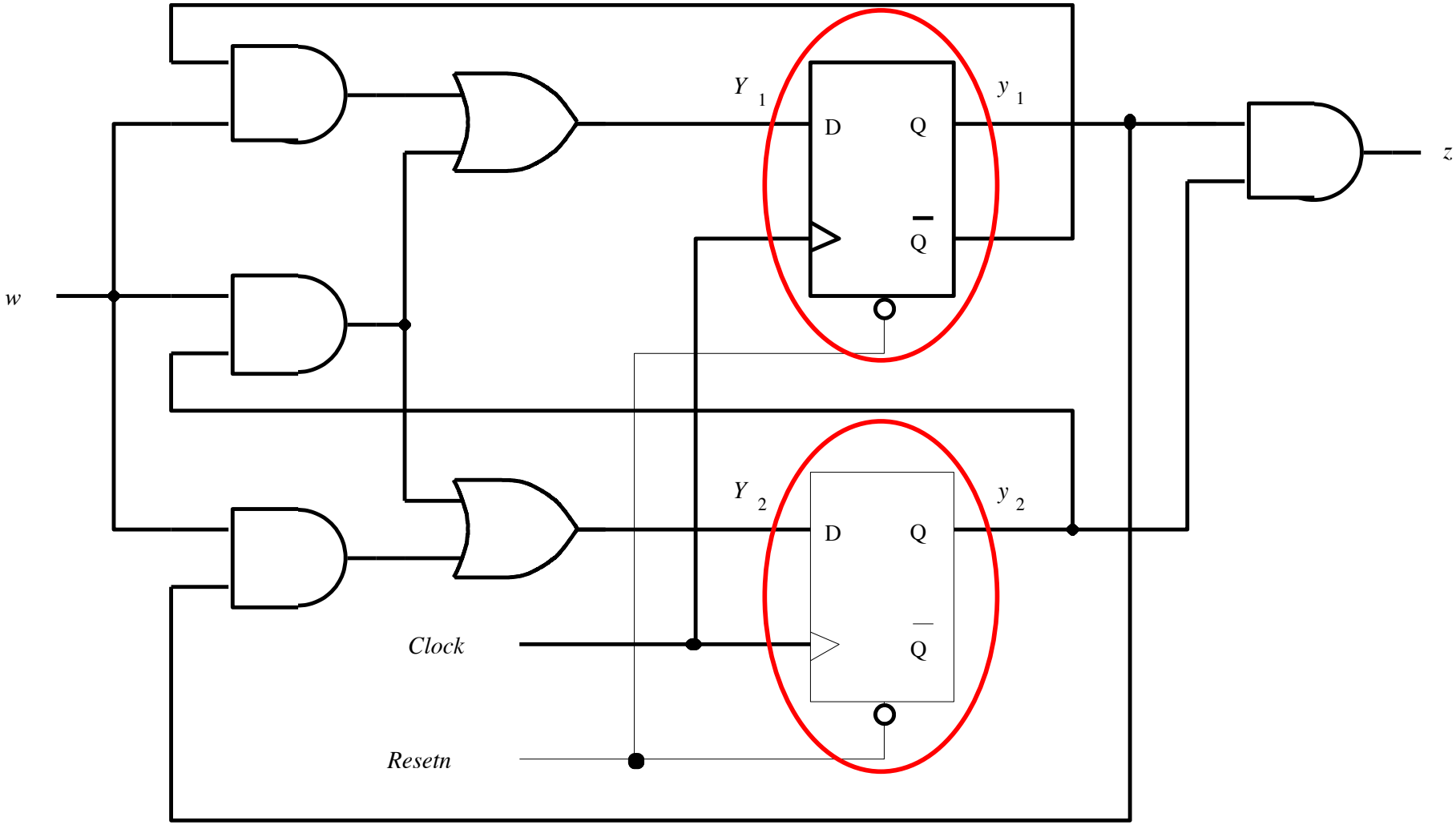
Where are the memory elements?



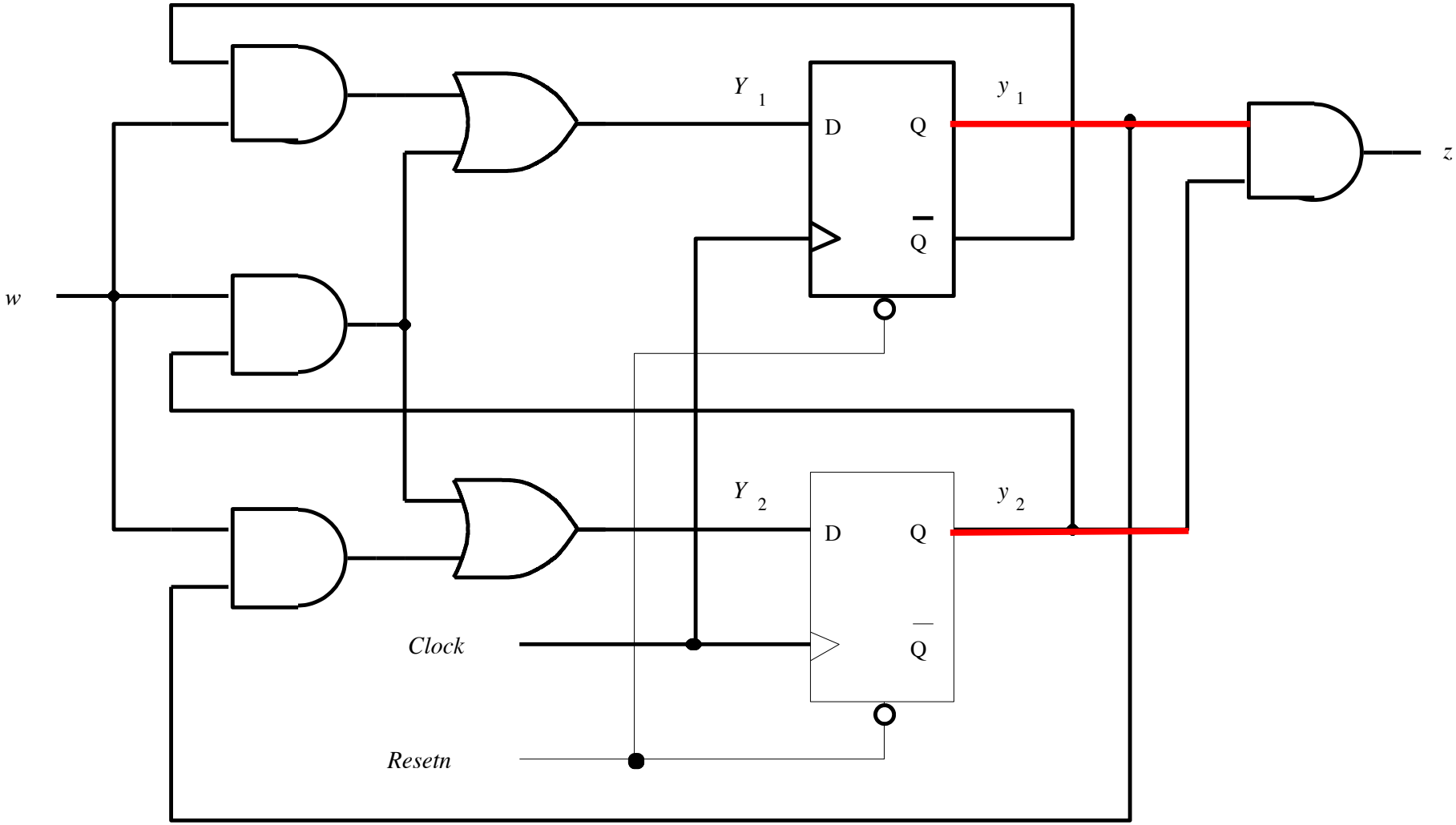
Where are the memory elements?



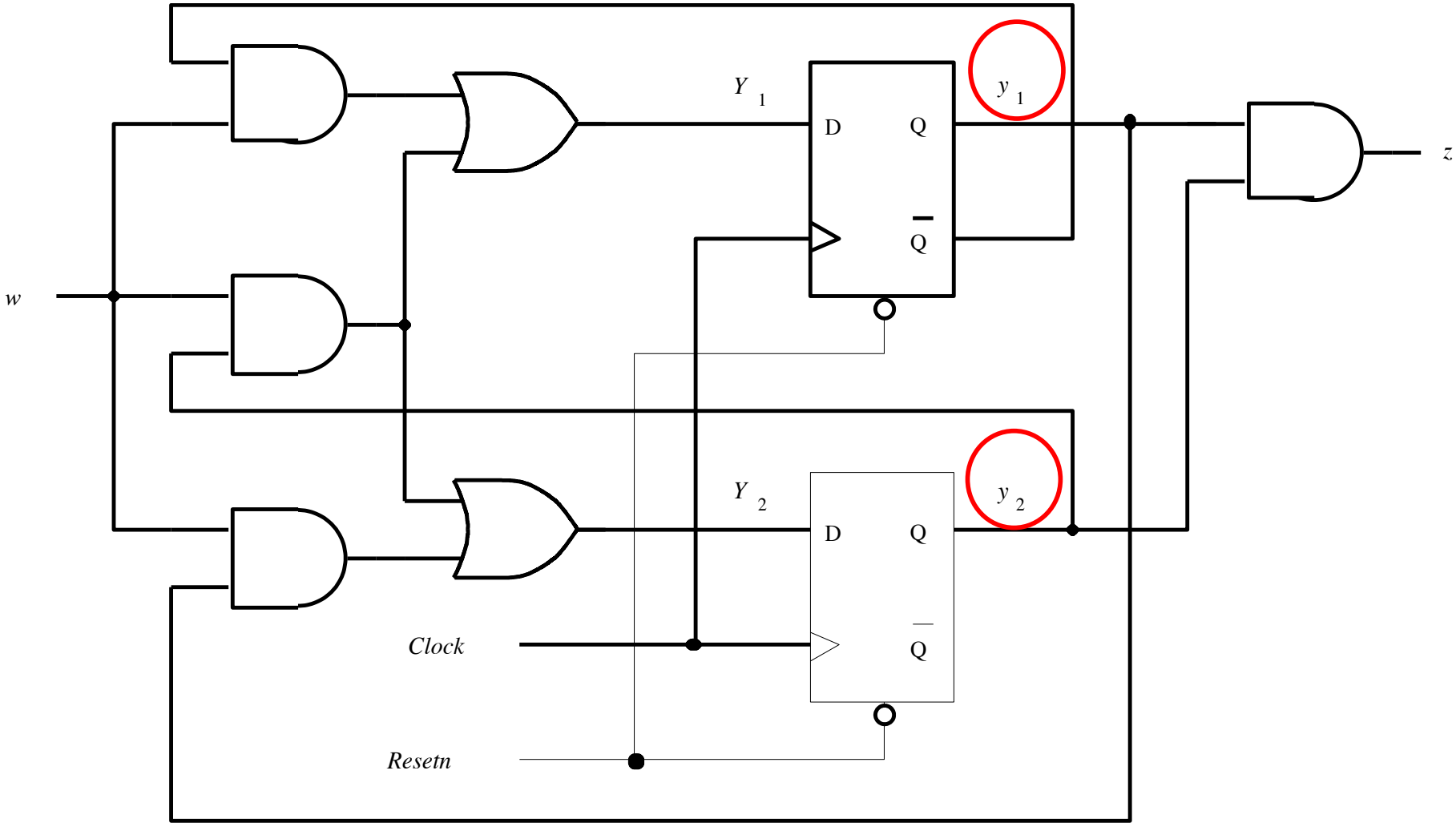
Where are the outputs of the flip-flops?



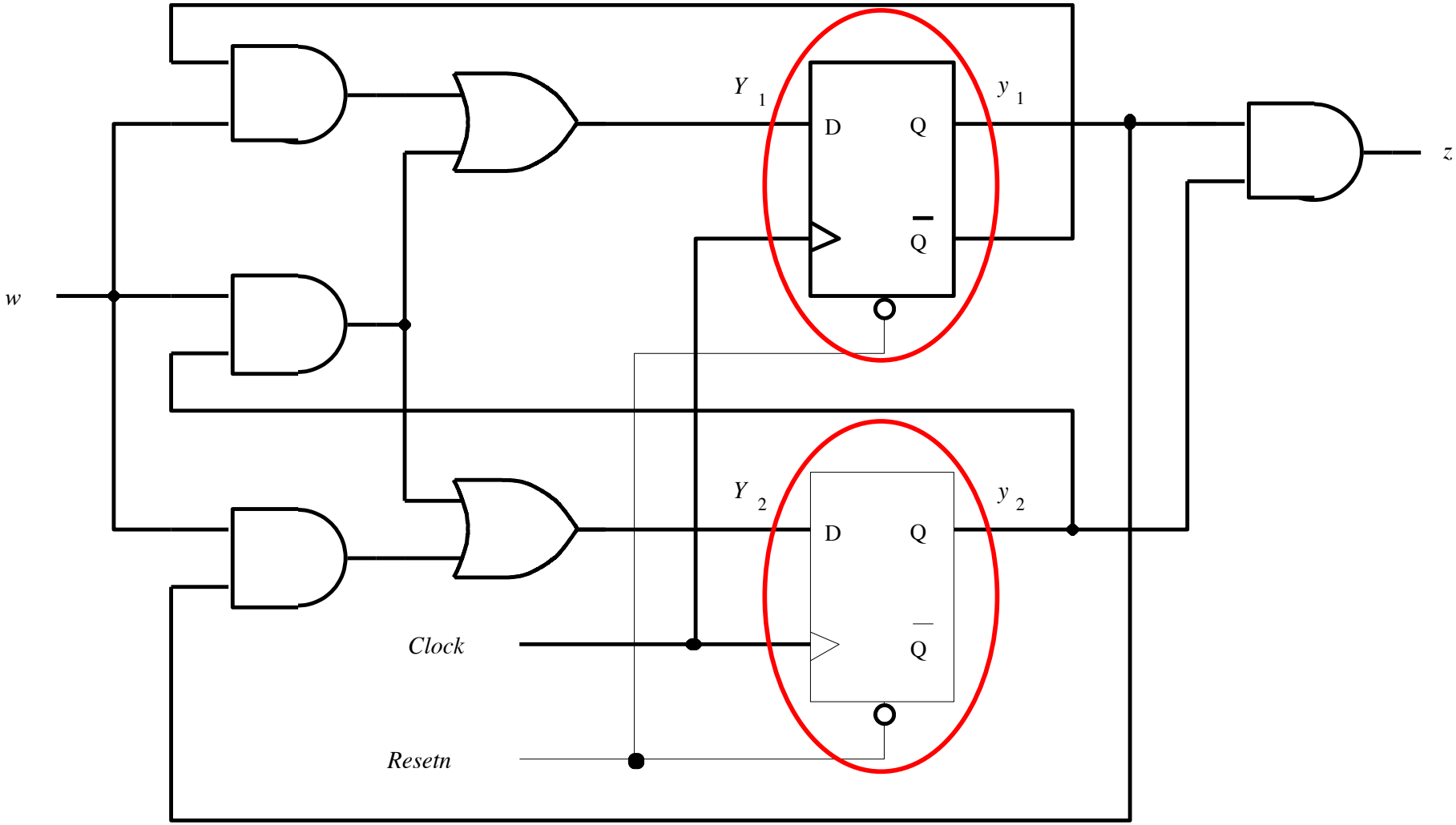
Where are the outputs of the flip-flops?



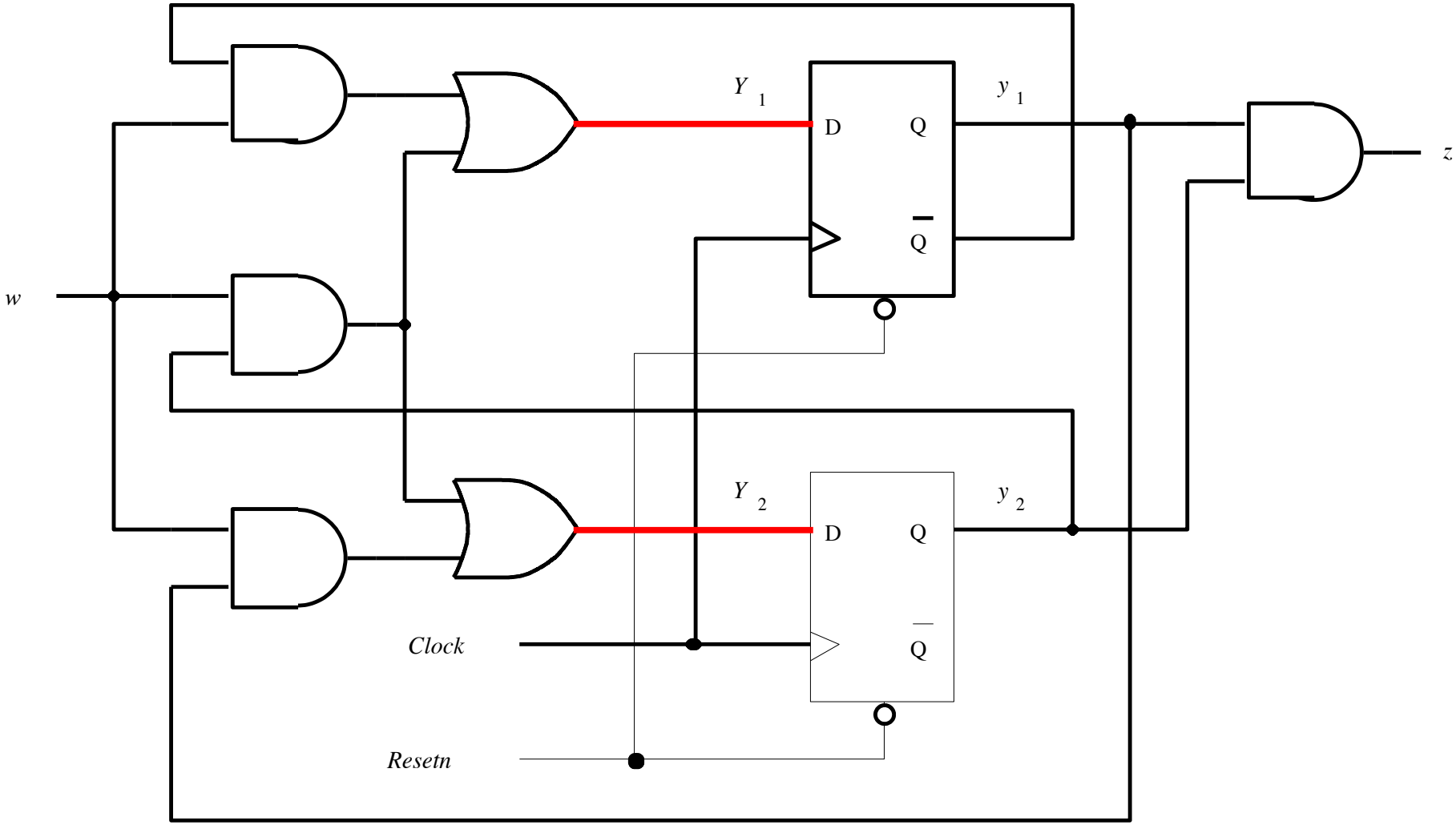
These are the present-state variables



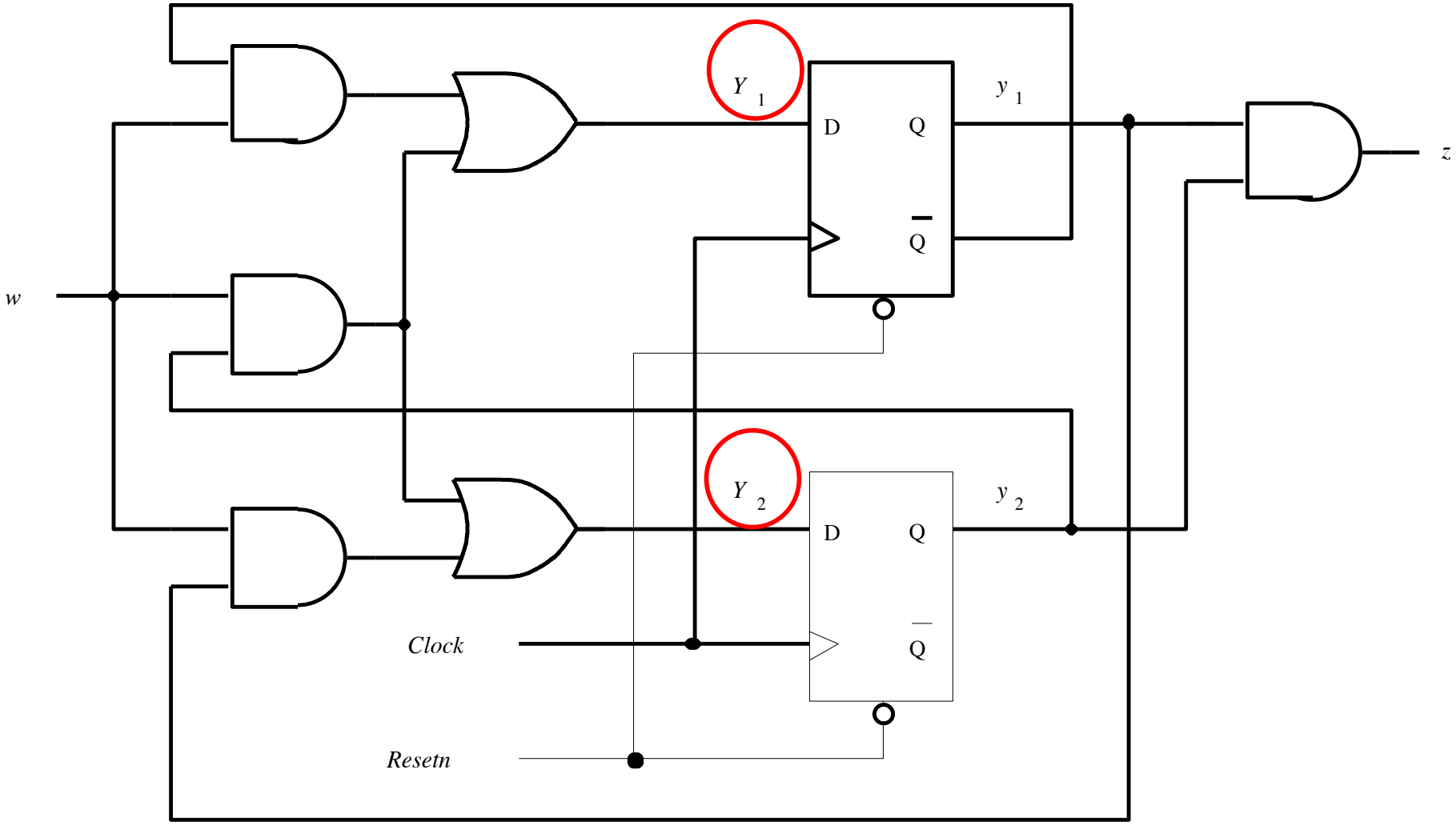
Where are the inputs of the flip-flops?



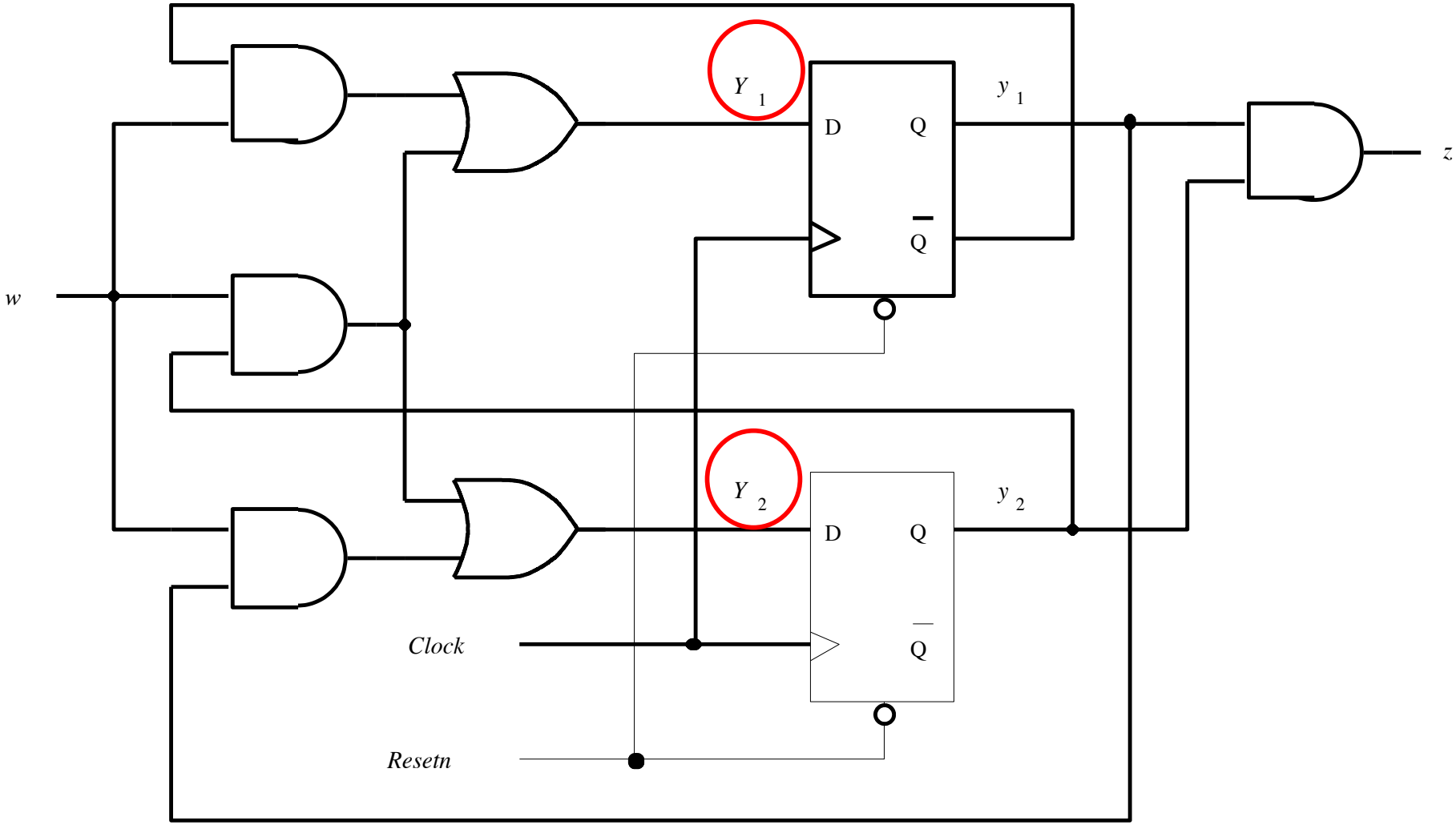
Where are the inputs of the flip-flops?



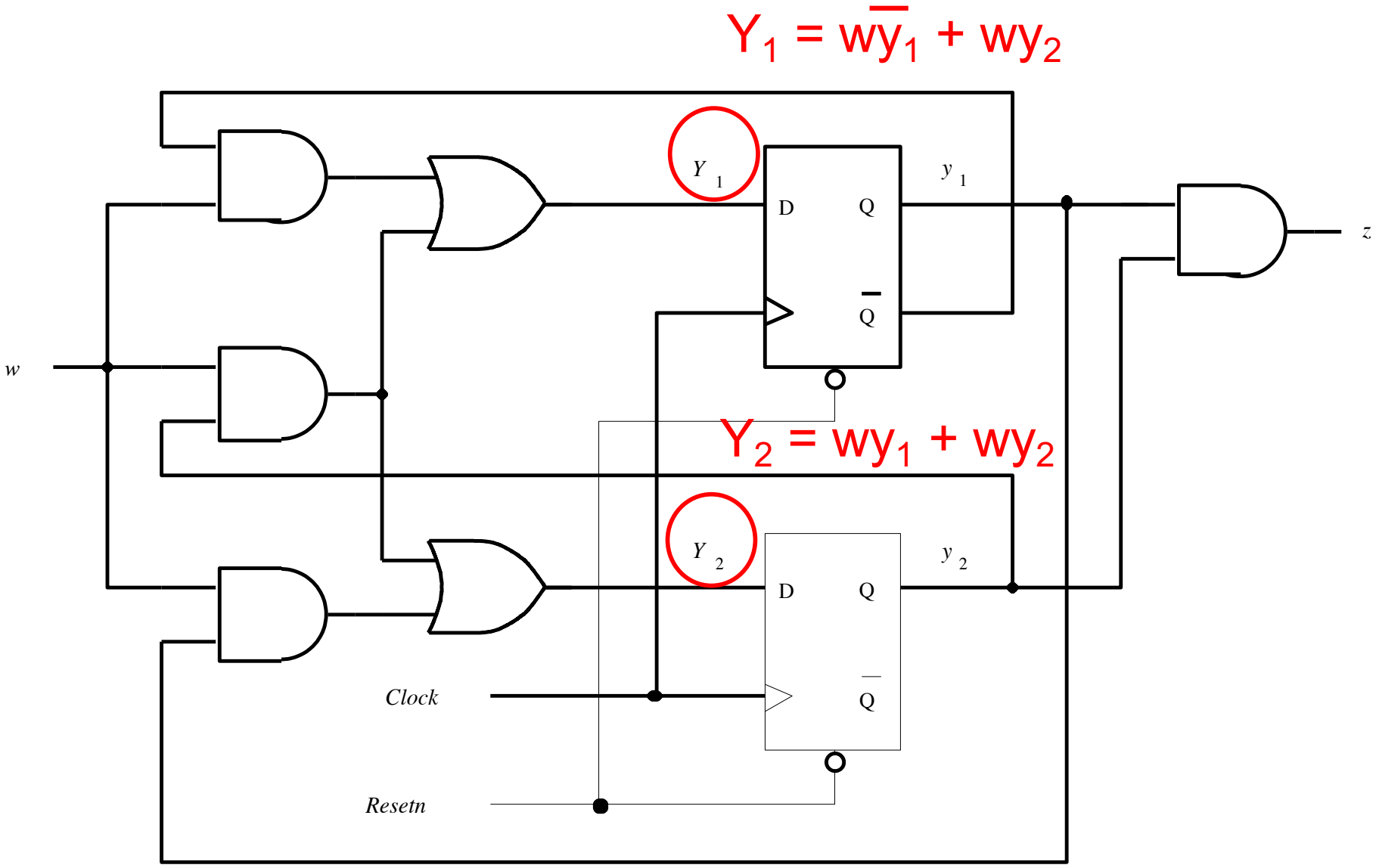
These are the next-state variables



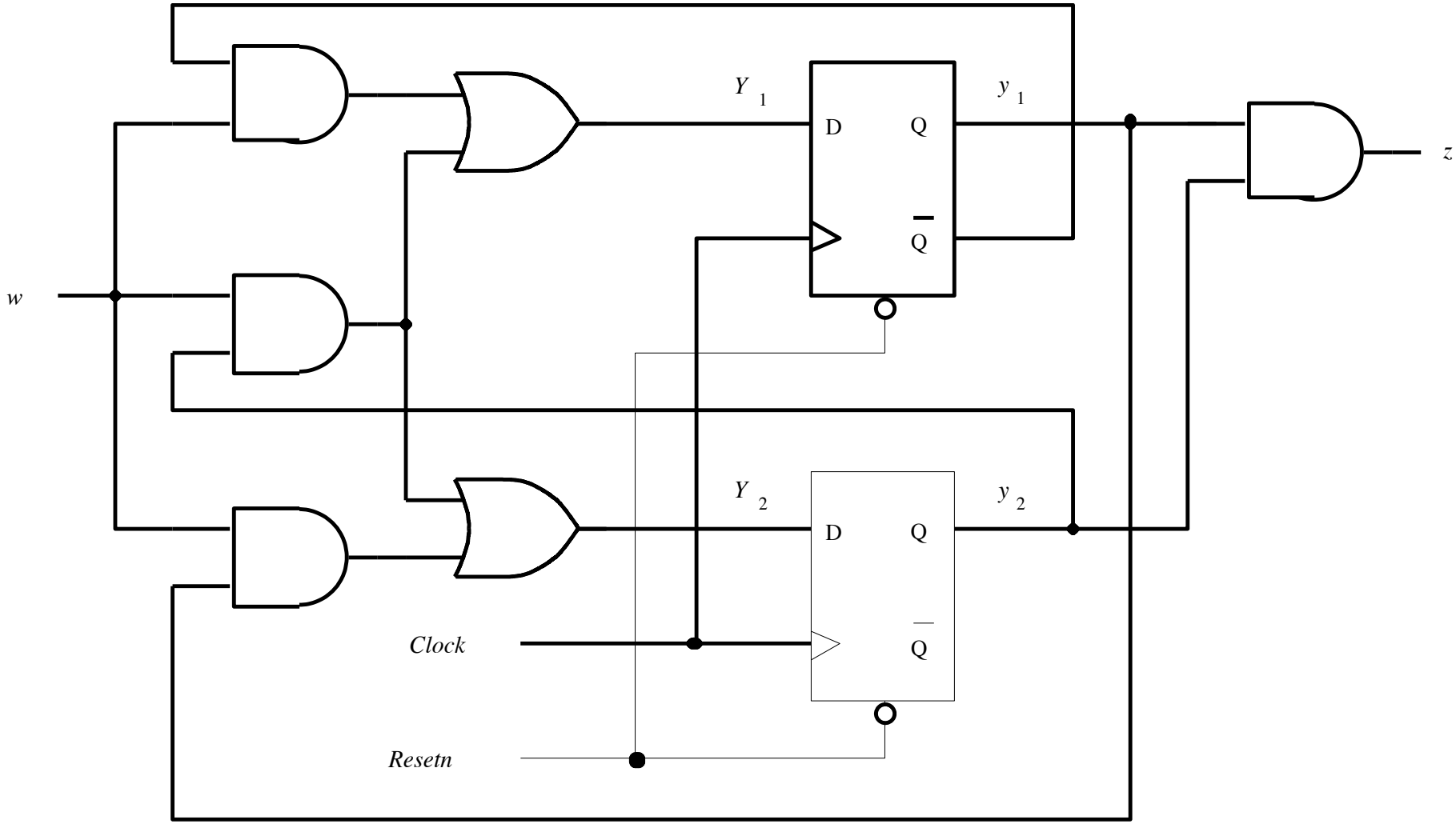
What are their logic expressions?



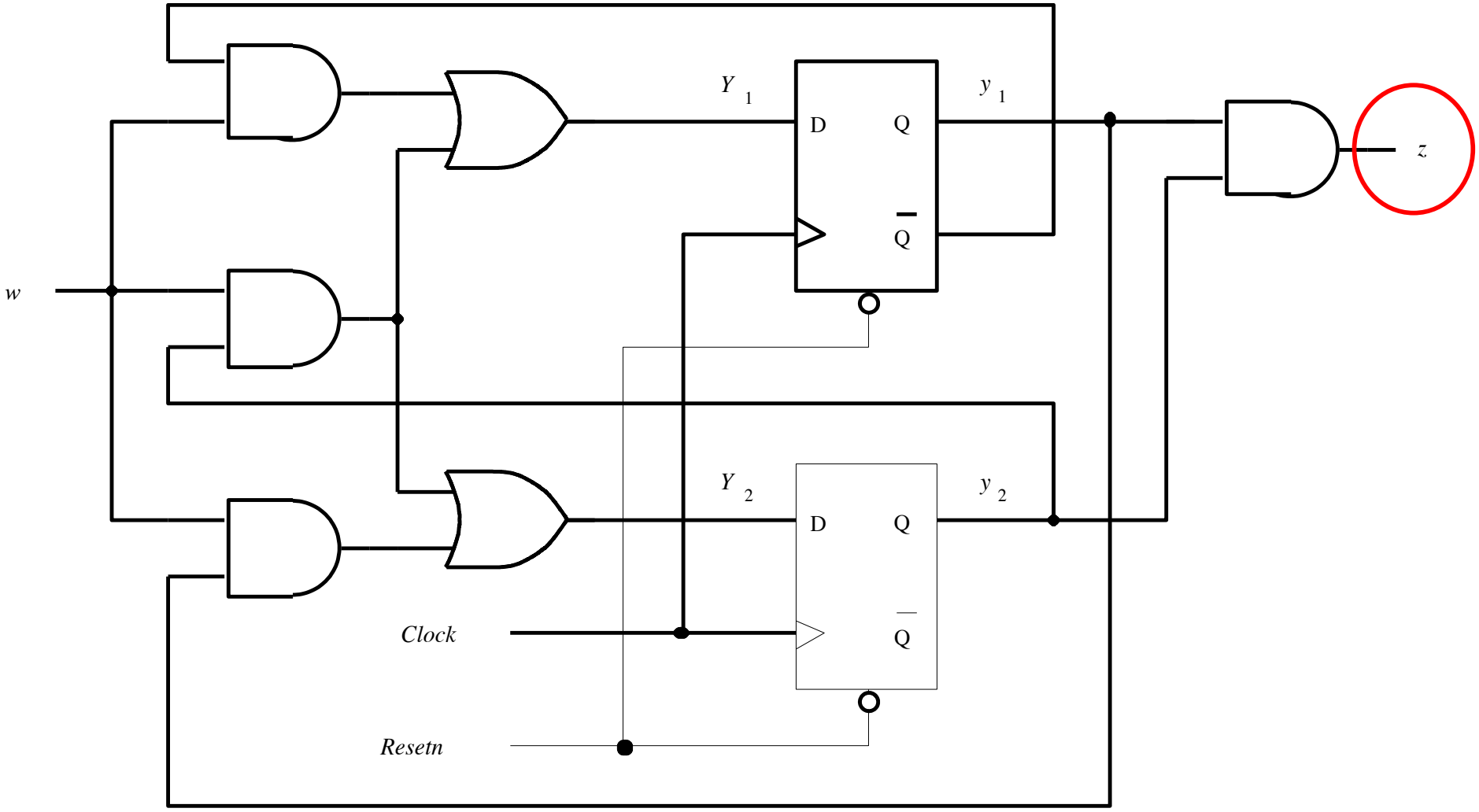
What are their logic expressions?



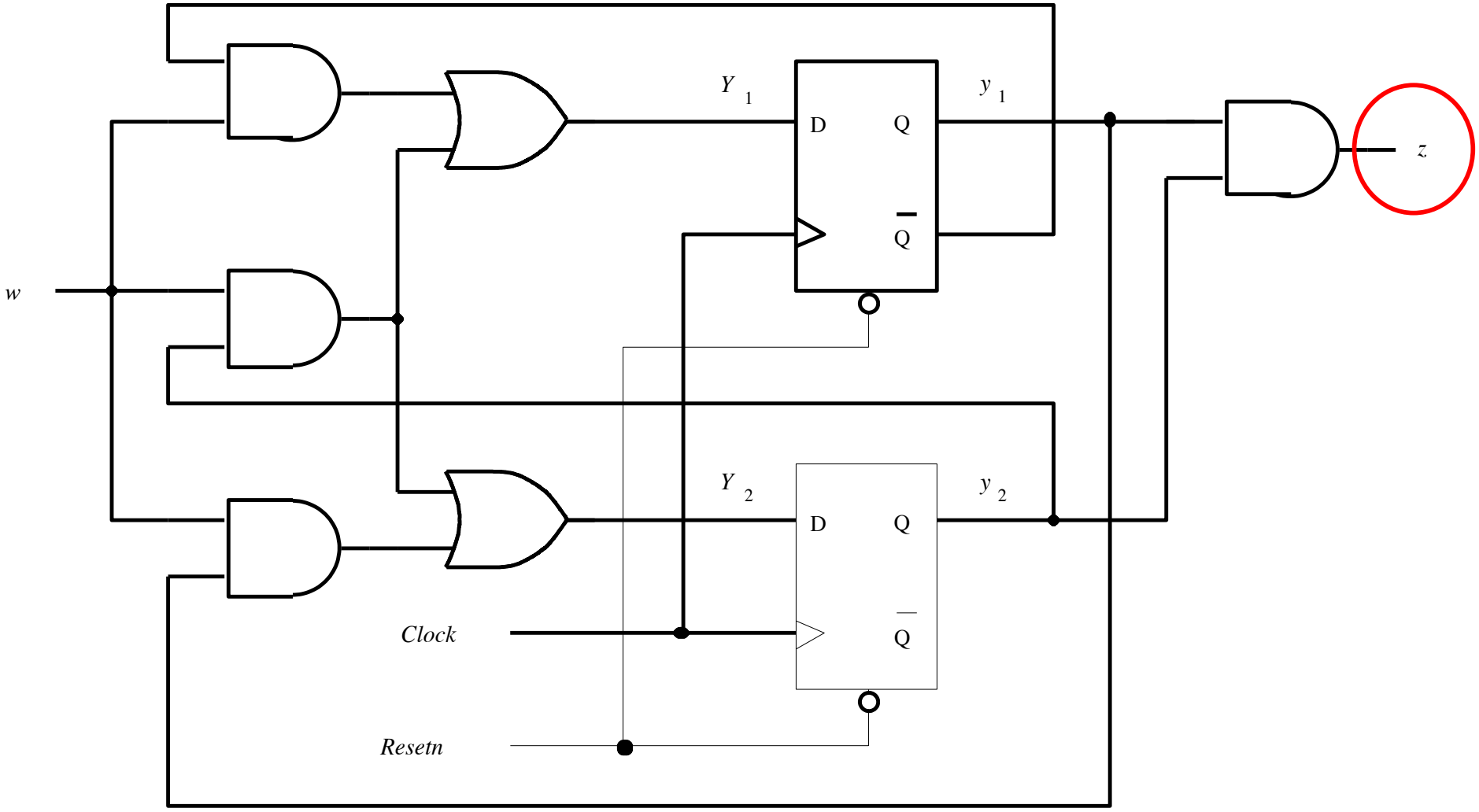
Where is the output, again?



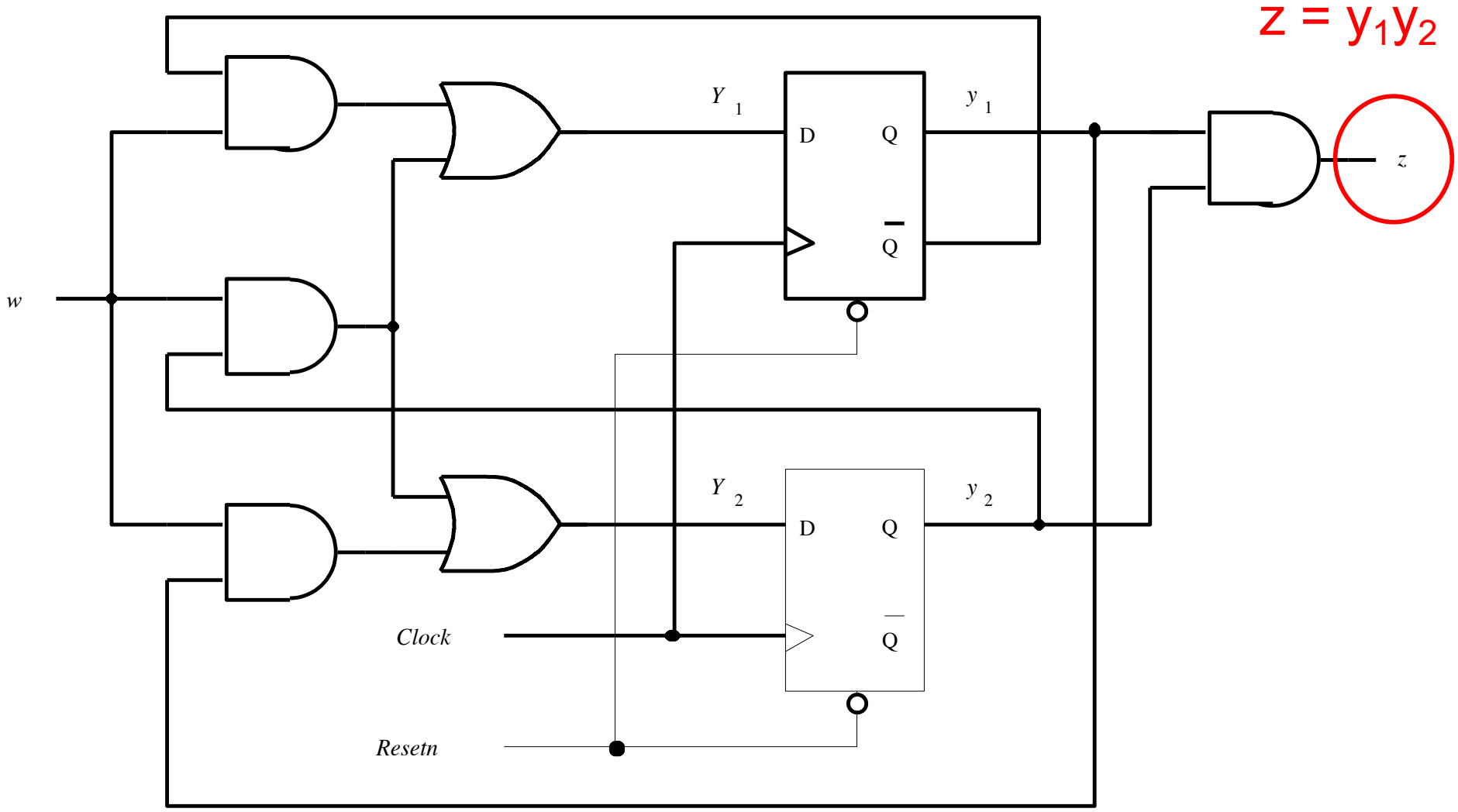
Where is the output, again?



What is its logic expression?



What is its logic expression?



This is what we have to work with now (we don't need the circuit anymore)

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1 y_2$$

Let's derive the state-assigned table

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1y_2$$

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0			
0 1			
1 0			
1 1			

Let's derive the state-assigned table

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1y_2$$

Present state y_2y_1	Next State		Output
	w = 0	w = 1	z
	Y_2Y_1	Y_2Y_1	
0 0			
0 1			
1 0			
1 1			

Let's derive the state-assigned table

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1y_2$$

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0			0
0 1			0
1 0			0
1 1			1

Let's derive the state-assigned table

$$Y_1 = w\overline{y_1} + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1 y_2$$

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0			0
0 1			0
1 0			0
1 1			1

Let's derive the state-assigned table

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1 y_2$$

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0	0	1	0
0 1	0	0	0
1 0	0	1	0
1 1	0	1	1

Let's derive the state-assigned table

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1 y_2$$

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0	0	1	0
0 1	0	0	0
1 0	0	1	0
1 1	0	1	1

Let's derive the state-assigned table

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1y_2$$

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

We don’t need the logic expressions anymore

$$Y_1 = w\overline{y}_1 + wy_2$$

$$Y_2 = wy_1 + wy_2$$

$$z = y_1y_2$$

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

We don’t need the logic expressions anymore

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state w = 0	Next state w = 1	Output z
A			
B			
C			
D			

State table

Present state y_2y_1	Next State w = 0 Y_2Y_1	Next State w = 1 Y_2Y_1	Output z
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			
B			
C			
D			

State table

Present state	Next State		Output
	w = 0	w = 1	
y_2y_1	Y_2Y_1	Y_2Y_1	z
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	
A	A		
B	A		
C	A		
D	A		

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	
A	A		
B	A		
C	A		
D	A		

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	
B	A	C	
C	A	D	
D	A	D	

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	
B	A	C	
C	A	D	
D	A	D	

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	
B	A	C	
C	A	D	
D	A	D	

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

The output is the same in both tables

The two tables for the initial circuit

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

[Figure 6.76 from the textbook]

We don’t need the state-assigned table anymore

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1

State-assigned table

[Figure 6.76 from the textbook]

We don’t need the state-assigned table anymore

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table

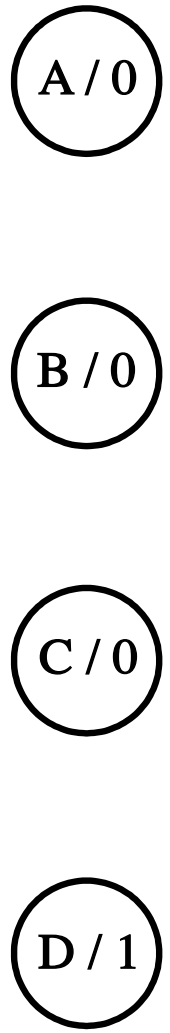
Let's Draw the State Diagram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

Let's Draw the State Diagram

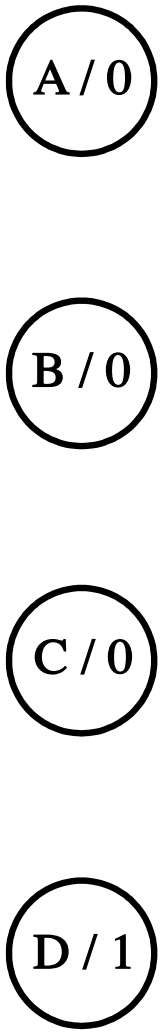
Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

Because this is a Moore machine
the output is tied to the state



Let's Draw the State Diagram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

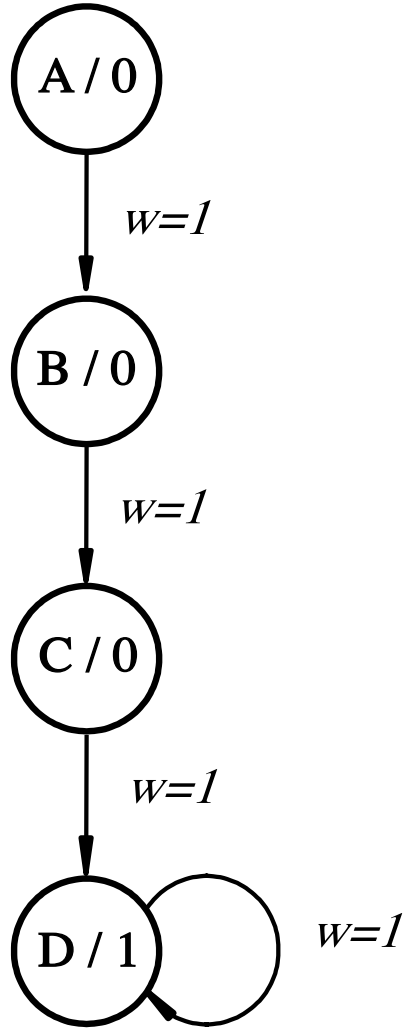


All transitions when the input w is equal to 1

Let's Draw the State Diagram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

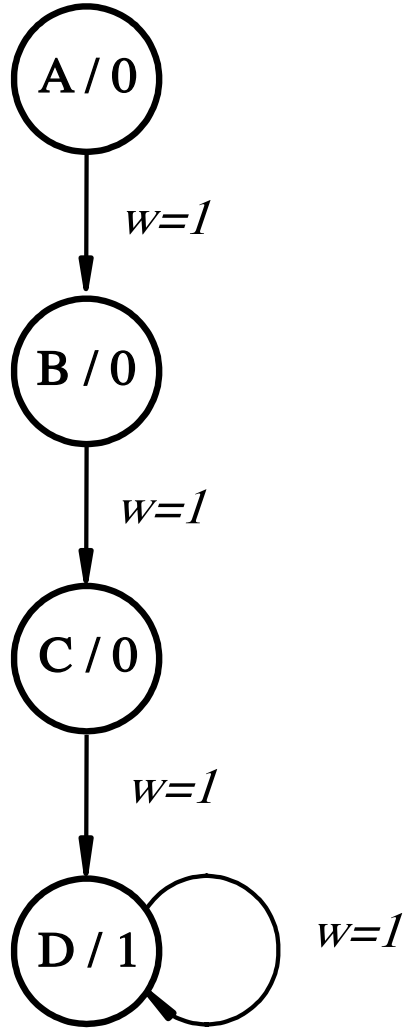
All transitions when the input w is equal to 1



Let's Draw the State Diagram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

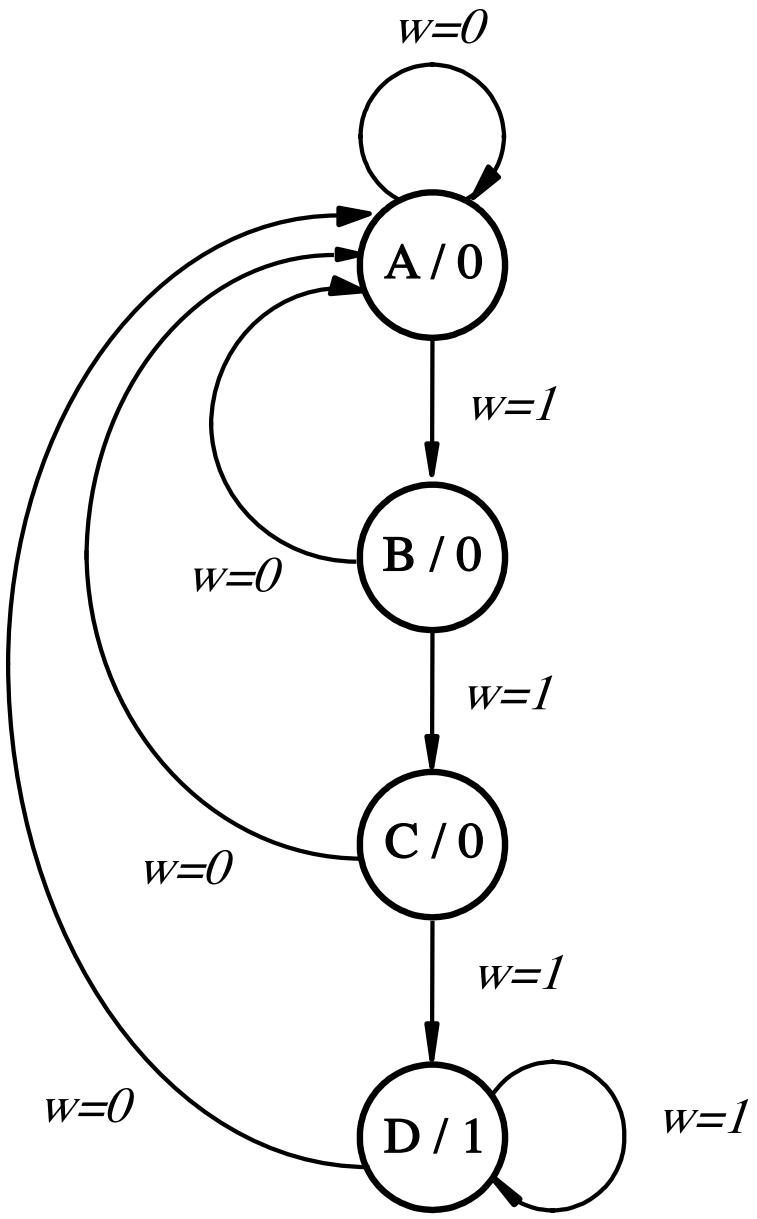
All transitions when the input w is equal to 0



Let's Draw the State Diagram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

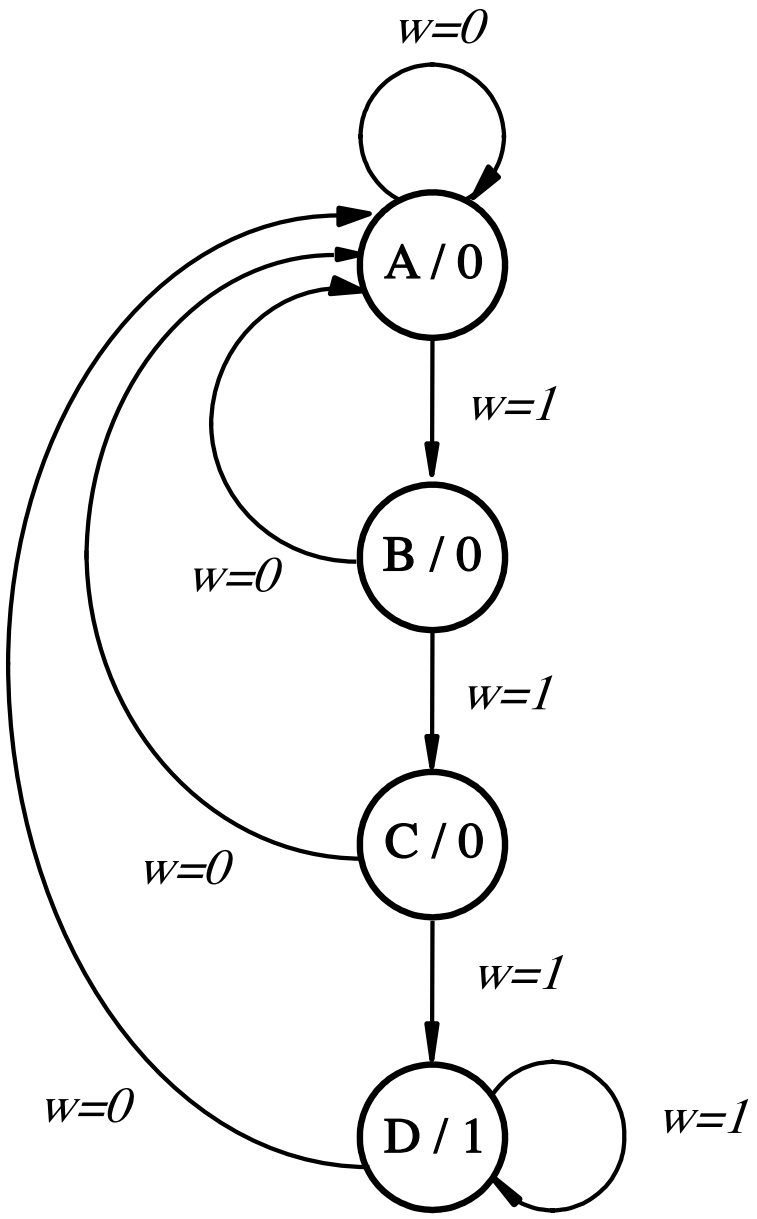
All transitions when the input w is equal to 0



We are done!

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table

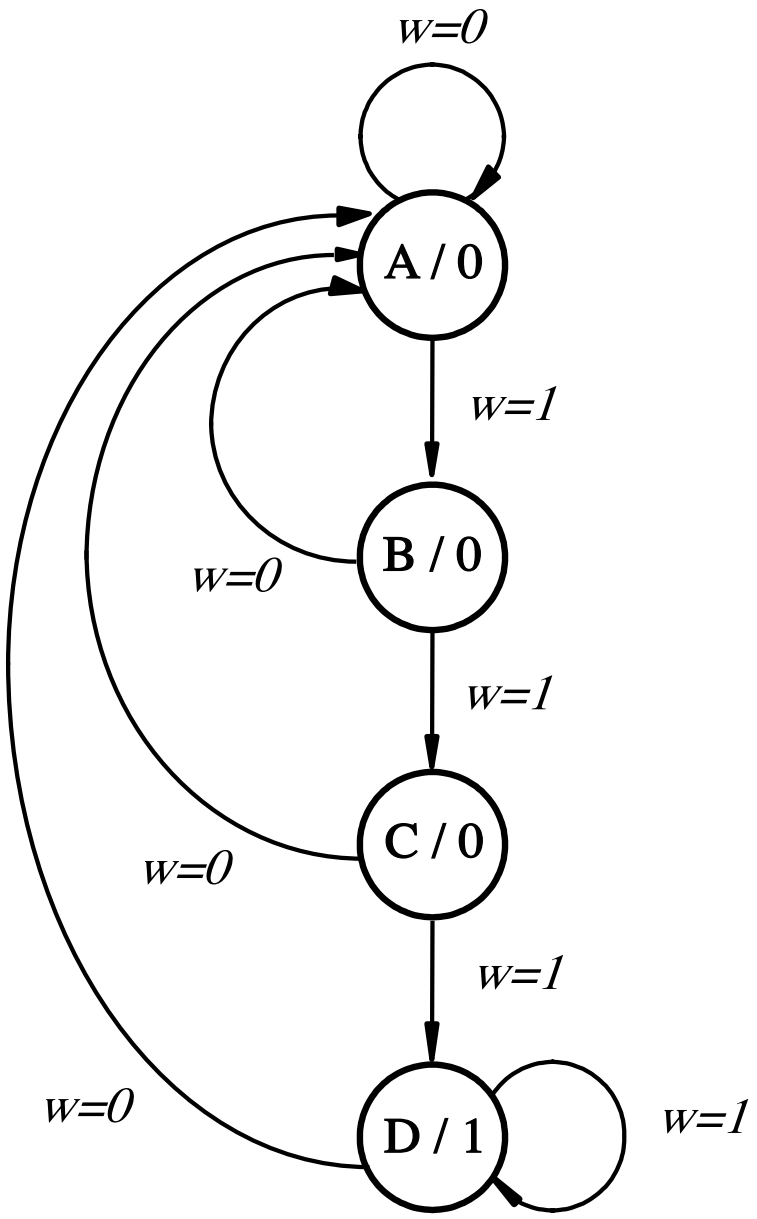


State diagram

Almost done. What does this FSM do?

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table



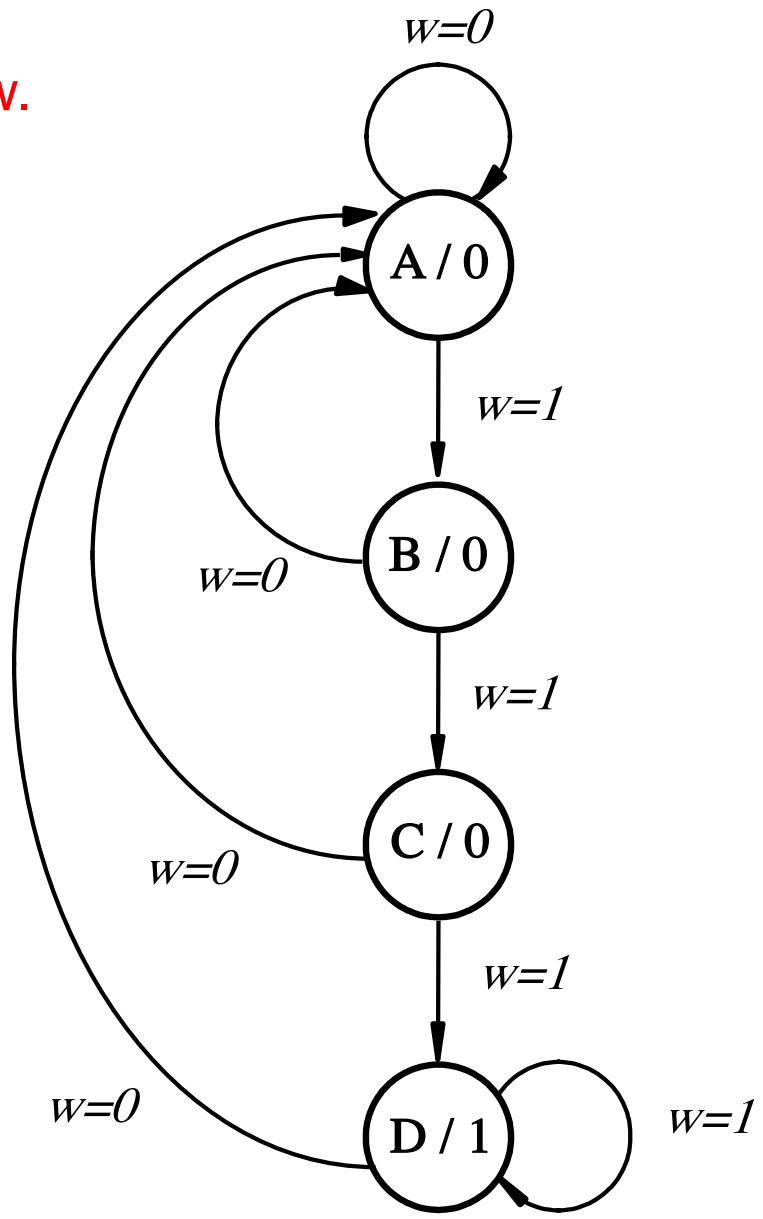
State diagram

Almost done. What does this FSM do?

It sets the output z to 1 when
three consecutive 1's occur on the input w.
In other words, it is a sequence detector
for the input pattern 111.

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

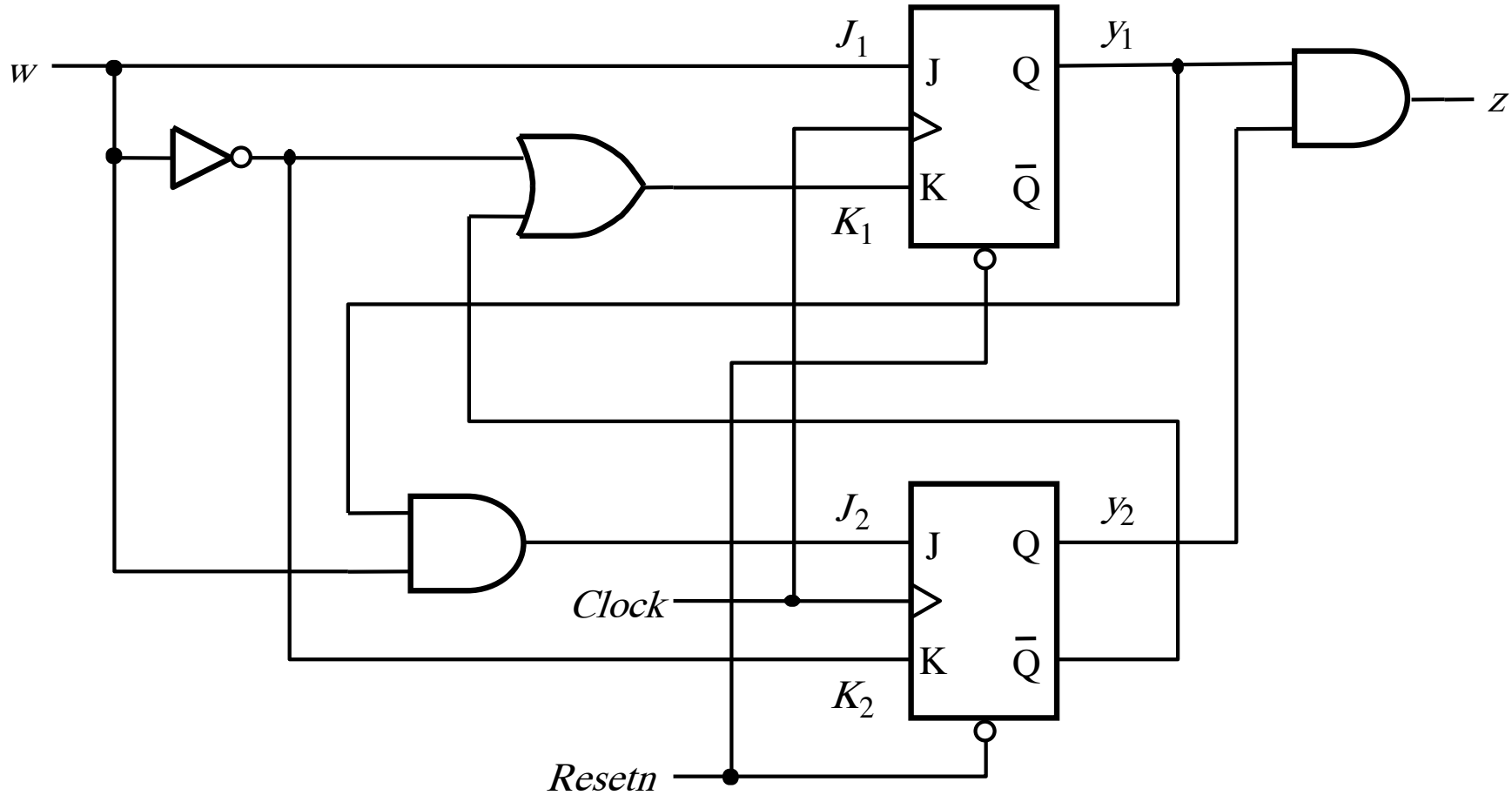
State table



State diagram

Another Example (with JK flip-flops)

What does this circuit do?

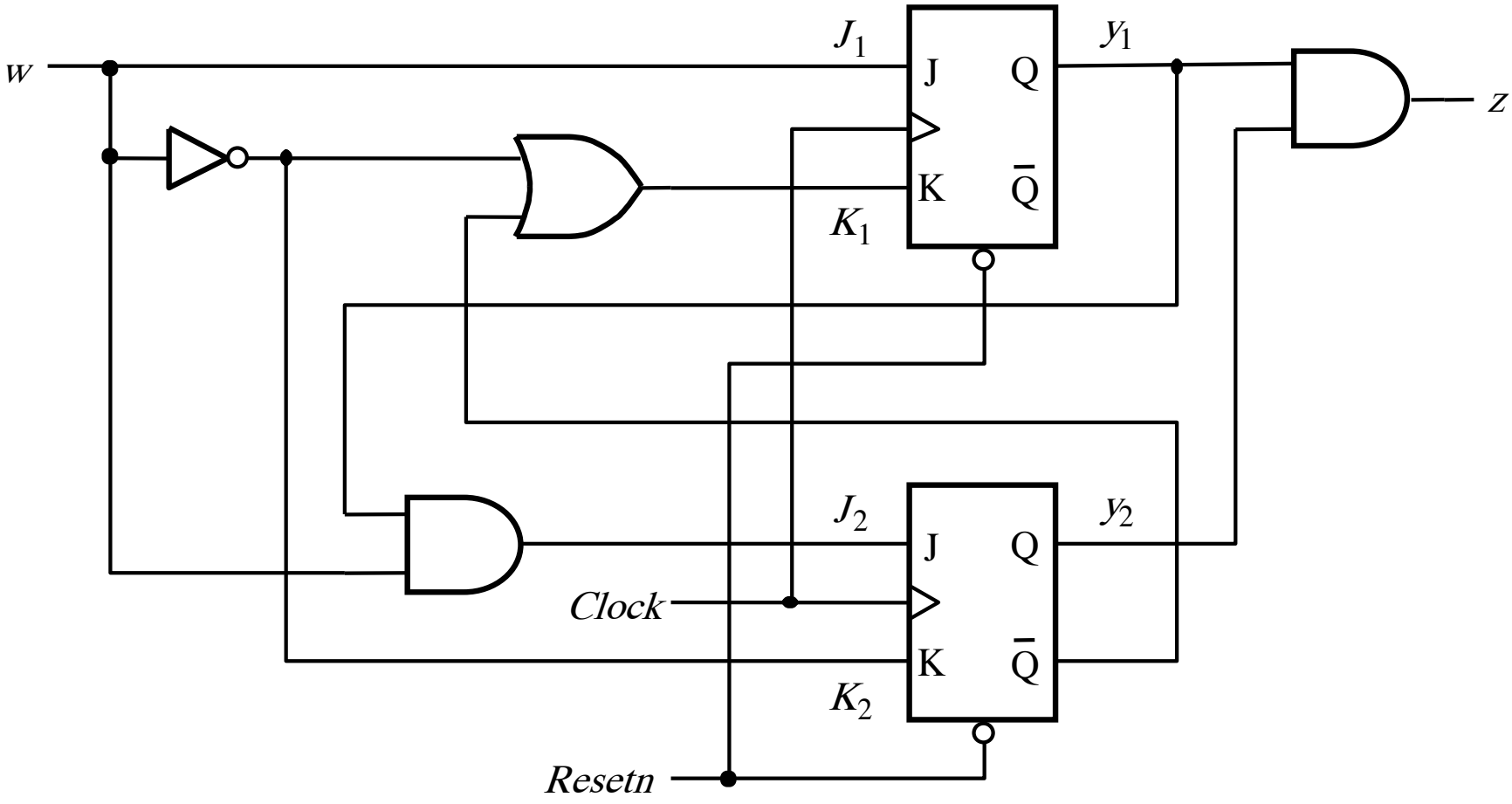


[Figure 6.77 from the textbook]

Approach

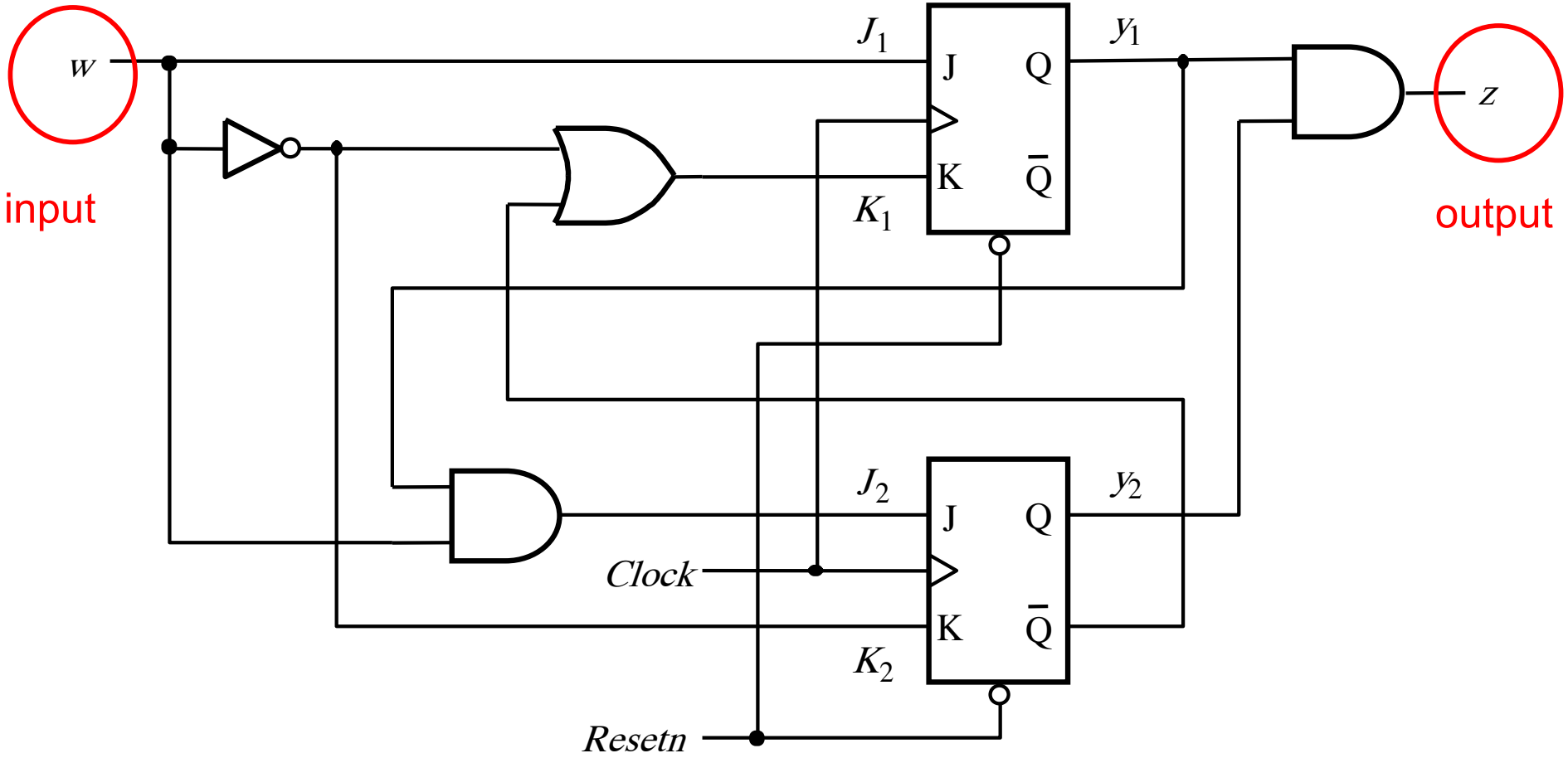
- Find the flip-flops
- Outputs of the flip-flops = present state variables
- Inputs of the flip-flops determine the next state variables
- Determine the logical expressions for the outputs
- Given this info it is easy to do the state-assigned table
- Next do the state table
- Finally, draw the state diagram.

Where are the inputs and outputs?

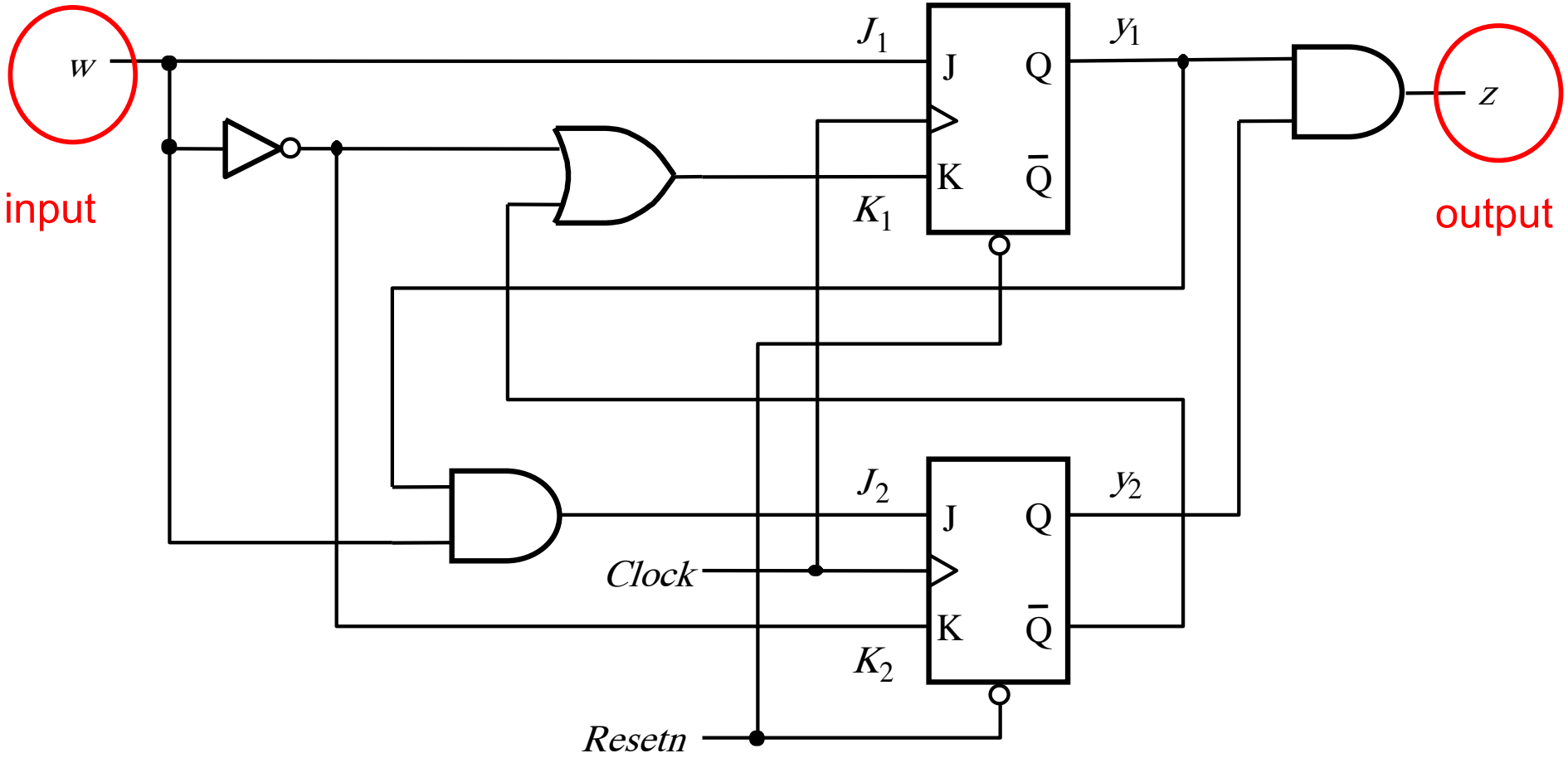


[Figure 6.77 from the textbook]

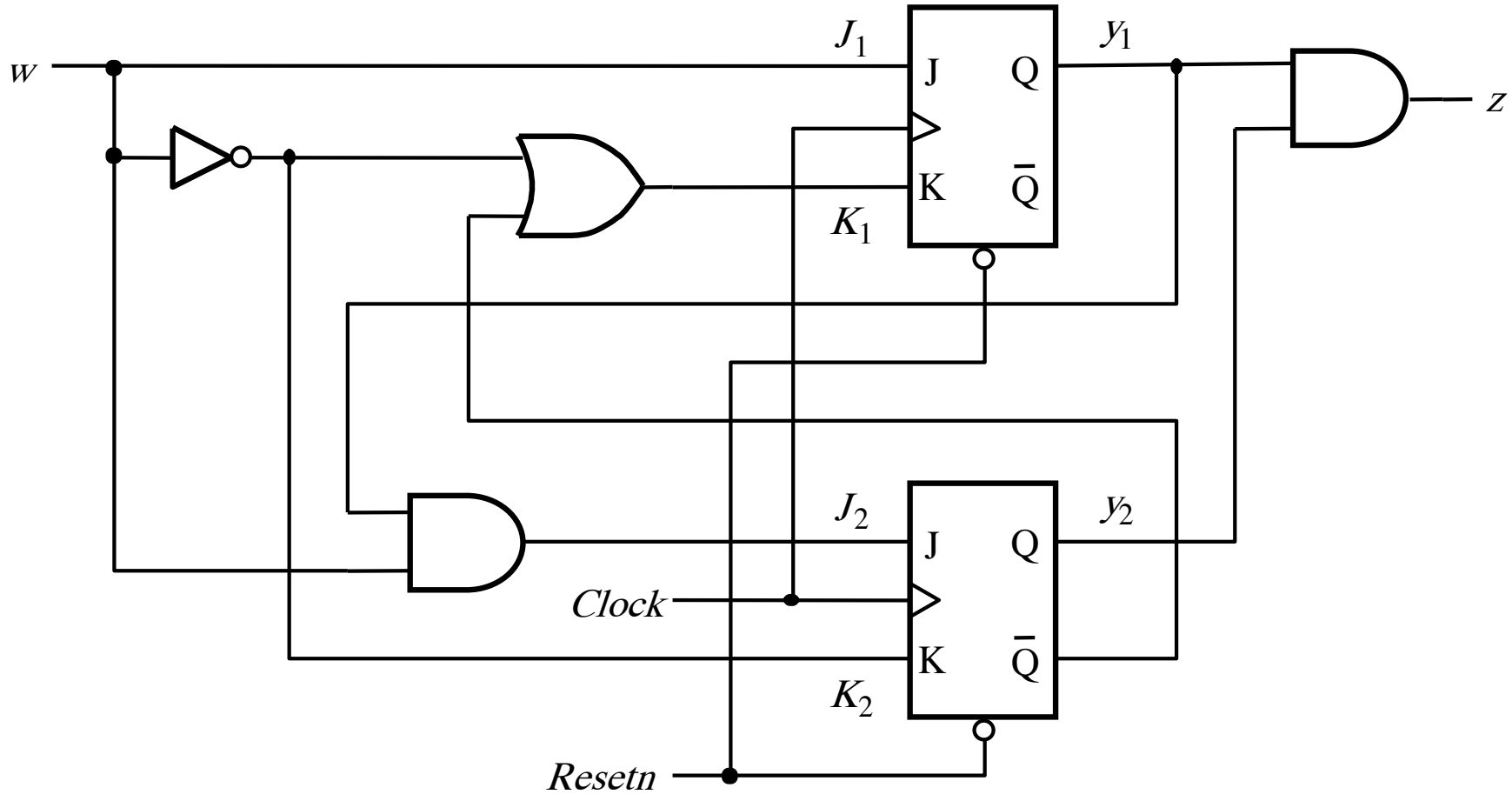
Where are the inputs and outputs?



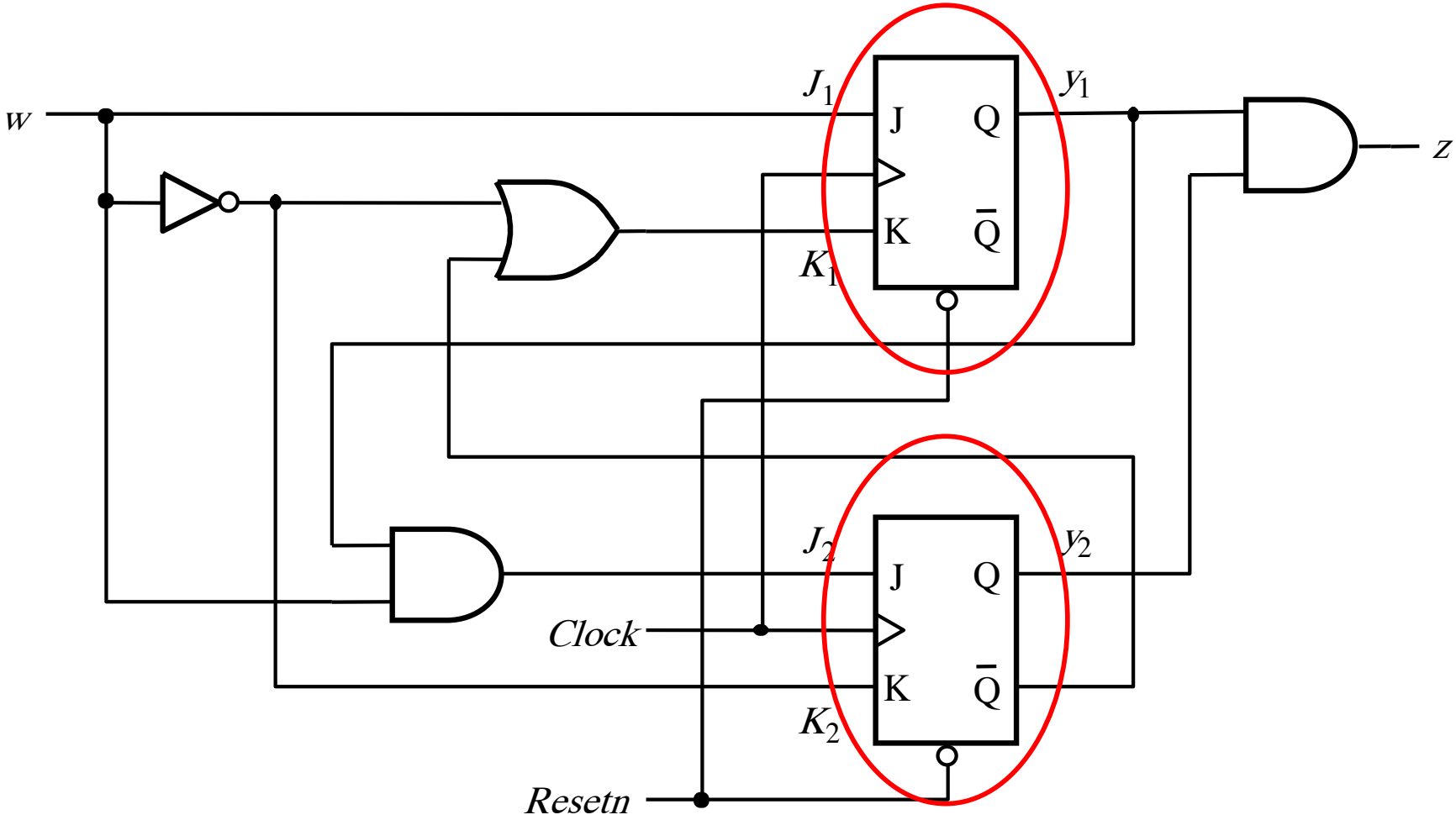
What kind of machine is this?



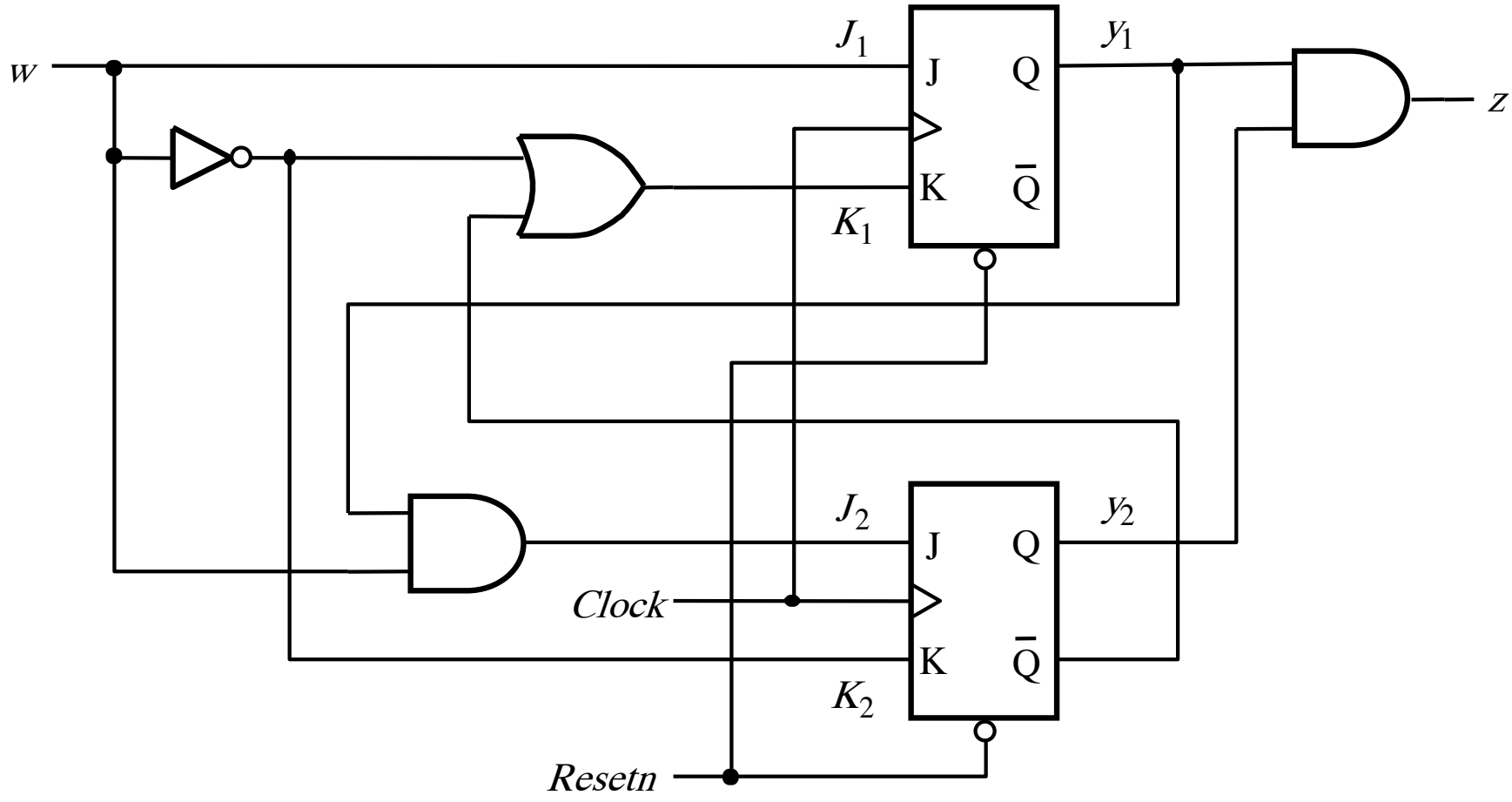
Where are the flip-flops?



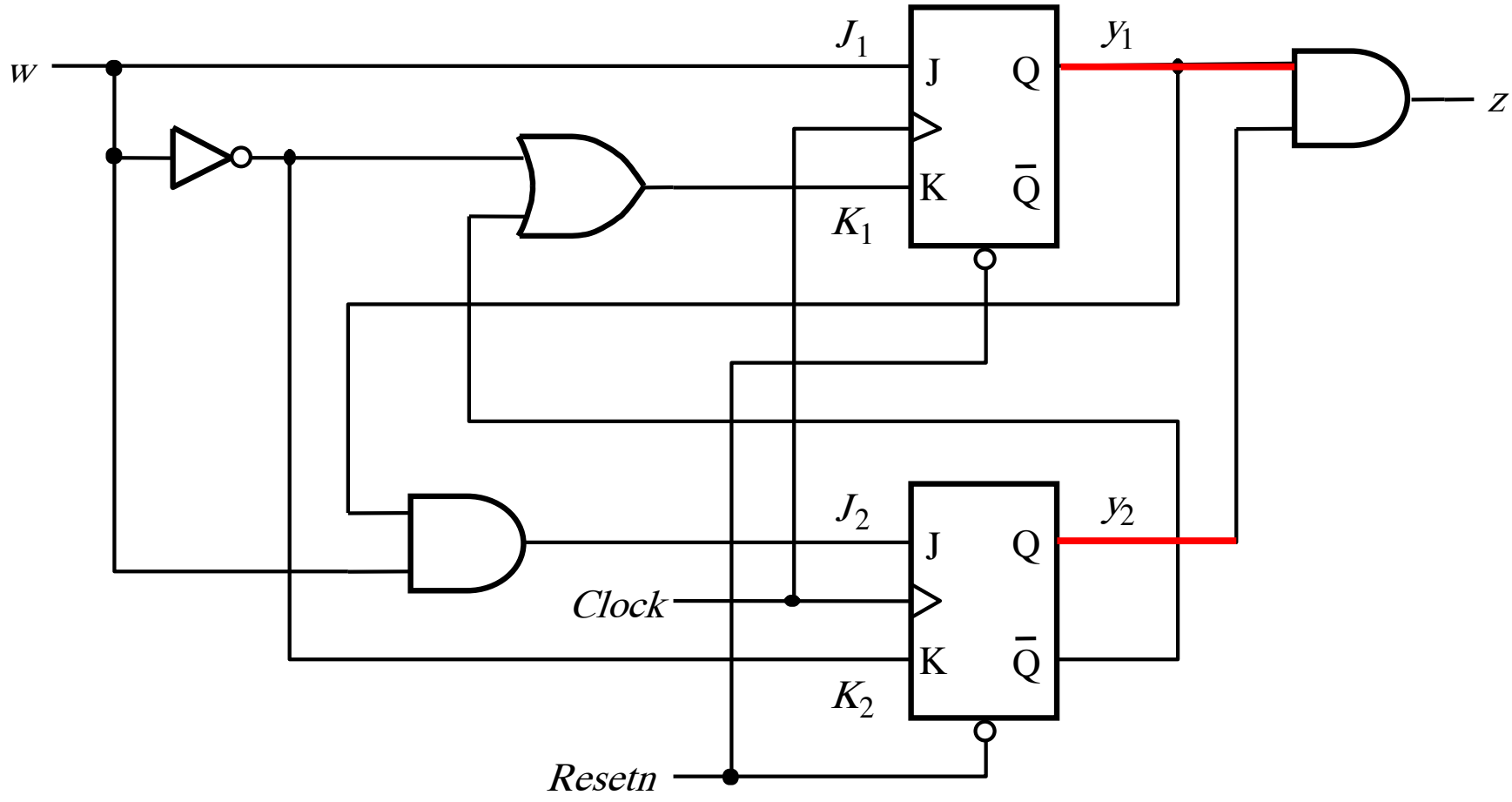
Where are the flip-flops?



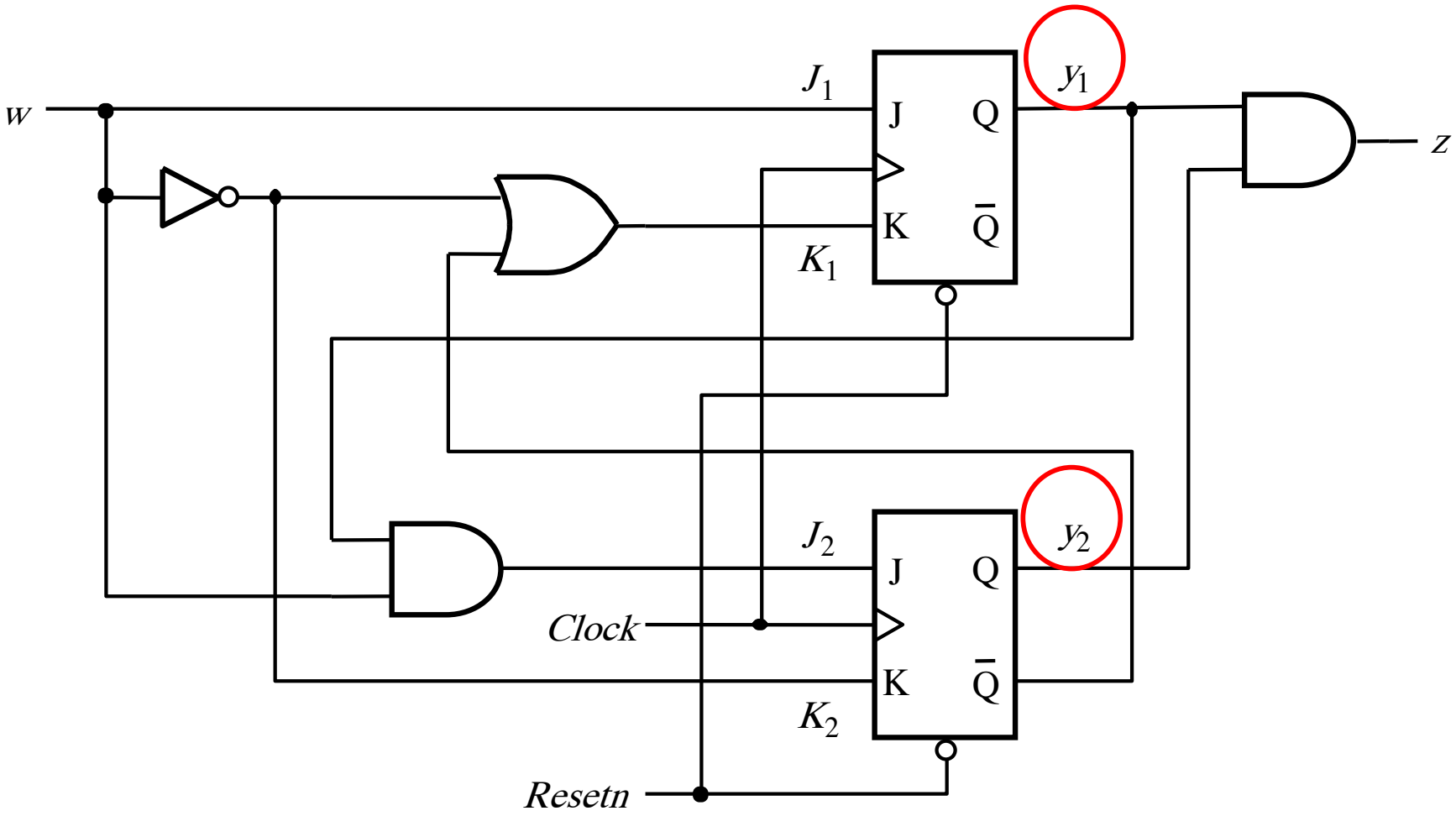
Where are the outputs of the flip-flops?



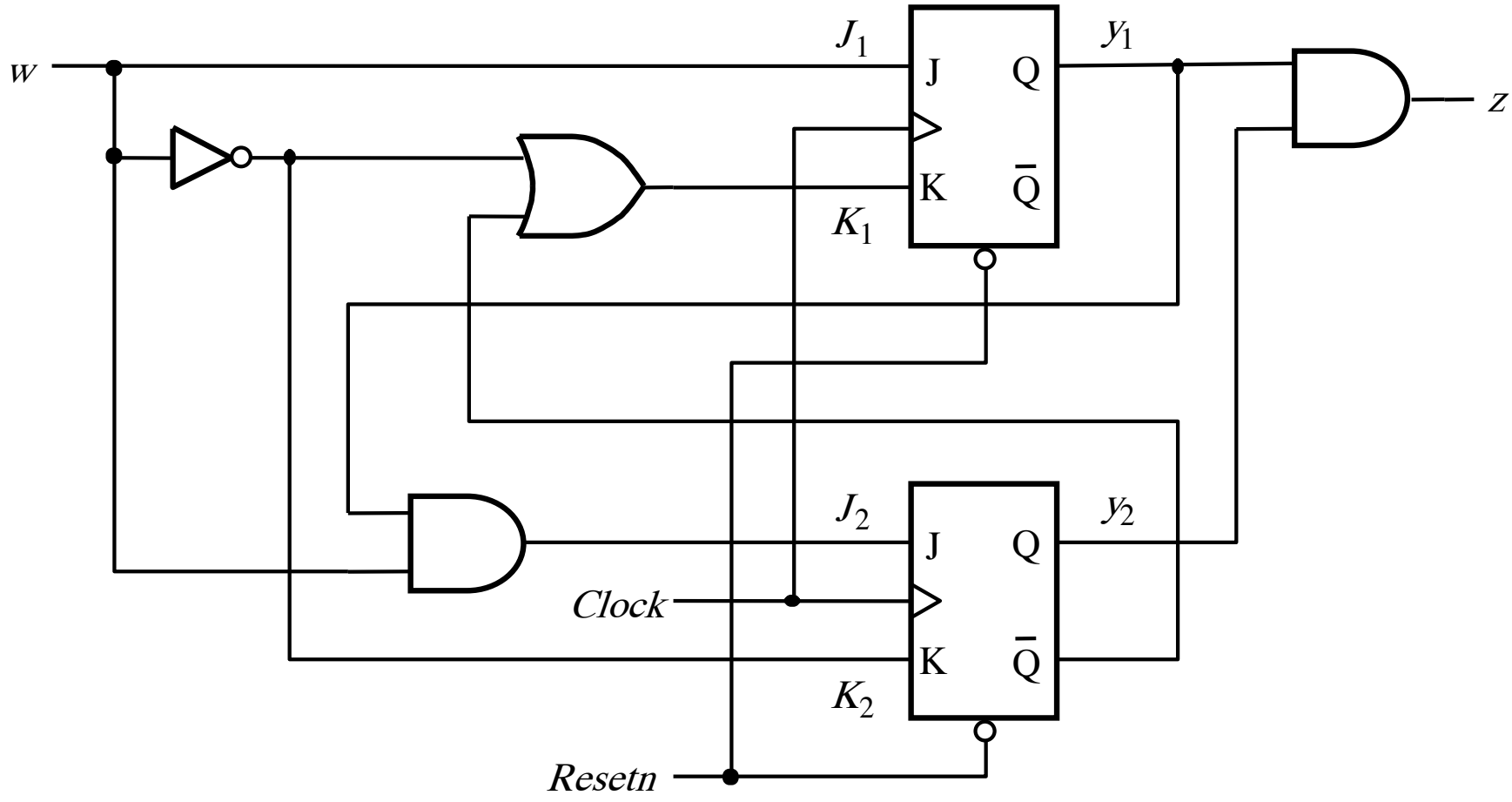
Where are the outputs of the flip-flops?



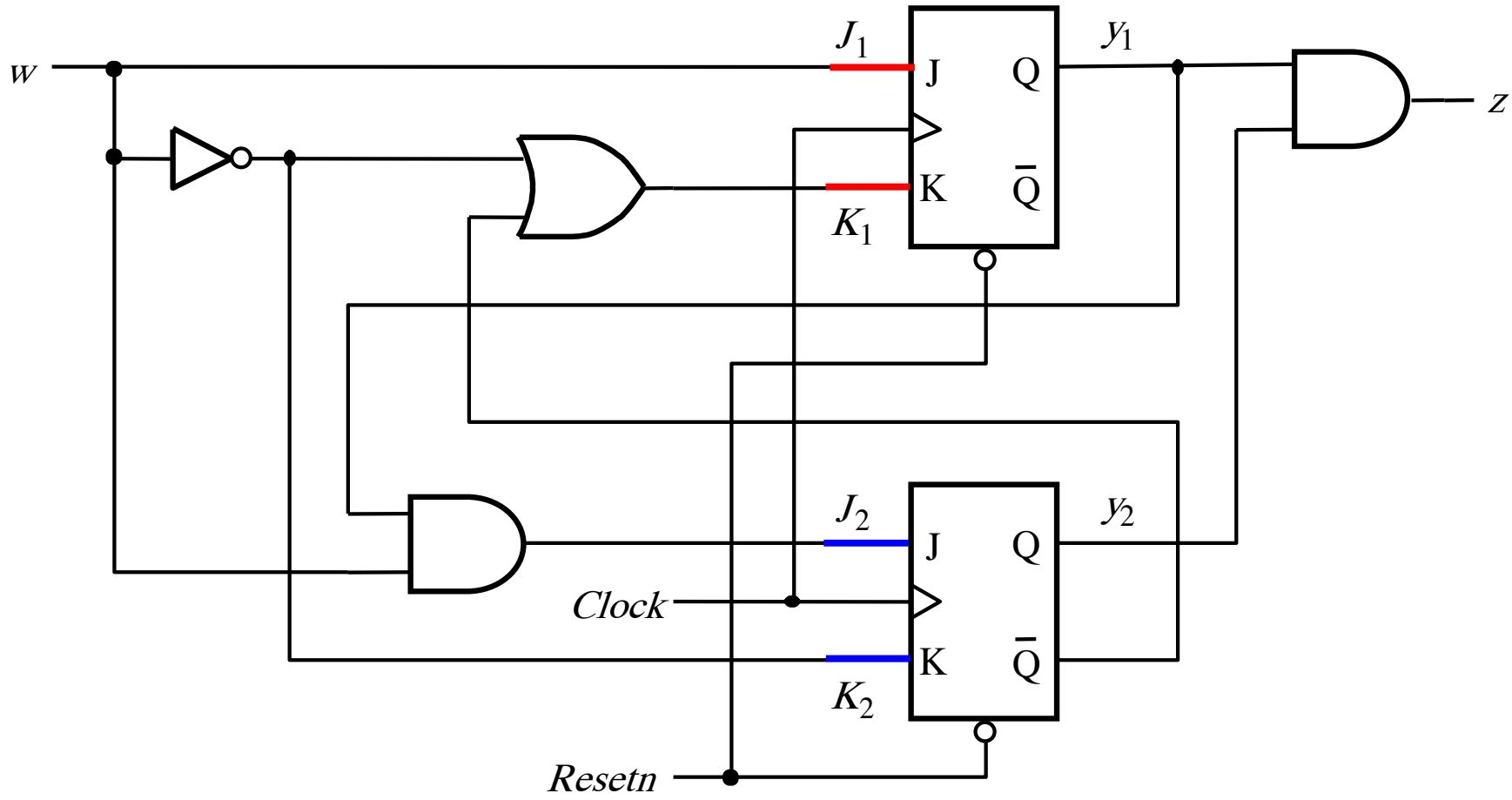
These are the next-state variables



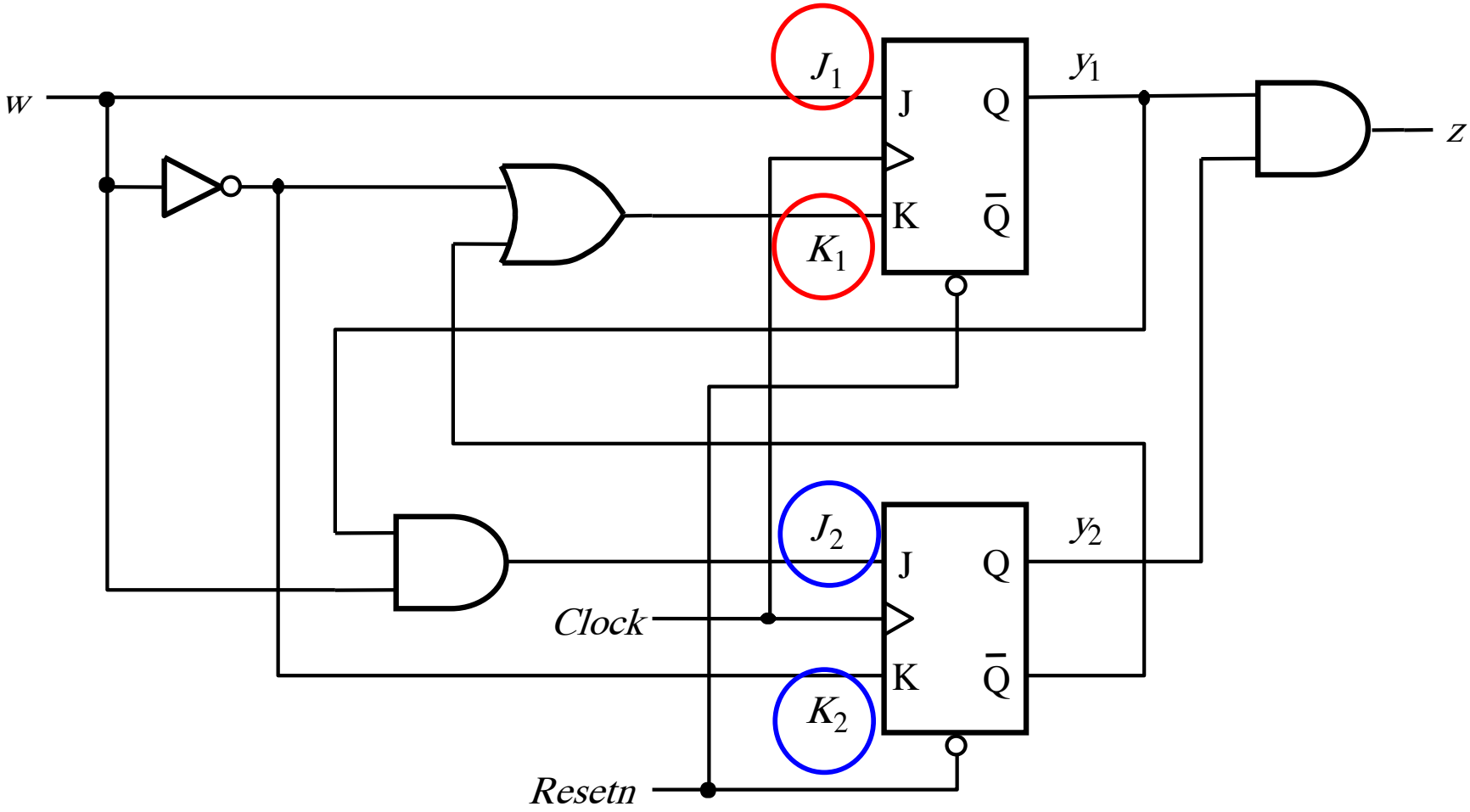
Where are the inputs of the flip-flops?



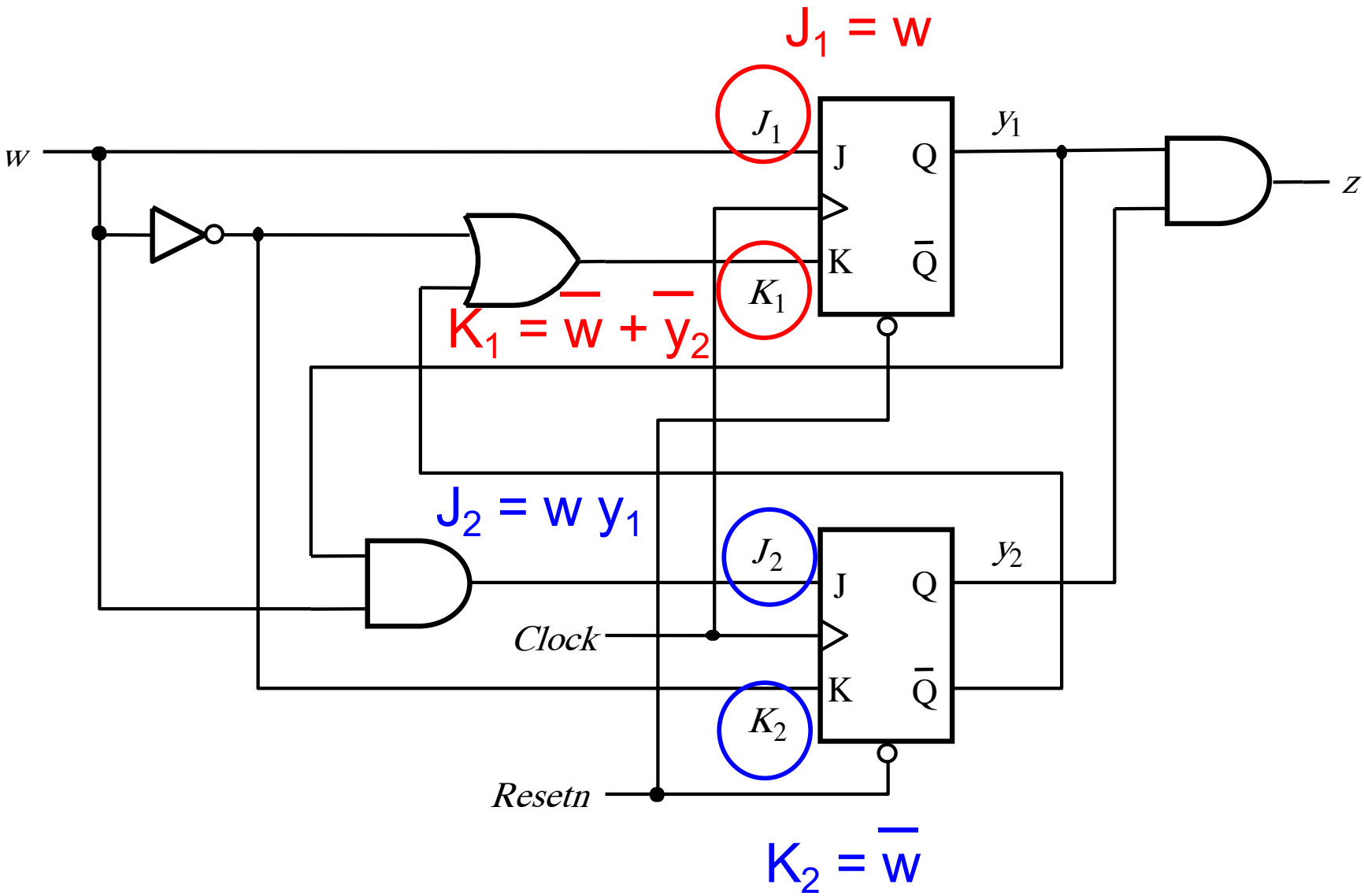
Where are the inputs of the flip-flops?



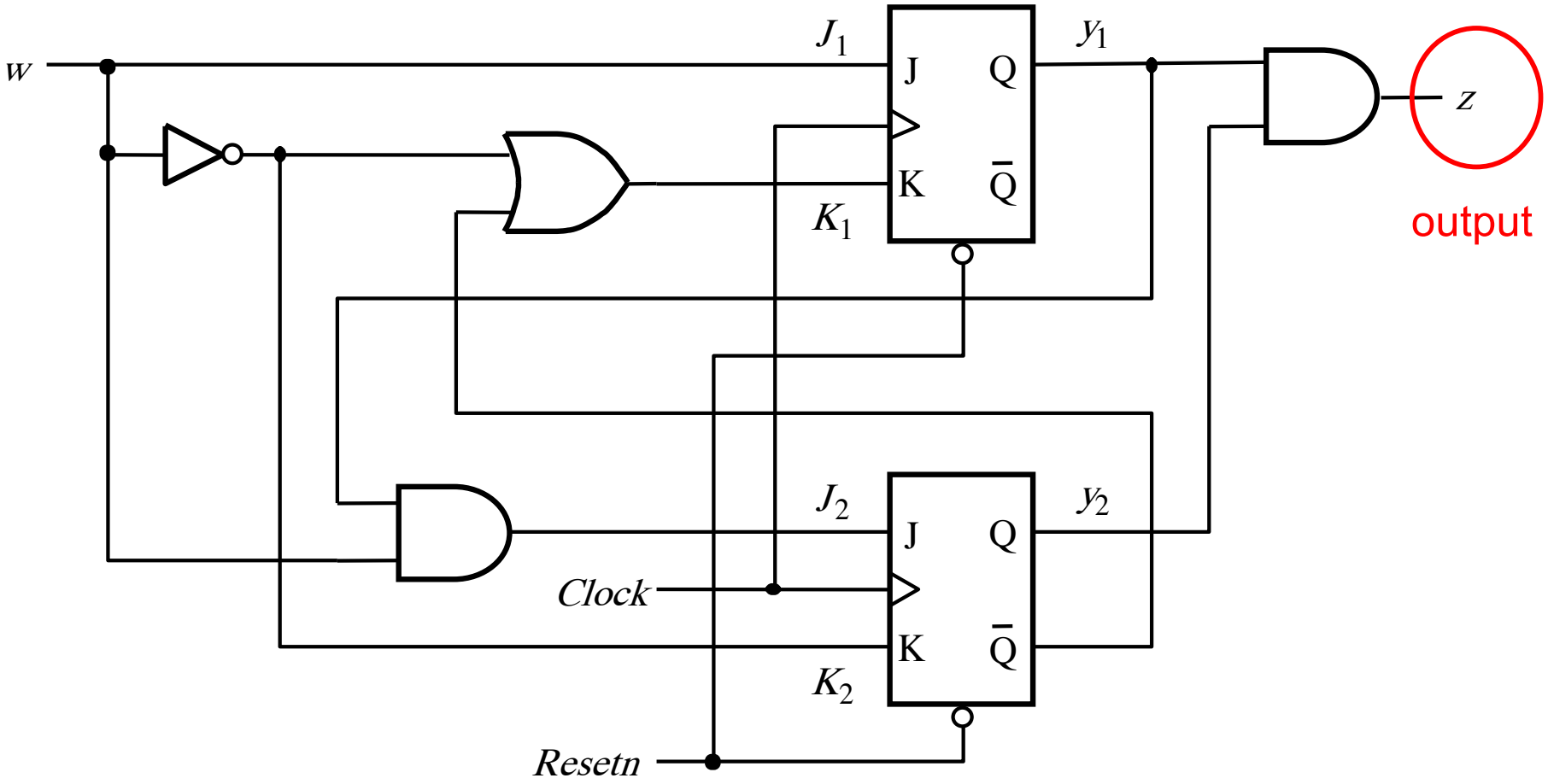
What are their logic expressions?



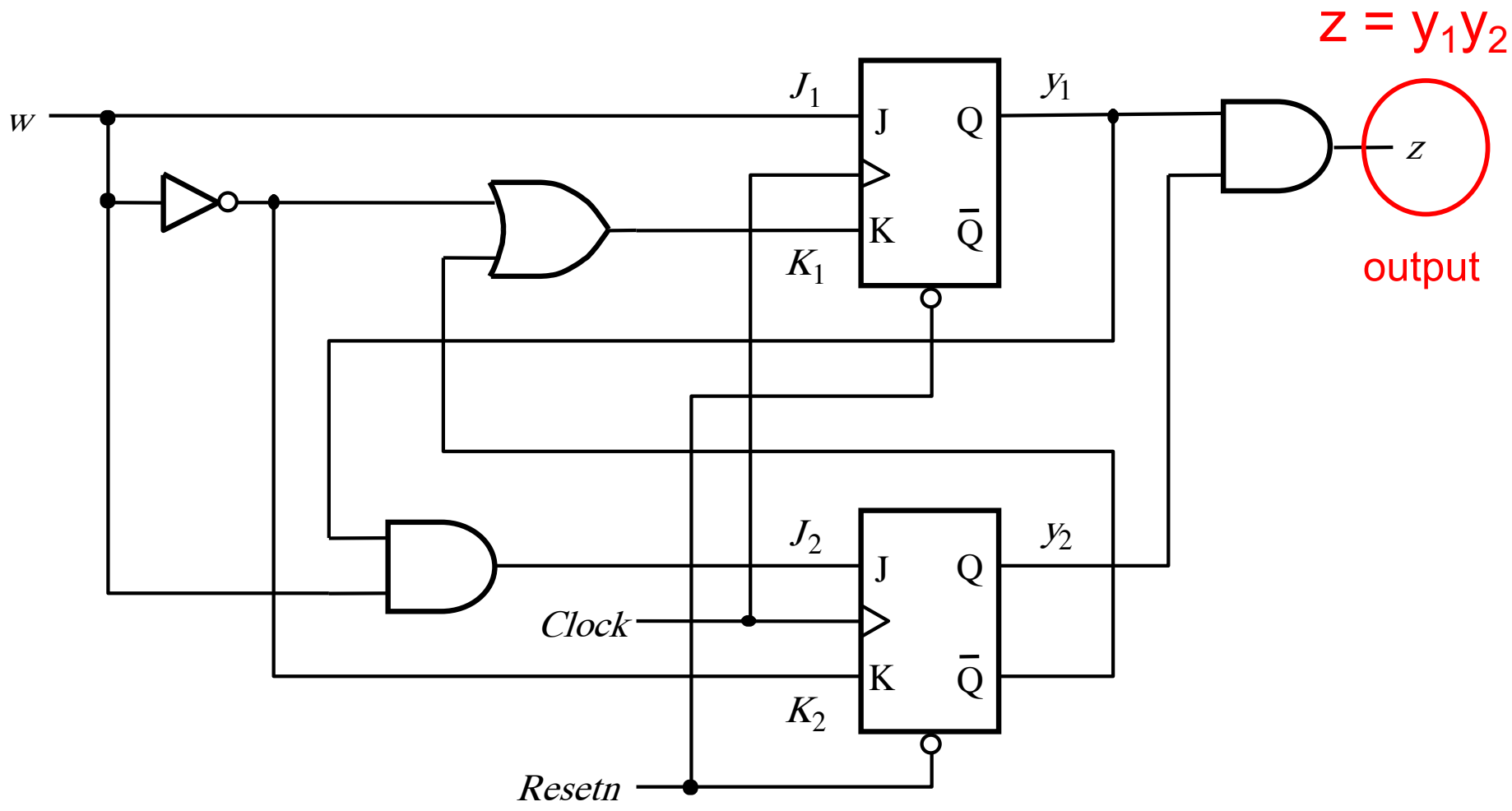
What are their logic expressions?



What is the logic expression of the output?



What is the logic expression of the output?



This is what we have to work with now (we don't need the circuit anymore)

$$J_1 = w$$

$$K_1 = \overline{w} + \overline{y_2}$$

$$J_2 = w\, y_1$$

$$K_2 = \overline{w}$$

$$z = y_1 y_2$$

Let's derive the excitation table

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

$z = y_1 y_2$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00					
01					
10					
11					

Let's derive the excitation table

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00					
01					
10					
11					

$z = y_1y_2$

Let's derive the excitation table

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00					0
01					0
10					0
11					1

$z = y_1y_2$

Let's derive the excitation table

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

$z = y_1 y_2$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00					0
01					0
10					0
11					1

Let's derive the excitation table

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

$z = y_1 y_2$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	0	1	1	1	0
01	0	1	1	1	0
10	0	1	1	0	0
11	0	1	1	0	1

Let's derive the excitation table

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

$z = y_1 y_2$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	0	1	1	1	0
01	0	1	1	1	0
10	0	1	1	0	0
11	0	1	1	0	1

The excitation table

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

$z = y_1 y_2$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

[Figure 6.78 from the textbook]

We don't need the logic expressions anymore

$J_1 = w$

$K_1 = \overline{w} + \overline{y_2}$

$J_2 = w\, y_1$

$K_2 = \overline{w}$

$z = y_1 y_2$

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

[Figure 6.78 from the textbook]

We don’t need the logic expressions anymore

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	01	0 1	0 0	1 1	0
01	01	0 1	1 0	1 1	0
10	01	0 1	0 0	1 0	0
11	01	0 1	1 0	1 0	1

[Figure 6.78 from the textbook]

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	

State table

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

Excitation table

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			
B			
C			
D			

State table

Present state	Flip-flop inputs				Output
	w = 0		w = 1		
y_2y_1	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

Excitation table

This step is easy
(map 2-bit numbers to 4 letters)

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

State table

Present state	Flip-flop inputs				Output
	w = 0		w = 1		
y_2y_1	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

Excitation table

This step is easy too
(the outputs are the same in both tables)

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	?		0
B			0
C			0
D			1

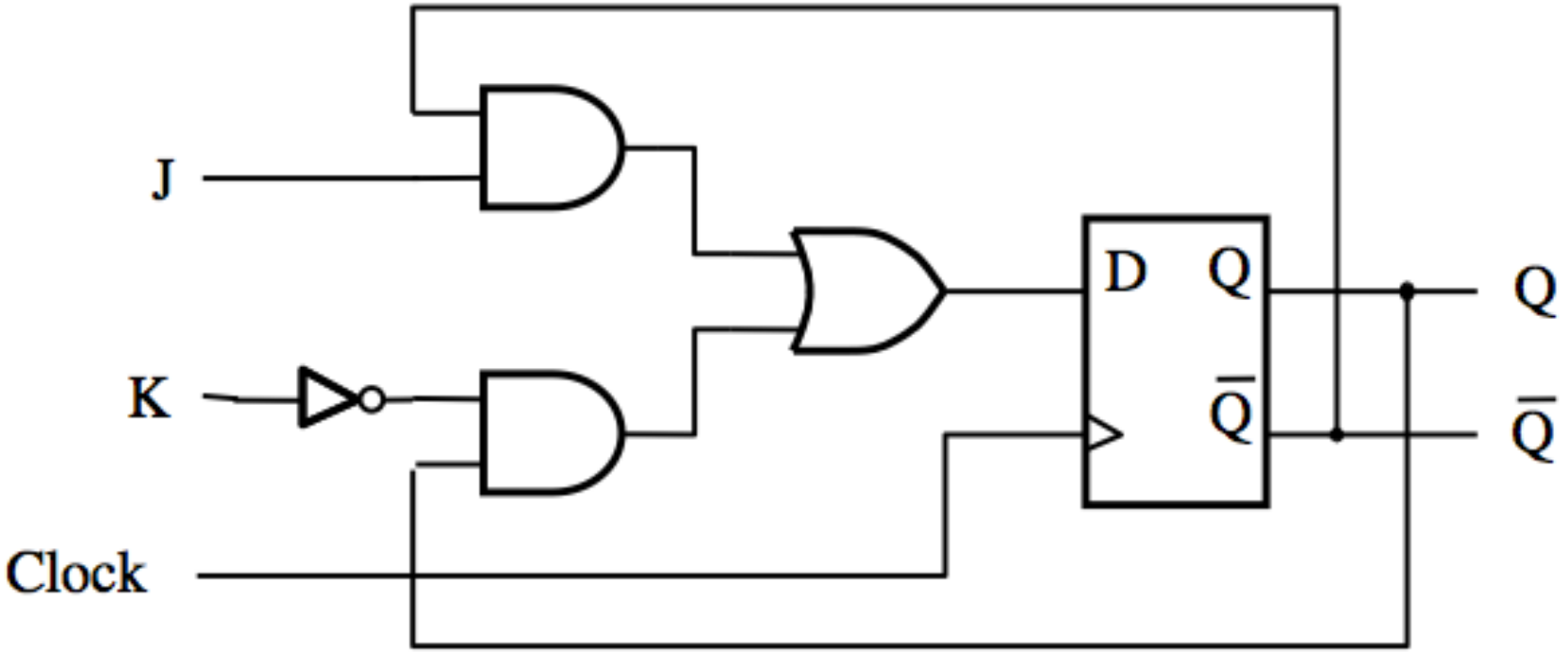
State table

Present state	Flip-flop inputs				Output
	w = 0		w = 1		
y_2y_1	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

Excitation table

How should we do this?

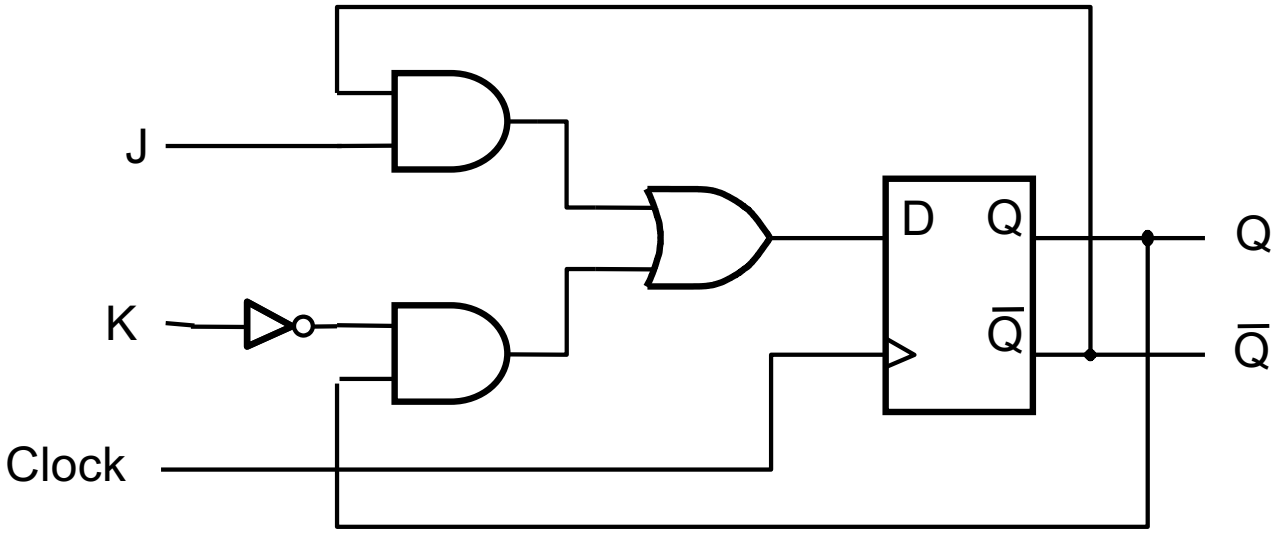
JK Flip-Flop Refresher



$$D = J\overline{Q} + \overline{K}Q$$

[Figure 5.16a from the textbook]

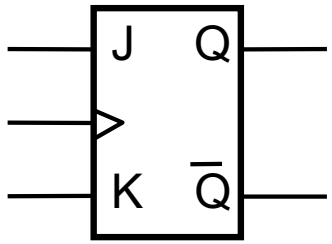
JK Flip-Flop Refresher



(a) Circuit

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Truth table



(c) Graphical symbol

[Figure 5.16 from the textbook]

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	?		0
B			0
C			0
D			1

State table

Present state	Flip-flop inputs				Output
	w = 0		w = 1		
y_2y_1	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

Excitation table

How should we do this?

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs				Output
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\overline{Q}$(t)

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs				Output
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\overline{Q}$(t)

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\overline{Q}$(t)

Let's derive the state table

Present state	Next state w = 0	Next state w = 1	Output z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs $w = 0$ J_2K_2	$w = 0$ J_1K_1	$w = 1$ J_2K_2	$w = 1$ J_1K_1	Output z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

Note that A = 00

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}(t)$

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}(t)$

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	
A	A	?	0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\overline{Q}$(t)

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\overline{Q}$(t)

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs				Output z
	w = 0		w = 1		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}$(t)

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}$(t)

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs				Output
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}(t)$

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}(t)$

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs				Output
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

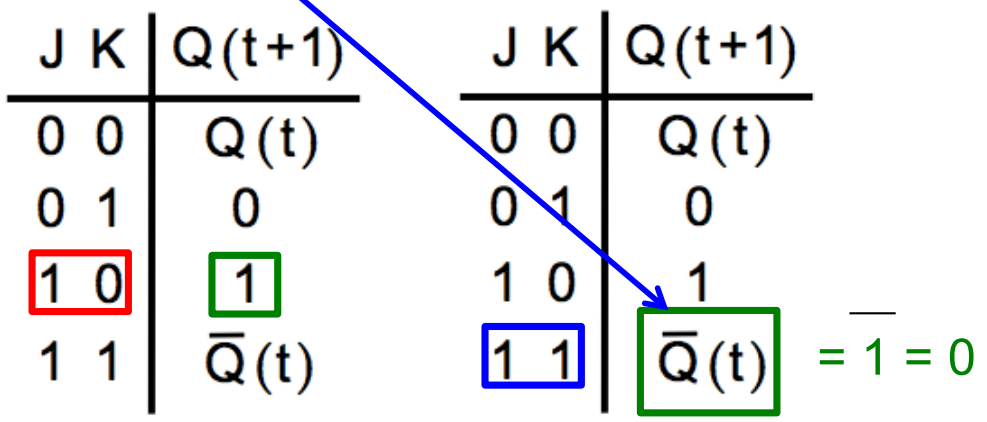
J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}(t)$

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\bar{Q}(t)$

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs				Output
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1



Let's derive the state table

Present state	Next state w = 0	Next state w = 1	Output z
A	A		0
B		C	0
C			0
D			1

Present state y_2y_1	Flip-flop inputs $w = 0$ J_2K_2	$w = 0$ J_1K_1	$w = 1$ J_2K_2	$w = 1$ J_1K_1	Output z
00	01	01	00	11	0
01	01	01	10	11	0
10	01	01	00	10	0
11	01	01	10	10	1

Note that C = 10

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\overline{Q}(t)$

J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	$\overline{Q}(t)$

=0

The two tables for the initial circuit

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table

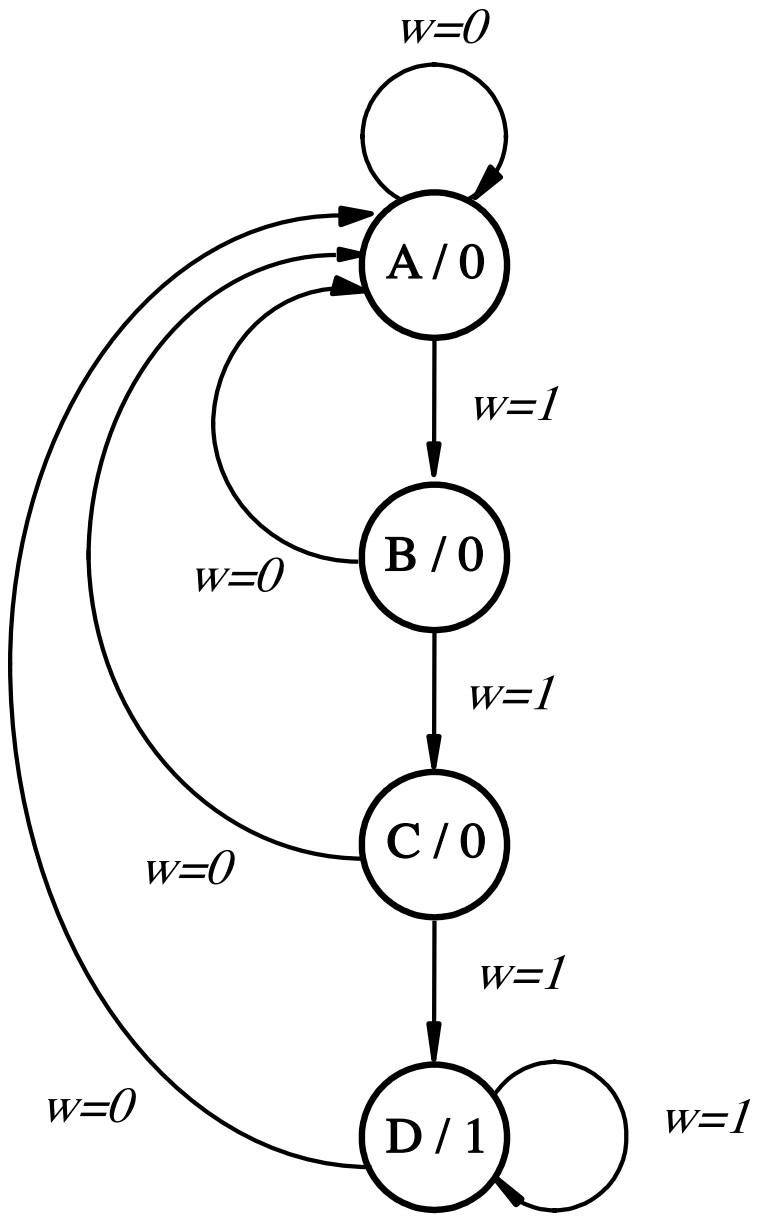
Present state y_2y_1	Flip-flop inputs				Output z
	$w = 0$		$w = 1$		
	J_2K_2	J_1K_1	J_2K_2	J_1K_1	
00	01	0 1	0 0	1 1	0
01	01	0 1	1 0	1 1	0
10	01	0 1	0 0	1 0	0
11	01	0 1	1 0	1 0	1

Excitation table

The state diagram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table



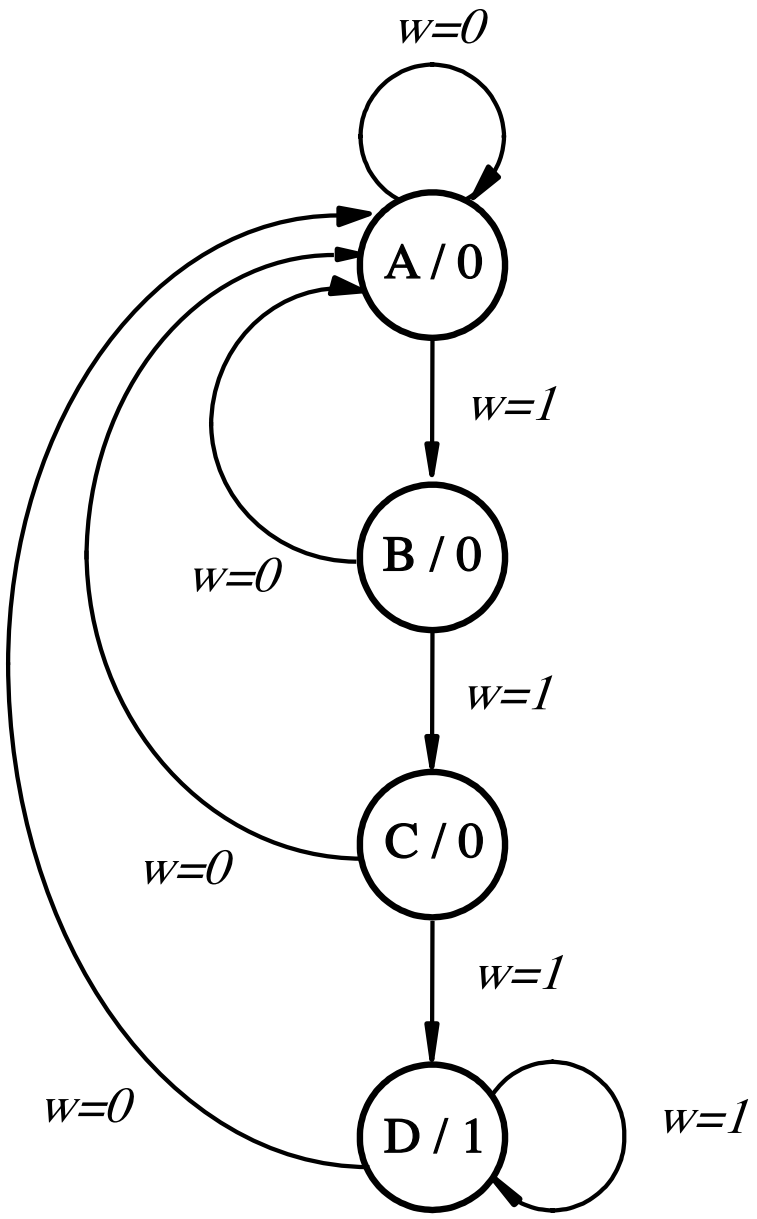
State diagram

The state diagram

Thus, this FSM is identical to the one in the previous example, even though the circuit uses JK flip-flops.

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

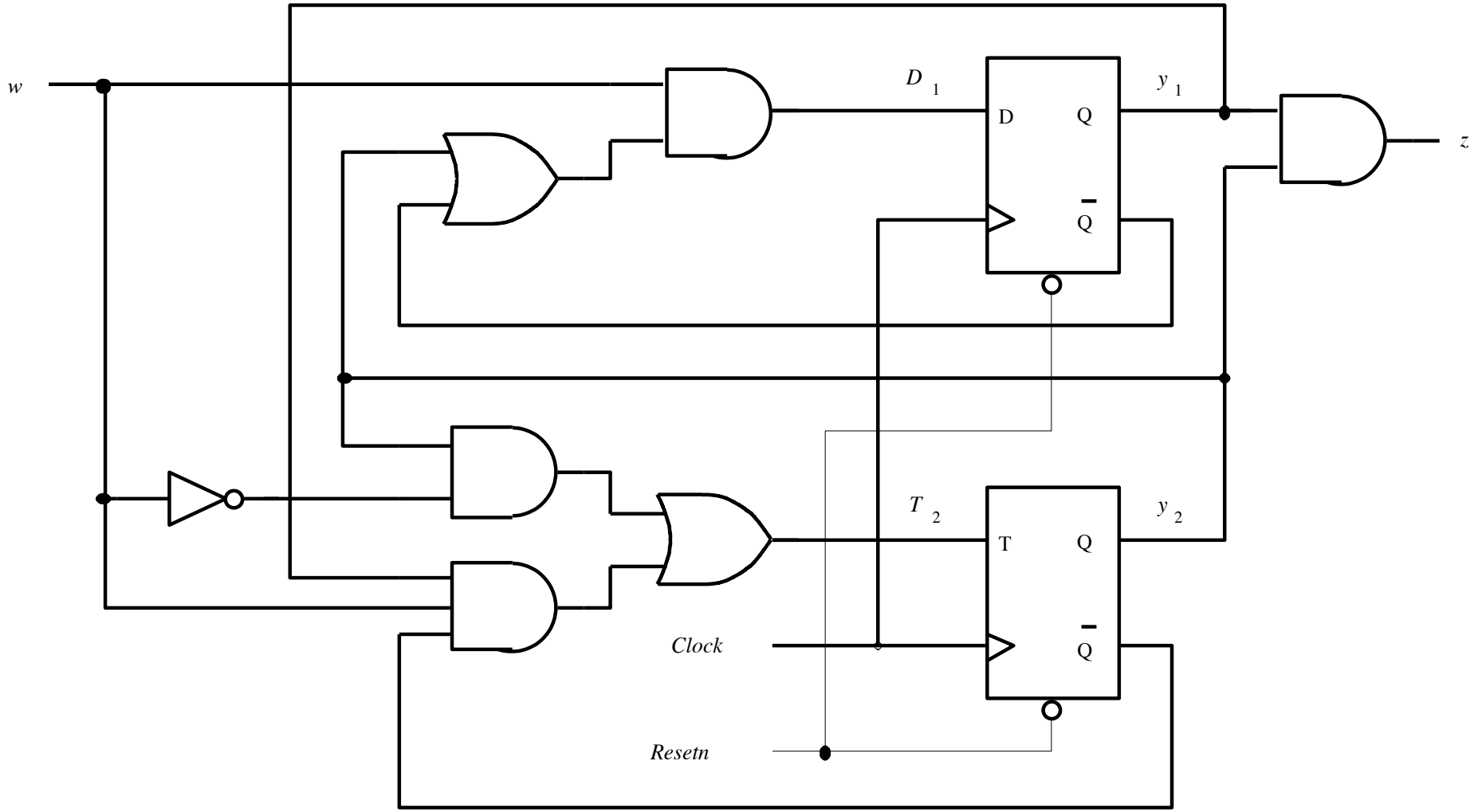
State table



State diagram

Yet Another Example
(with mixed flip-flops)

What does this circuit do?

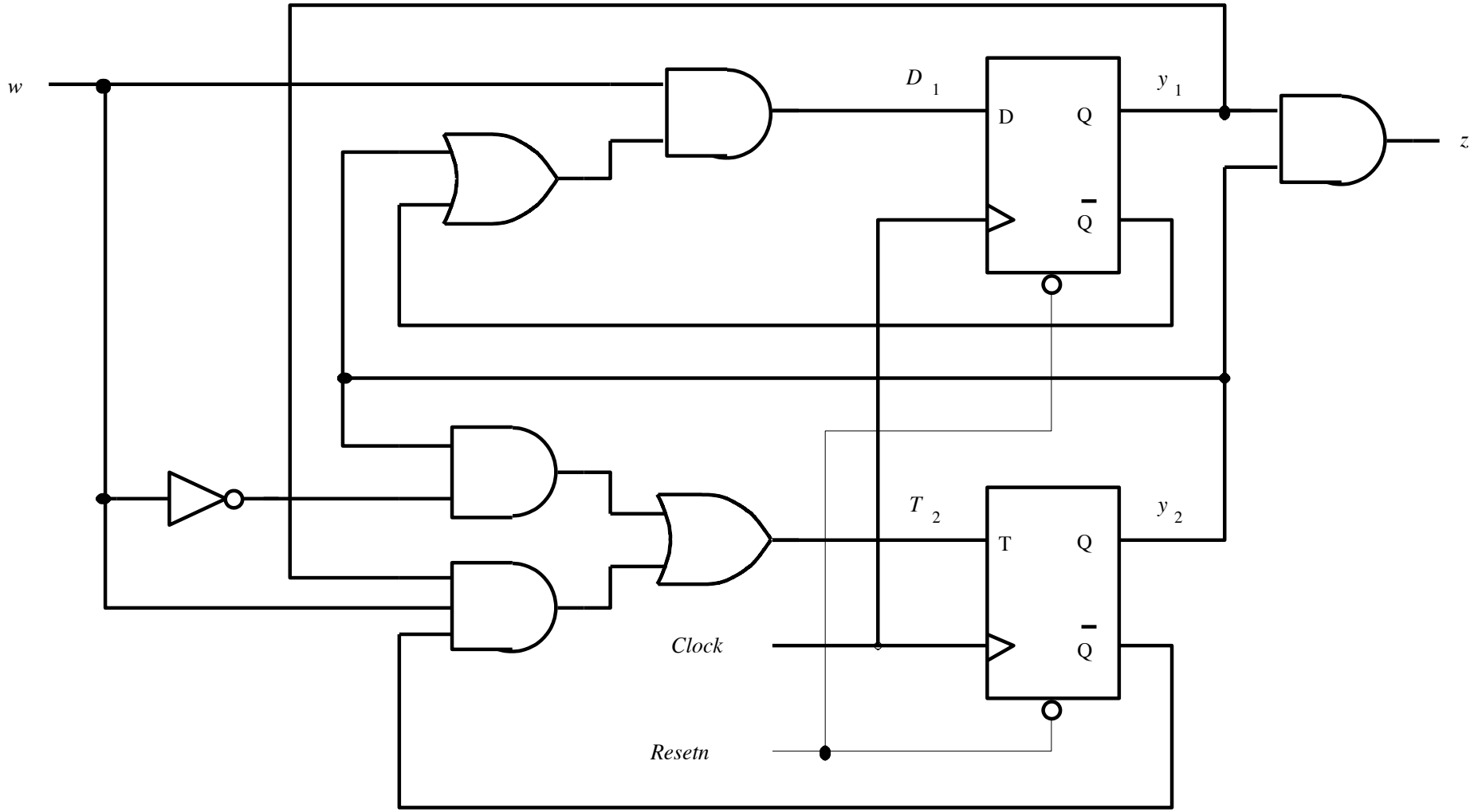


[Figure 6.79 from the textbook]

Approach

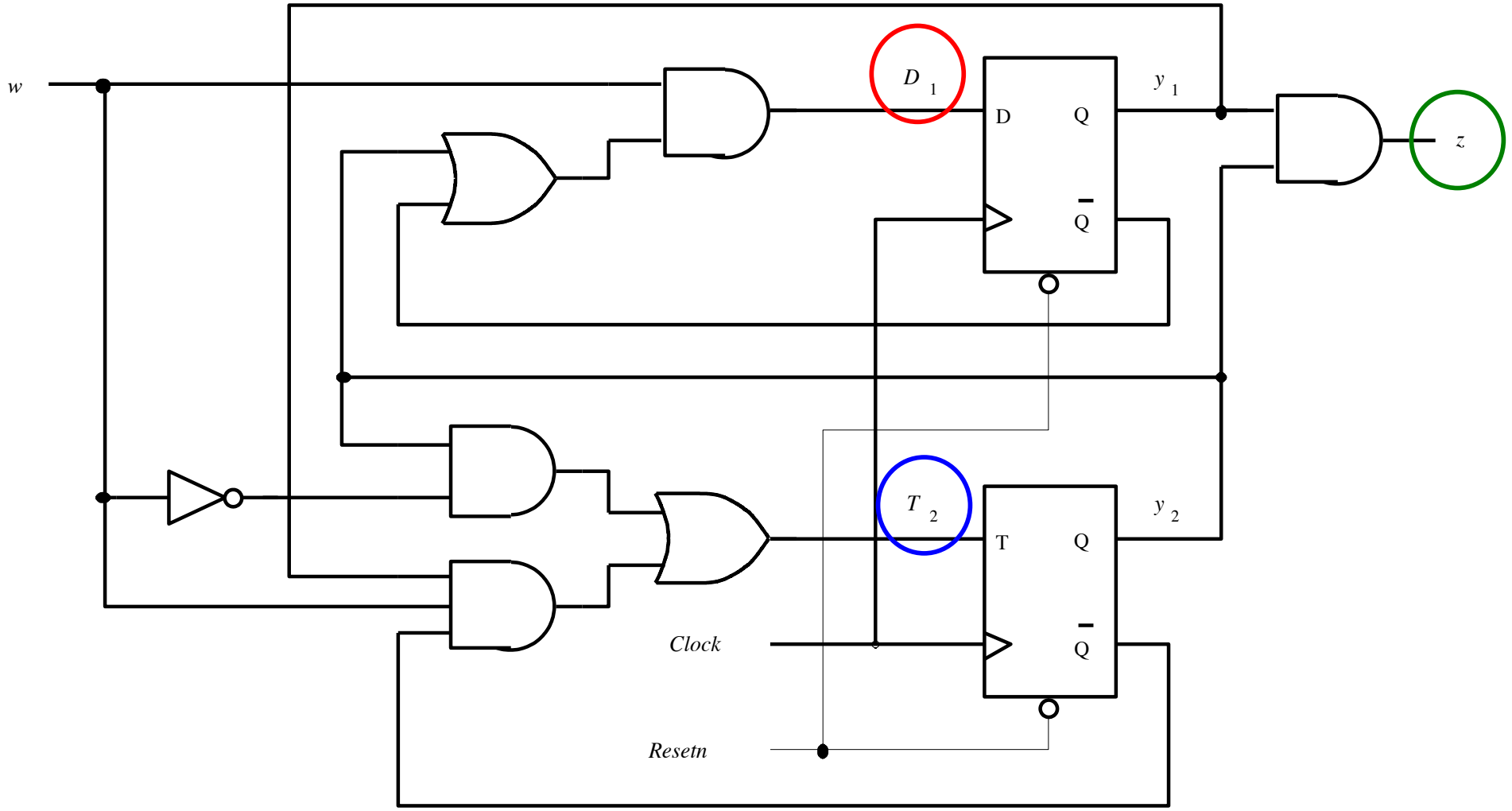
- Find the flip-flops
- Outputs of the flip-flops = present state variables
- Inputs of the flip-flops determine the next state variables
- Determine the logical expressions for the outputs
- Given this info it is easy to do the state-assigned table
- Next do the state table
- Finally, draw the state diagram.

What are the logic expressions?



[Figure 6.79 from the textbook]

What are the logic expressions?

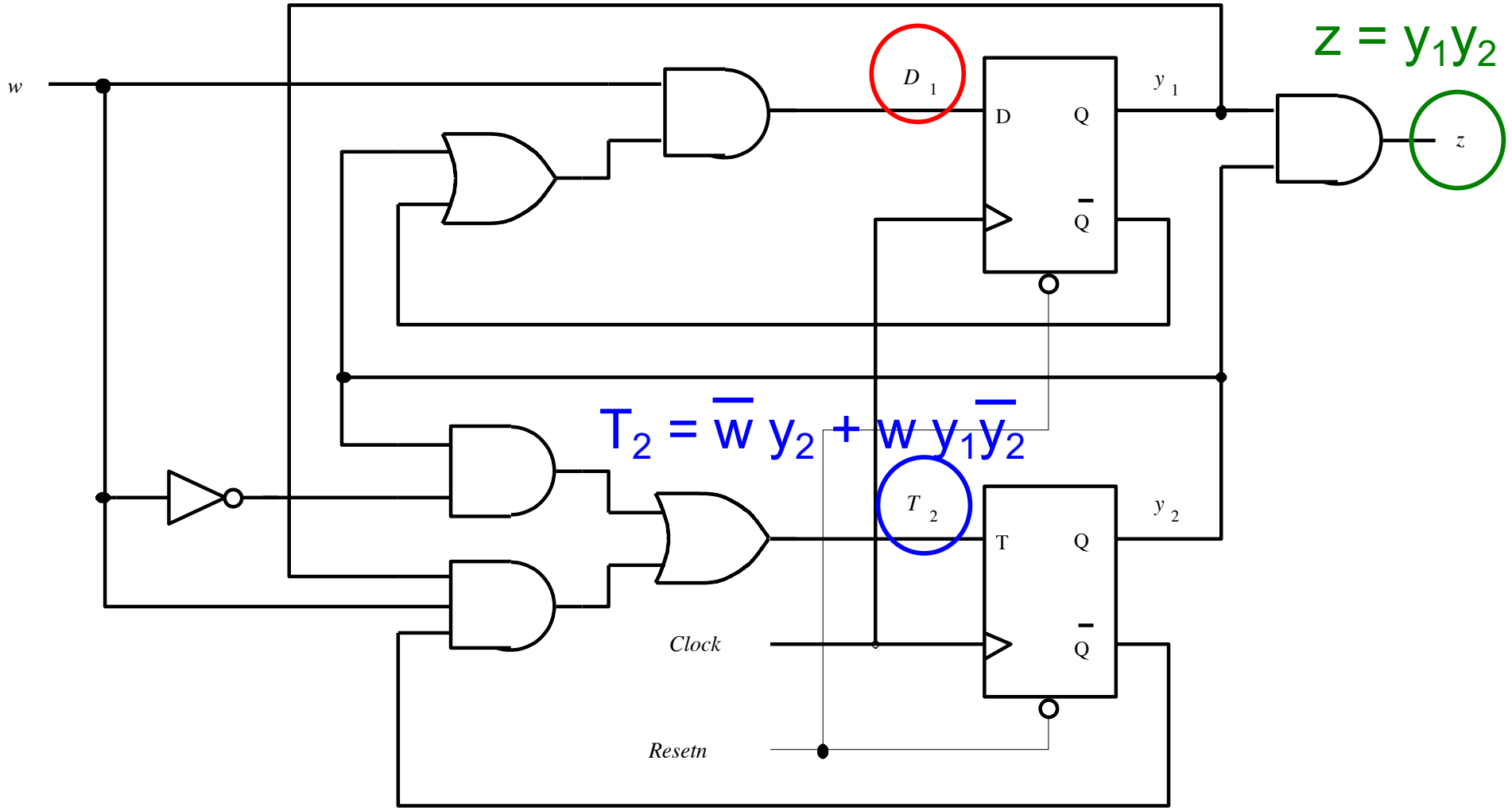


What are the logic expressions?

$D_1 = w(\overline{y_1} + y_2)$

$z = y_1 y_2$

$T_2 = \overline{w}\, y_2 + w\, y_1 \overline{y_2}$



The Excitation Table

$D_1 = w(\overline{y_1} + y_2)$

$T_2 = \overline{w}\, y_2 + w\, y_1\overline{y_2}$

$z = y_1 y_2$

Present state y_2y_1	Flip-flop inputs		Output z
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	1 0	0 1	0
1 1	1 0	0 1	1

Excitation table

Let's derive the state table

Present state	Next state		Output z
	w = 0	w = 1	

Present state y_2y_1	Flip-flop inputs		Output z
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			
B			
C			
D			

Present state	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
y_2y_1	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

This step is easy
(map 2-bit numbers to 4 letters)

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
y_2y_1	T_2D_1	T_2D_1	z
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	1 0	0 1	0
1 1	1 0	0 1	1

This step is easy too
(the outputs are the same in both tables)

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	?		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

What should we do here?

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	?		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	1 0	0 1	0
1 1	1 0	0 1	1

What should we do here?

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	1 0	0 1	0
1 1	1 0	0 1	1

T	Q(t + 1)
0	Q(t)
1	$\overline{Q(t)}$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	Q(t)
1	$\overline{Q(t)}$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	1 0	0 1	0
1 1	1 0	0 1	1

T	Q(t + 1)
0	0
1	$\bar{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	1 0	0 1	0
1 1	1 0	0 1	1

T	Q(t + 1)
0	0
1	$\overline{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A			0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	1 0	0 1	0
1 1	1 0	0 1	1

T	Q(t + 1)
0	0
1	$\overline{Q(t)}$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

Note that A = 00

T	$Q(t+1)$
0	0
1	$\overline{Q}(t)$

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

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C		?	0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

What should we do here?

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	Q(t)
1	$\overline{Q(t)}$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	Q(t)
1	$\overline{Q(t)}$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	1
1	$\bar{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	1
1	$\bar{Q}(t)$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C			0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	z
	T_2D_1	T_2D_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	1
1	$\overline{Q(t)}$

D	Q(t + 1)
0	0
1	1

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A		0
B			0
C		D	0
D			1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

Note that D = 11

T	$Q(t+1)$
0	1
1	$\overline{Q}(t)$

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

Let's derive the state table

Present state	Next state		Output
	w = 0	w = 1	z
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

Present state y_2y_1	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

D	Q(t + 1)
0	0
1	1

The two tables for the initial circuit

Present state	Next state		Output
	w = 0	w = 1	z
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table

Present state	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
y_2y_1	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

Excitation table

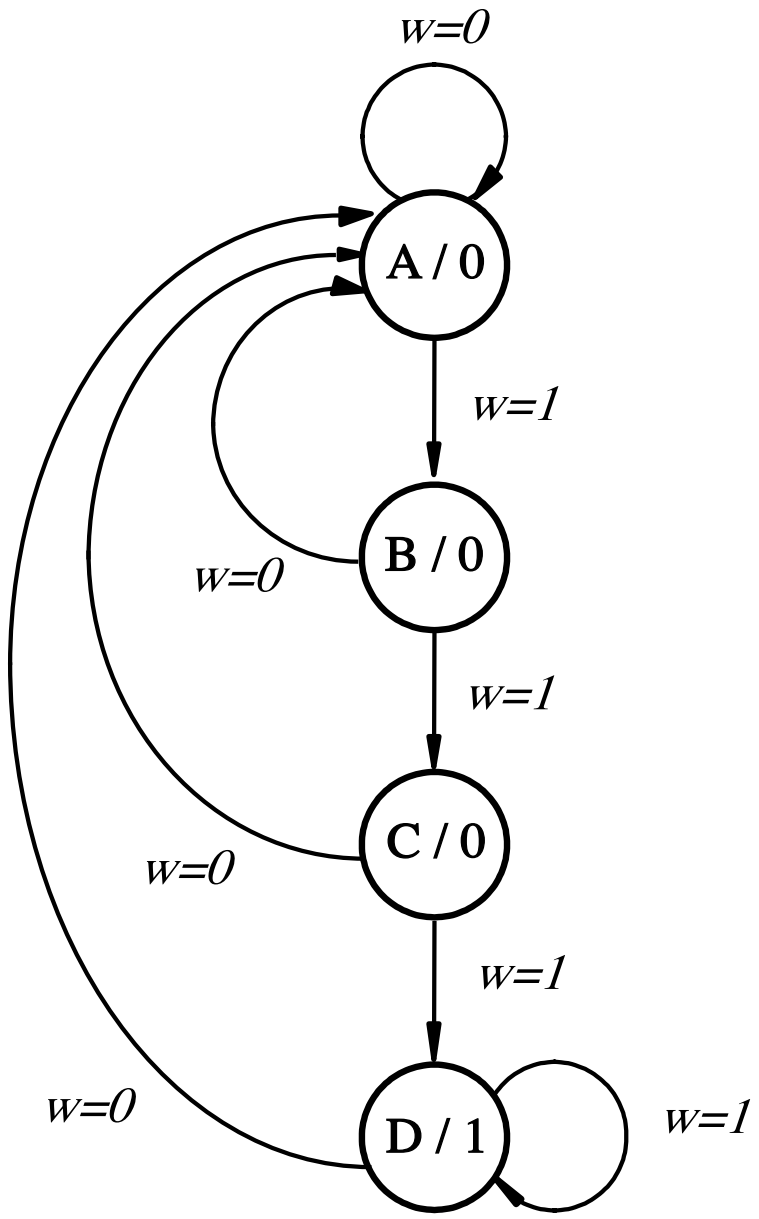
[Figure 6.75b from the textbook]

[Figure 6.80 from the textbook]

The state diagram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table



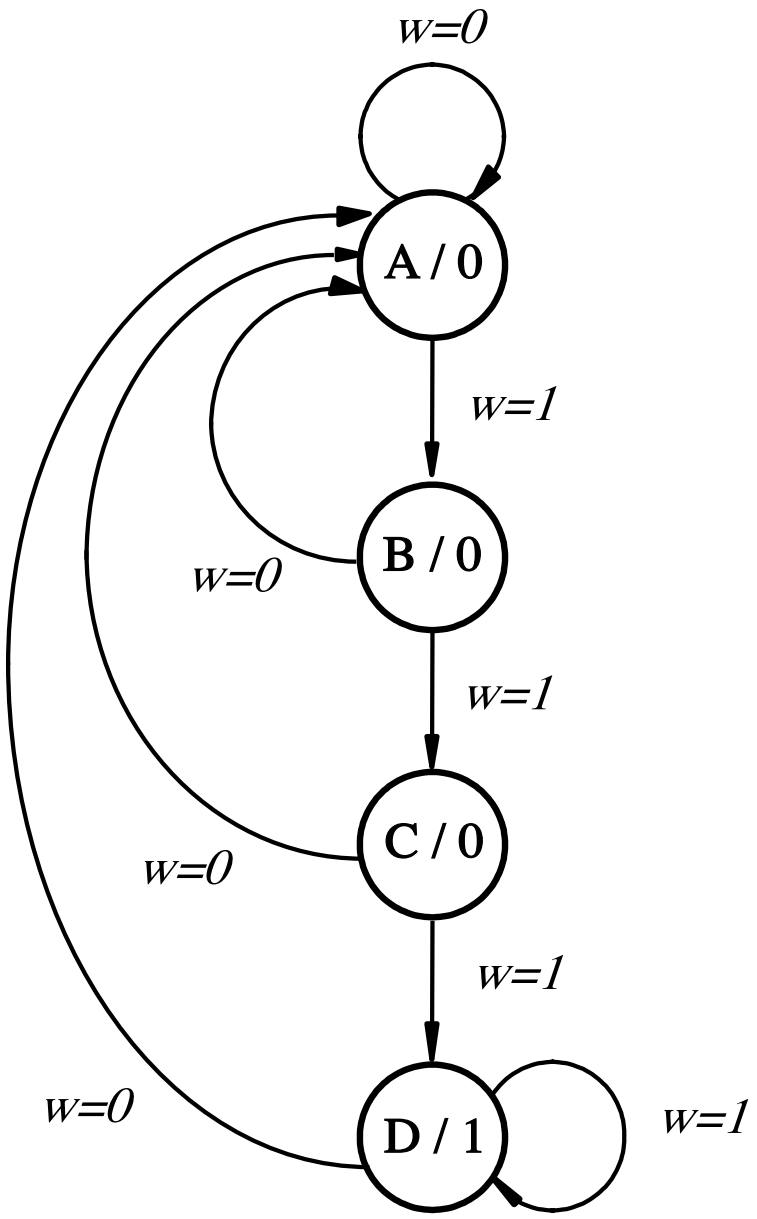
State diagram

The state diagram

Thus, this FSM is identical to the ones in the previous examples, even though the circuit uses JK flip-flops.

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

State table



State diagram

State Minimization

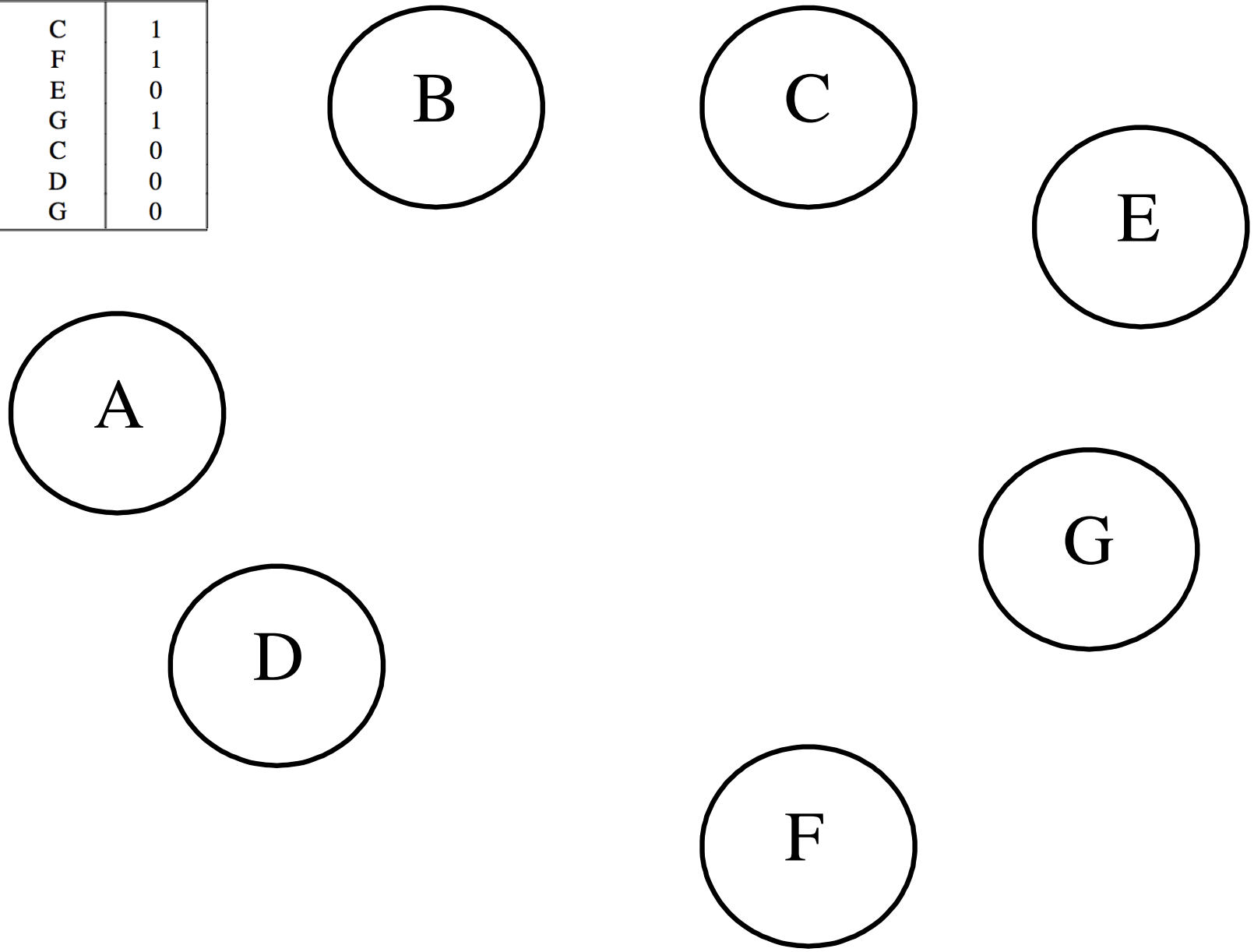
State Table for This Example

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

[Figure 6.51 from the textbook]

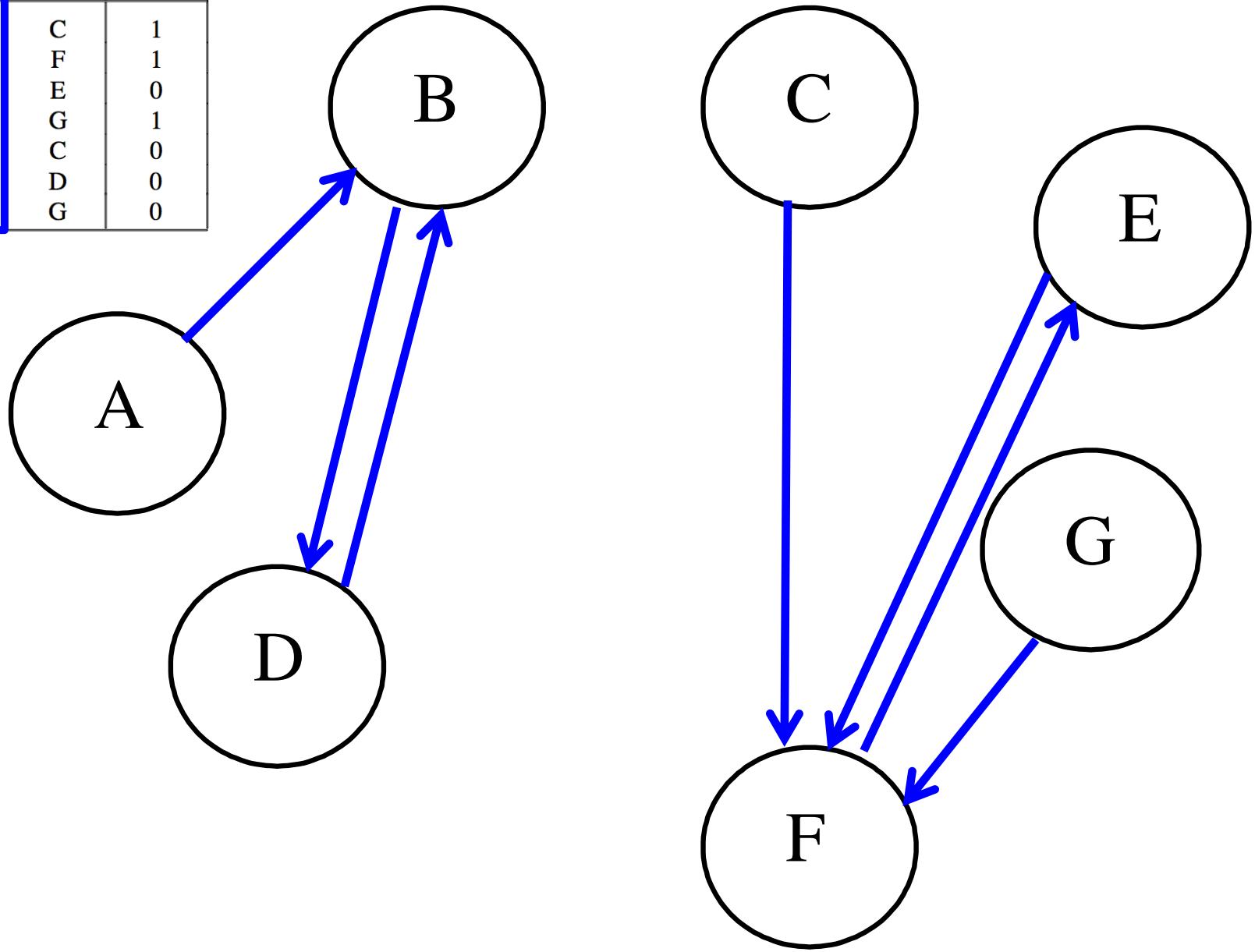
State Diagram
(just the states)

Present state	Next state $w = 0$	Next state $w = 1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0



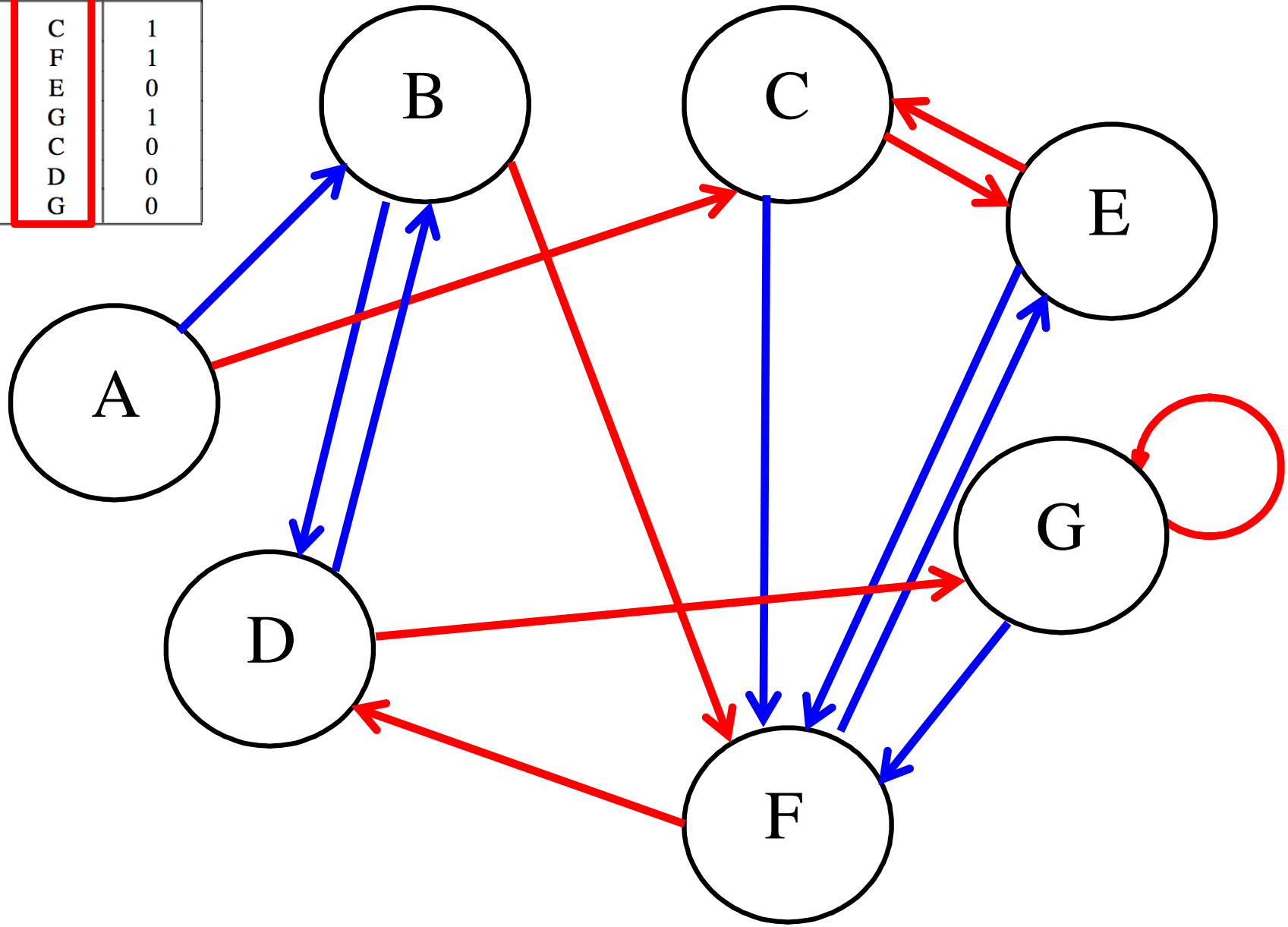
State Diagram
(transitions when w=0)

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0



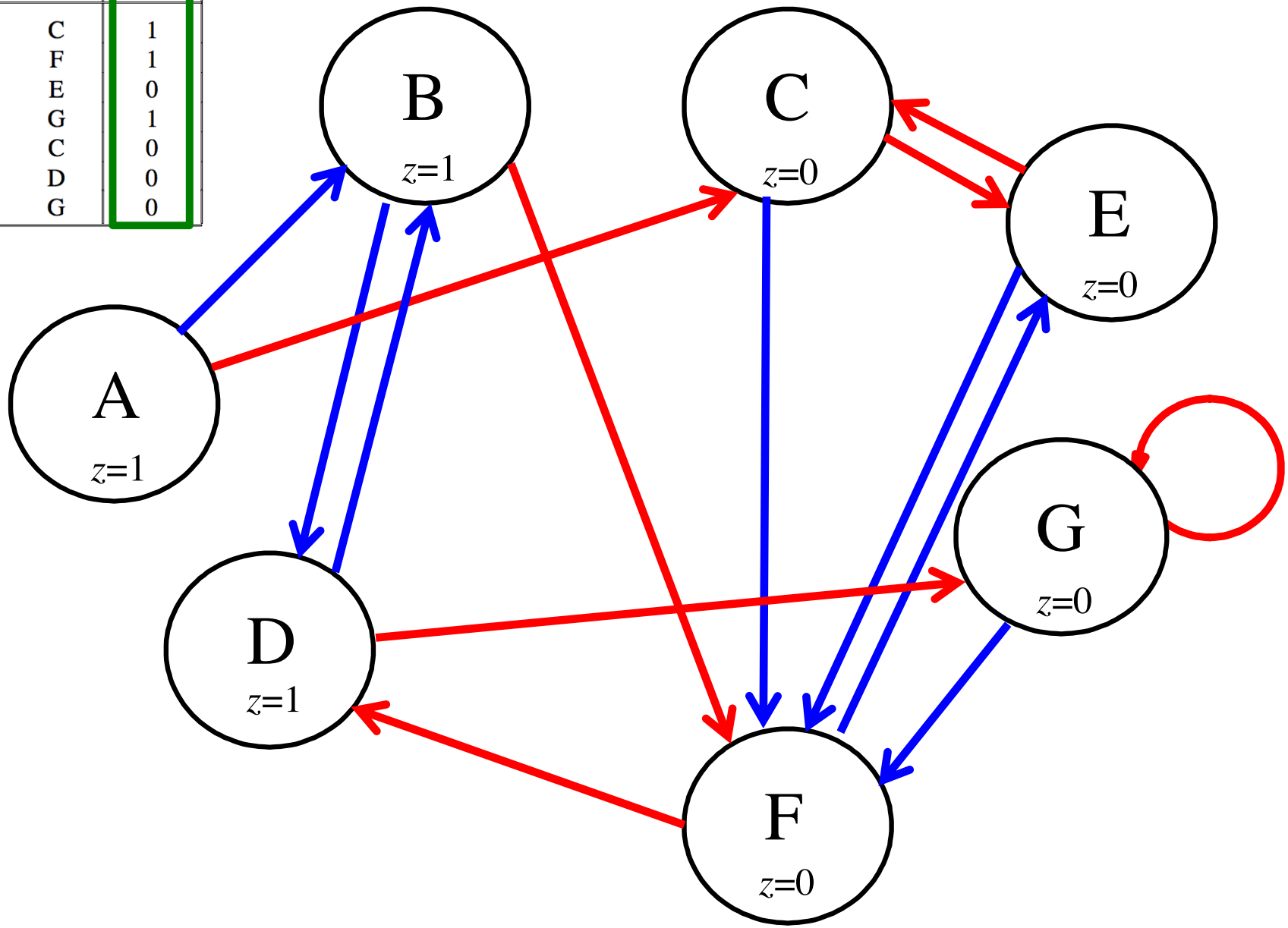
State Diagram
(transitions when w=1)

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0



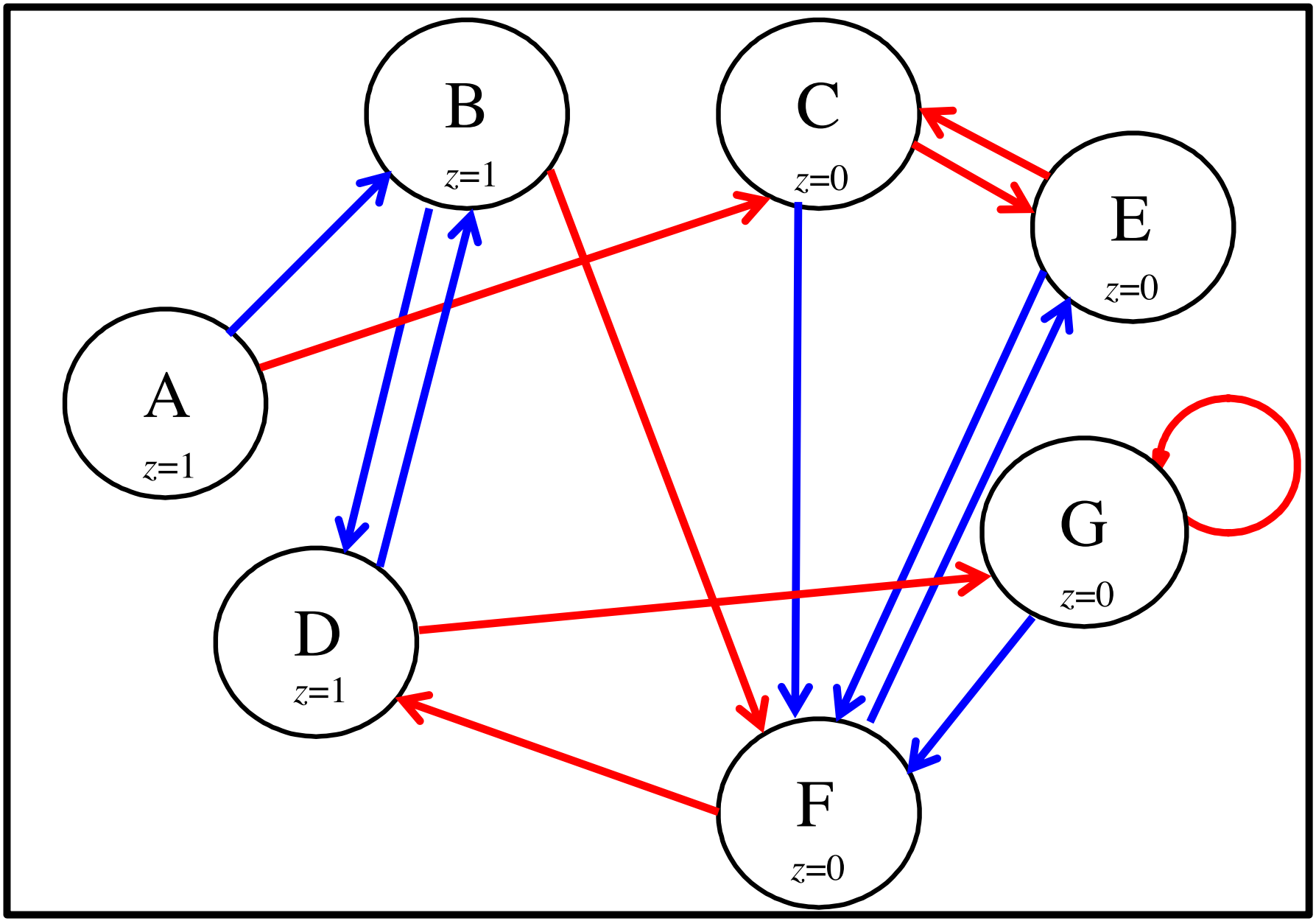
Outputs

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0



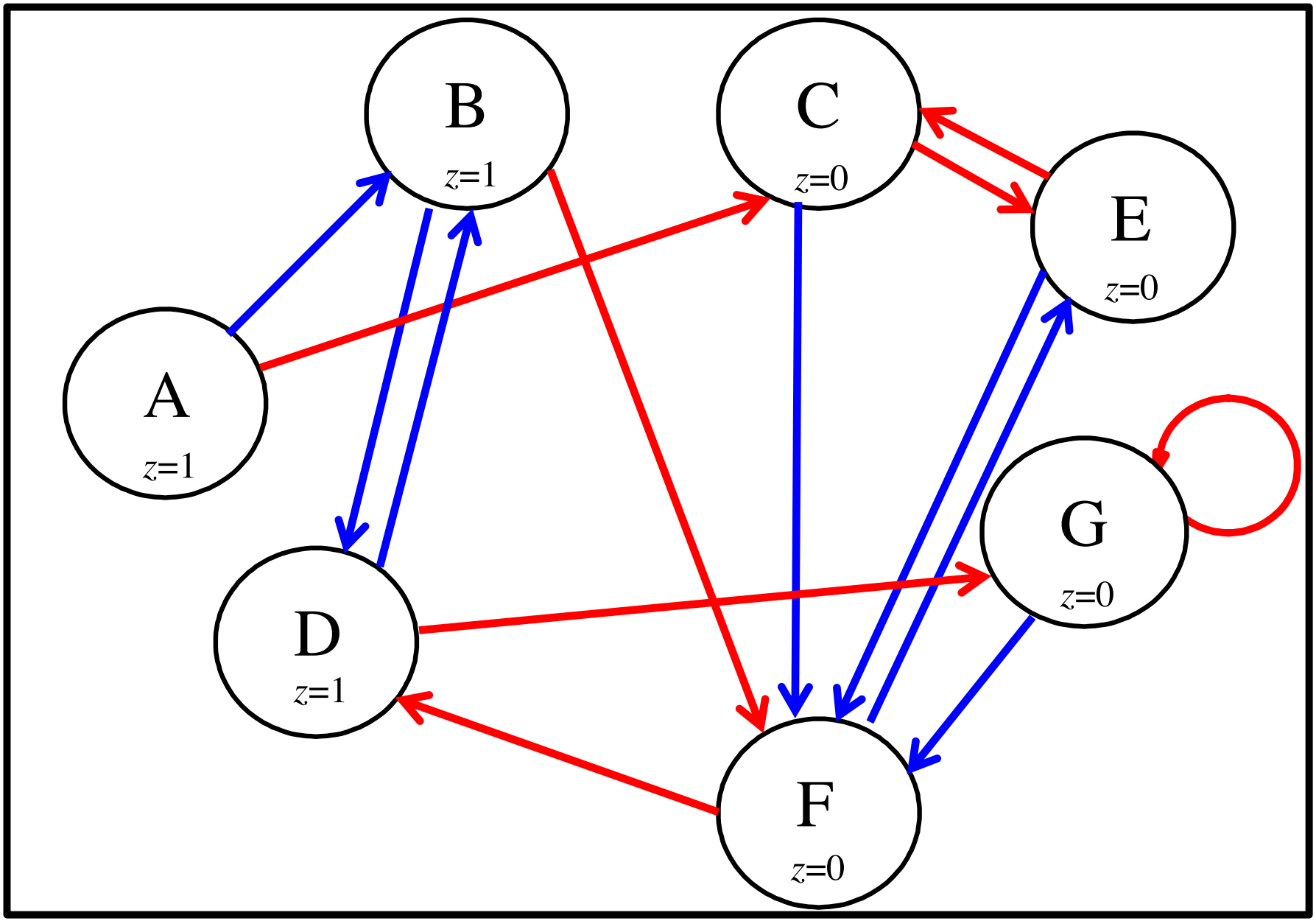
Partition #1

(All states in the same partition)



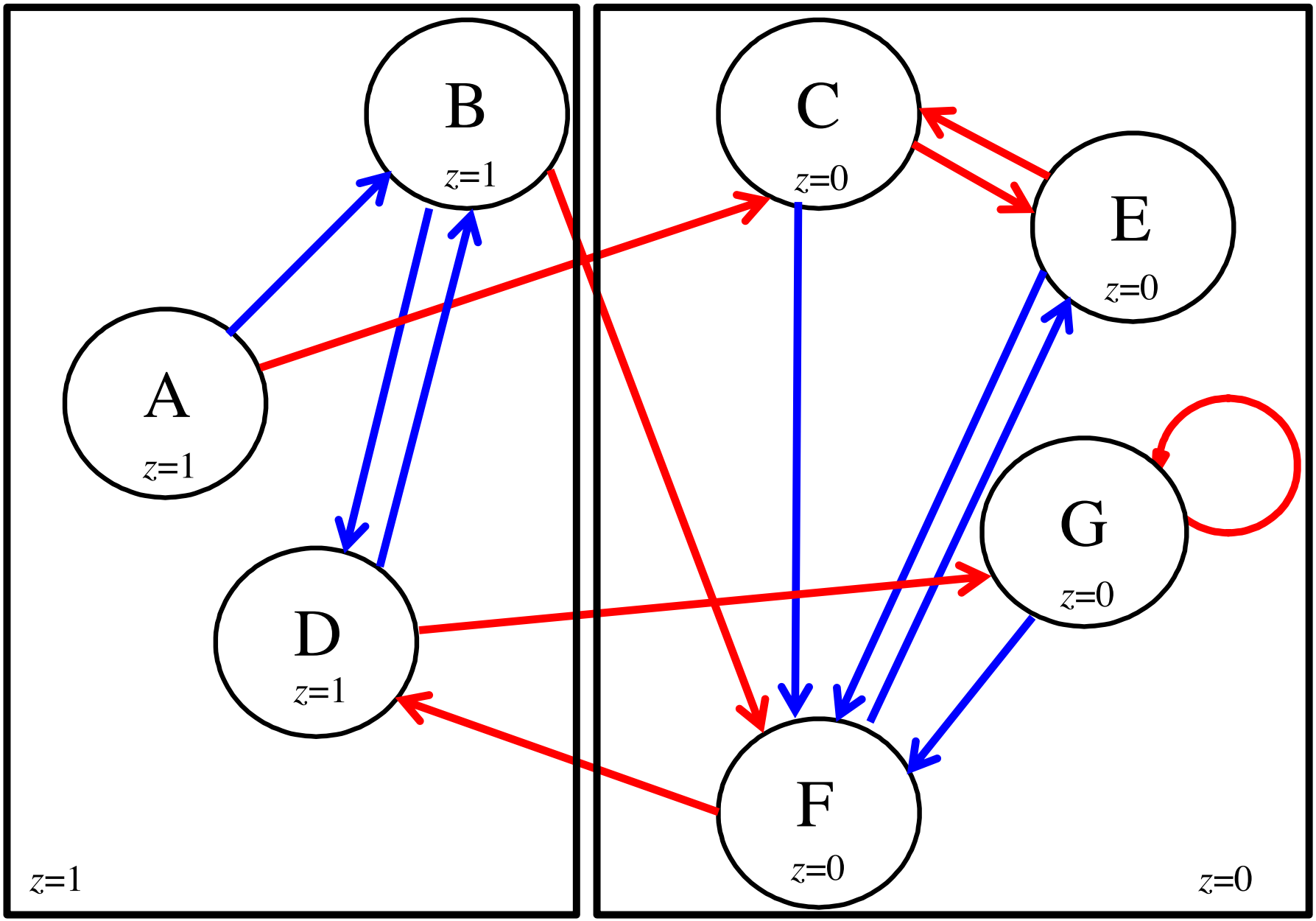
Partition #1

(ABCDEFG)



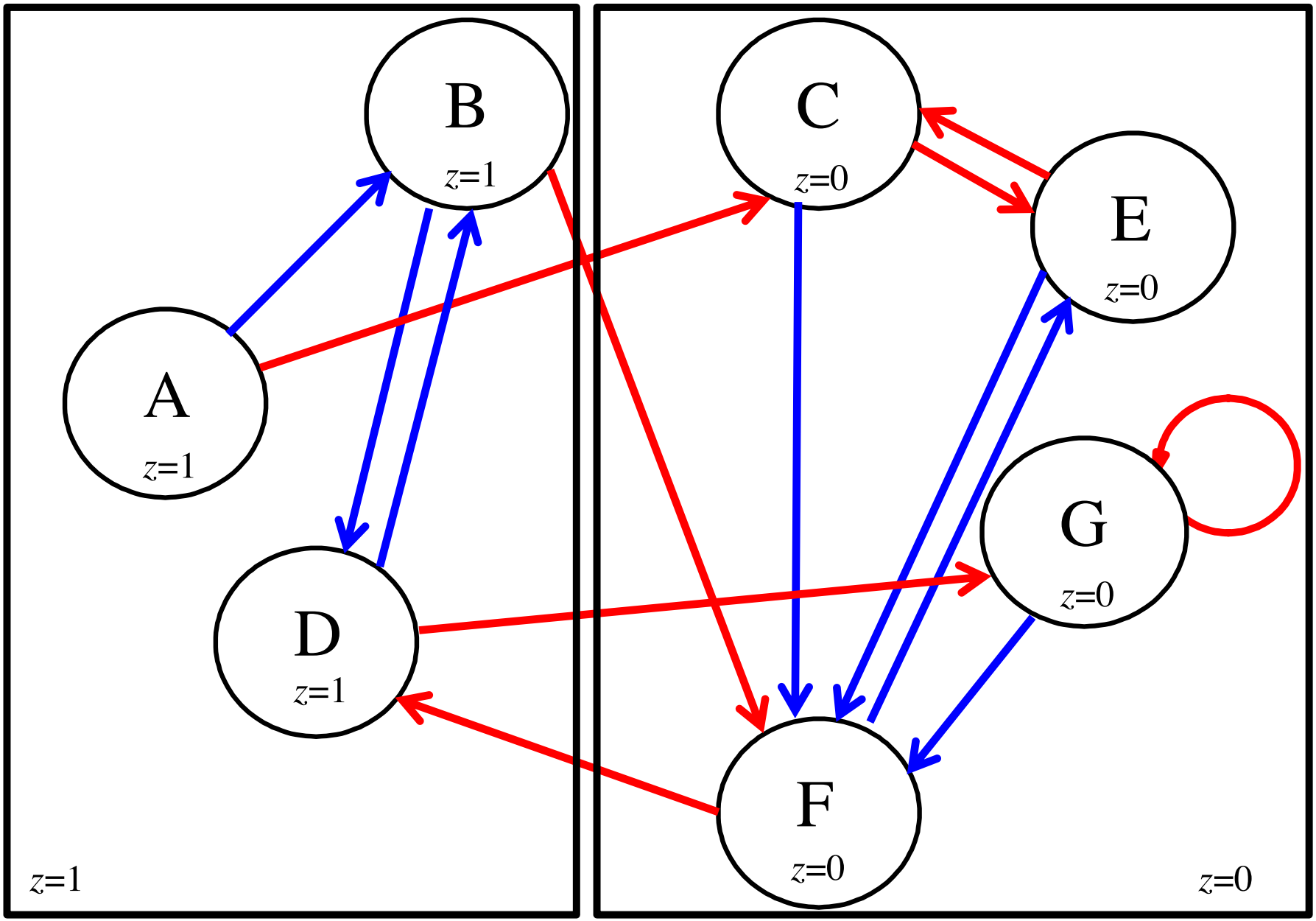
Partition #2

(based on outputs)



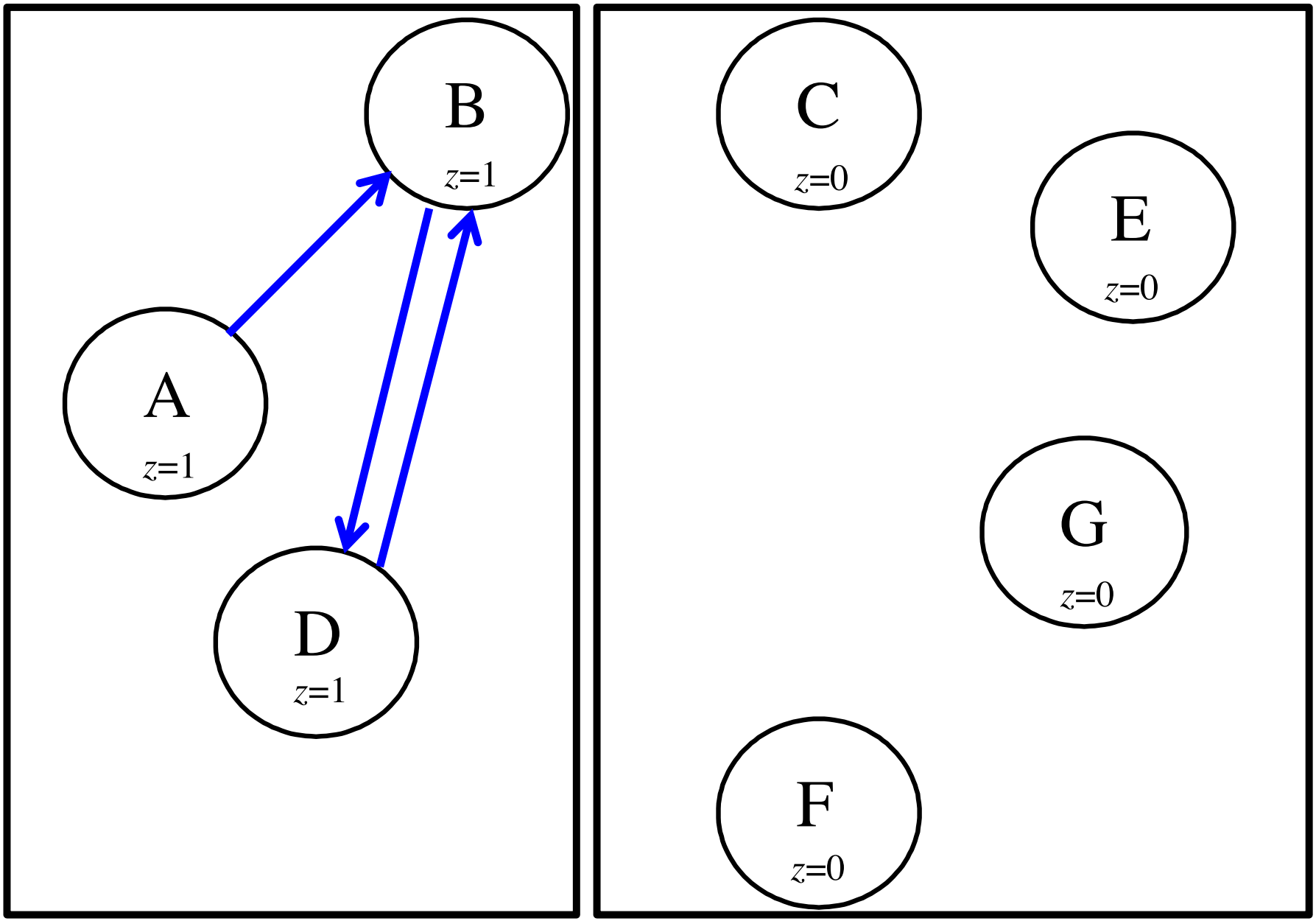
Partition #2

(ABD)(CEFG)



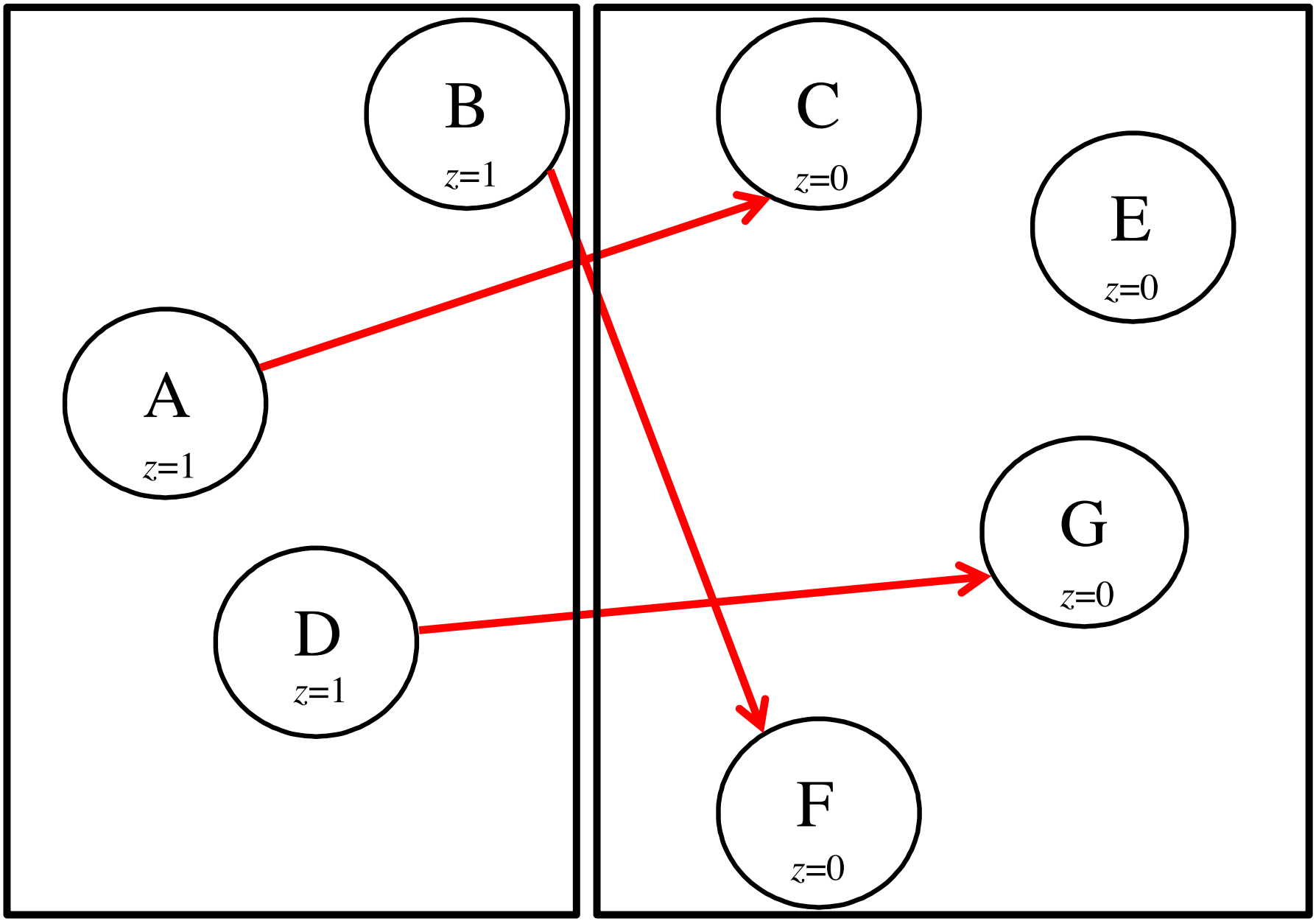
Partition #3.1

(Examine the 0-successors of ABD)



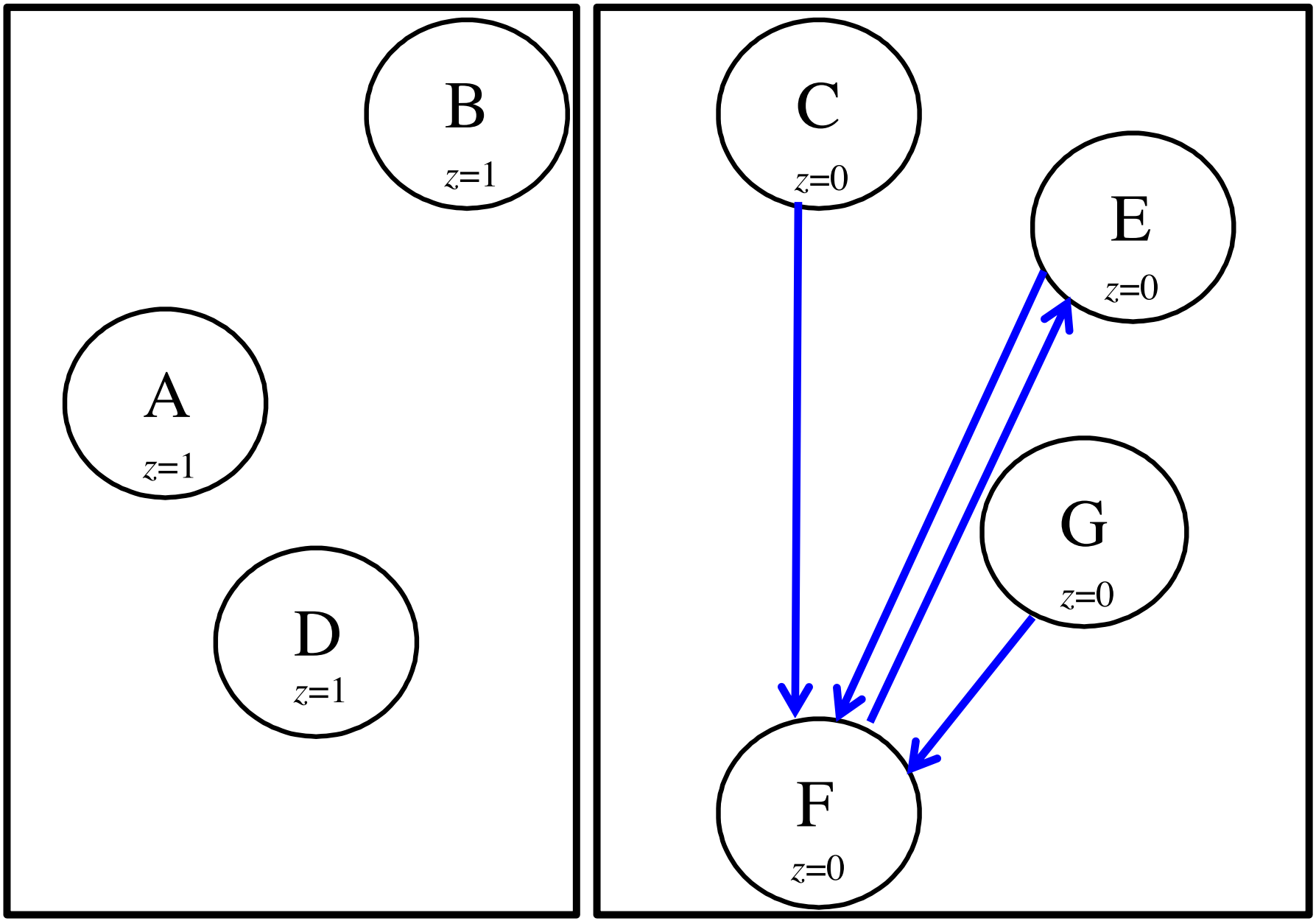
Partition #3.1

(Examine the 1-successors of ABD)



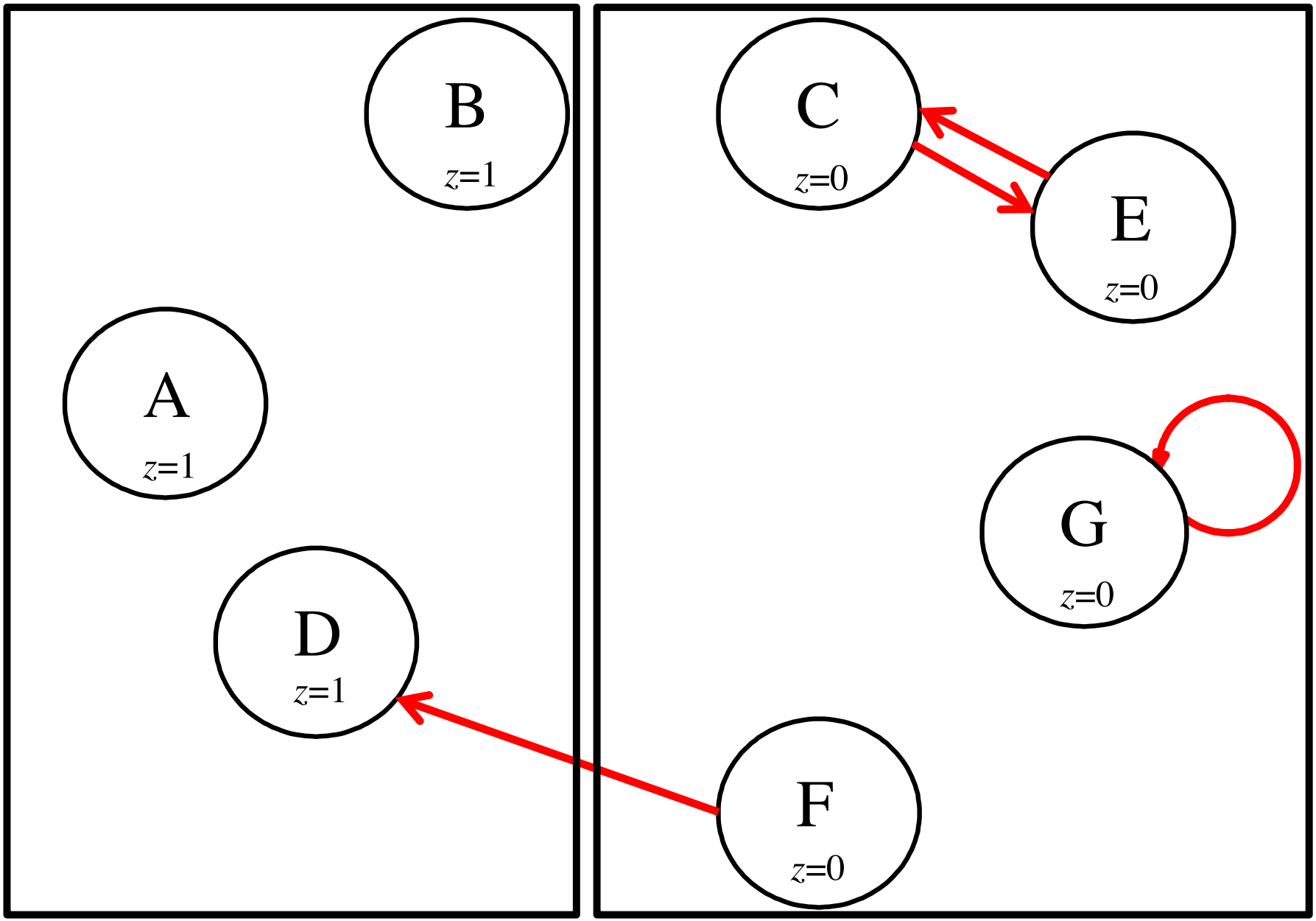
Partition #3.2

(Examine the 0-successors of CEFG)



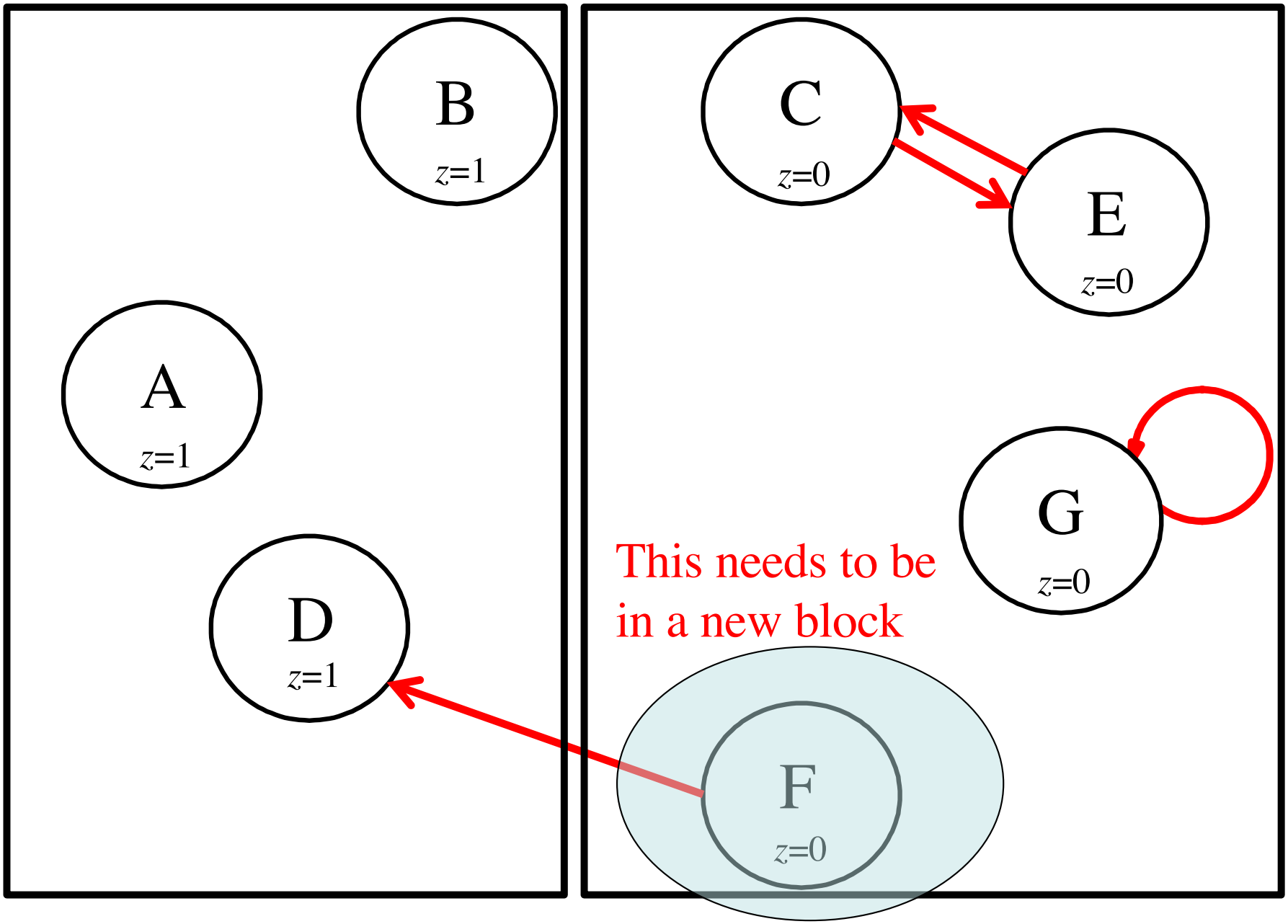
Partition #3.2

(Examine the 1-successors of CEFG)



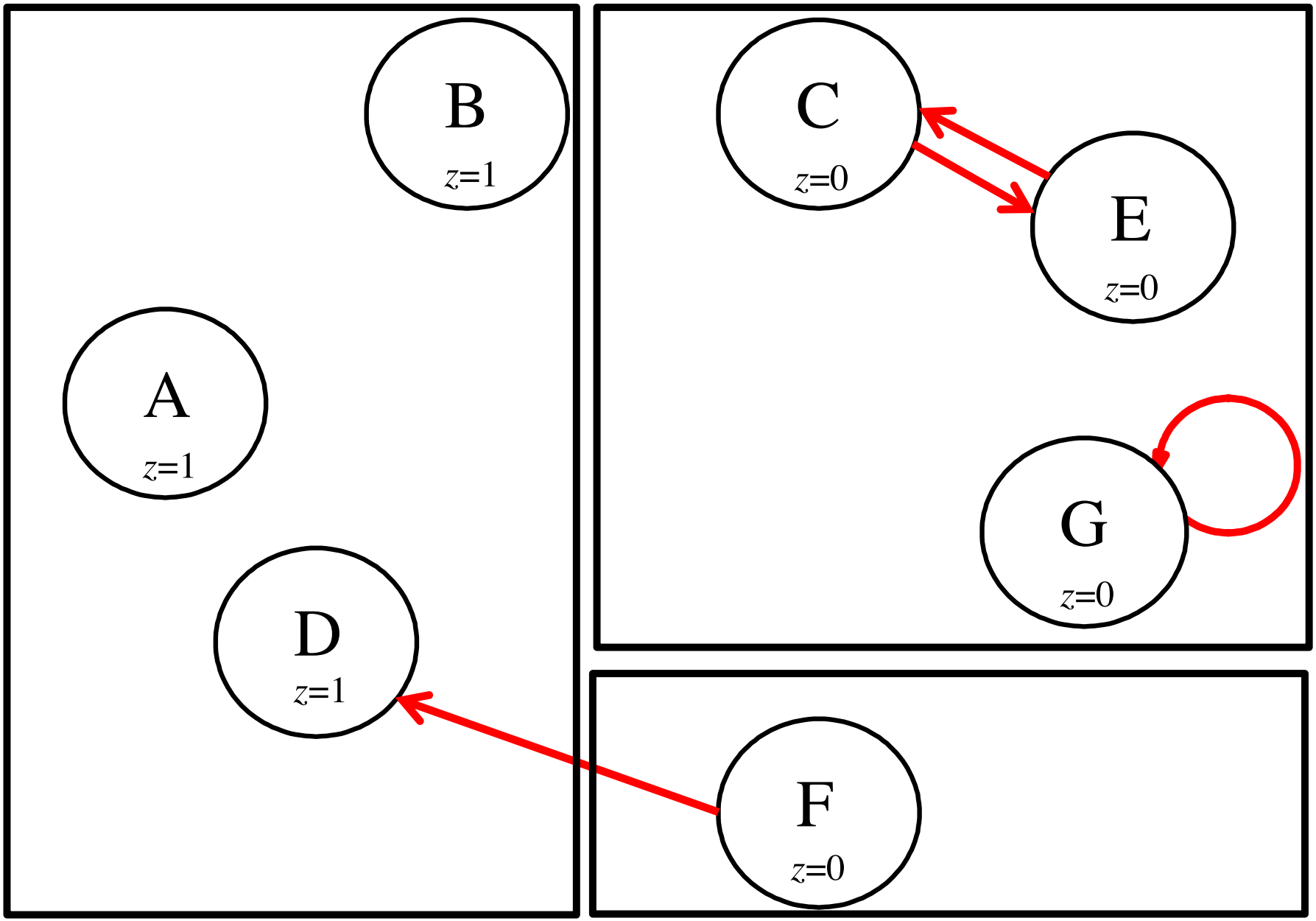
Partition #3.2

(Examine the 1-successors of CEFG)



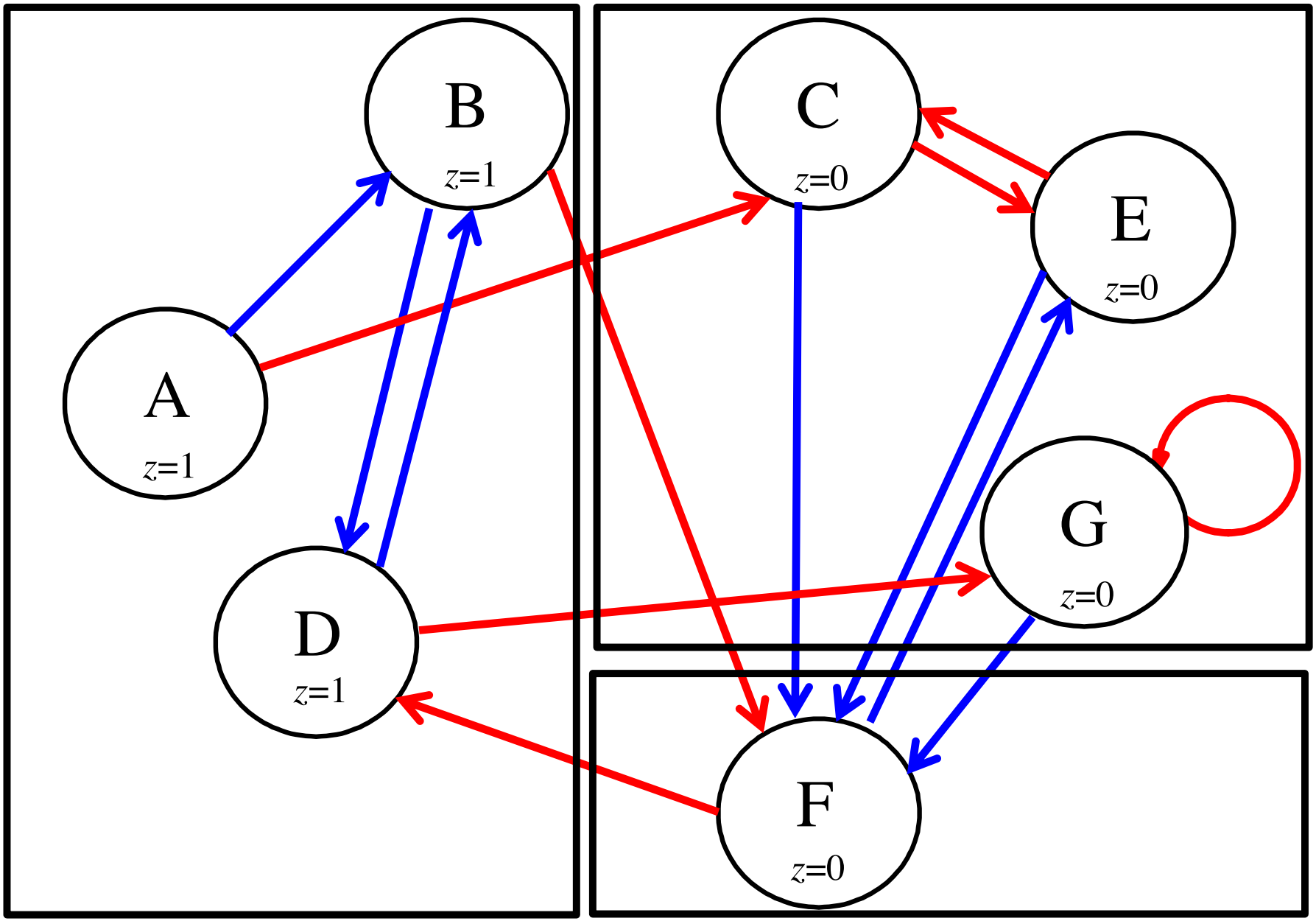
Partition #3

(ABD)(CEG)(F)



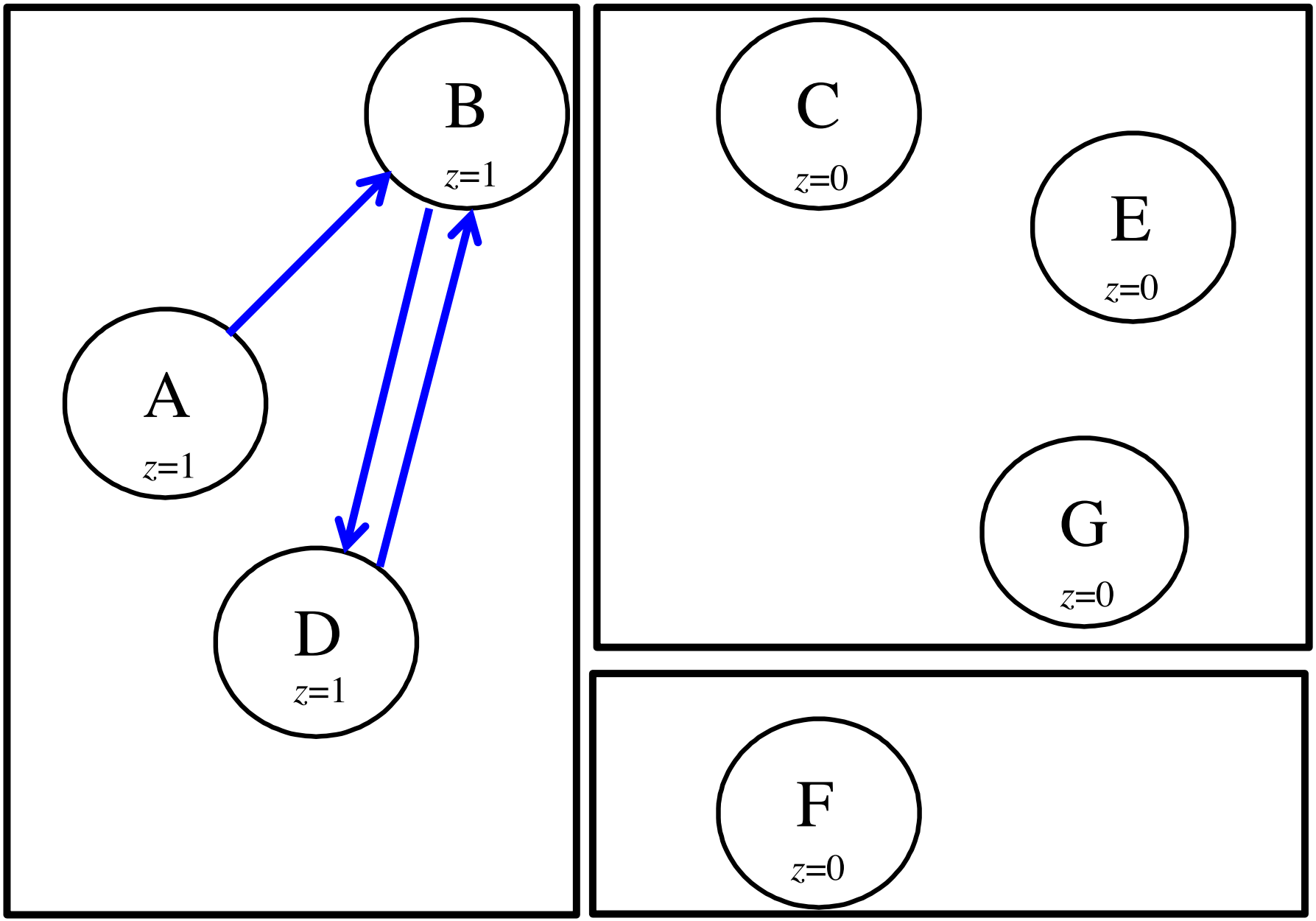
Partition #3

(ABD)(CEG)(F)



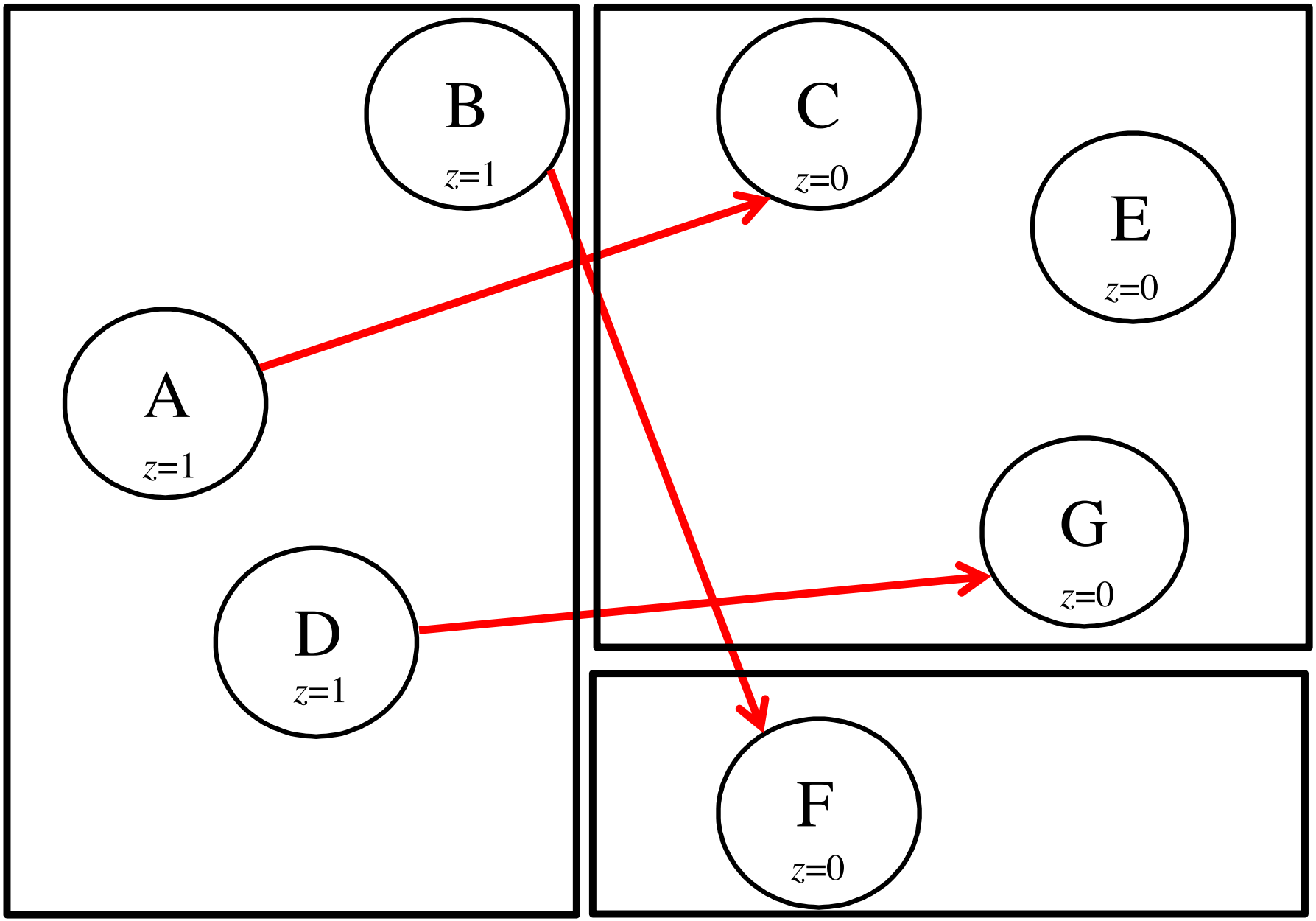
Partition #4.1

(Examine the 0-successors of ABD)



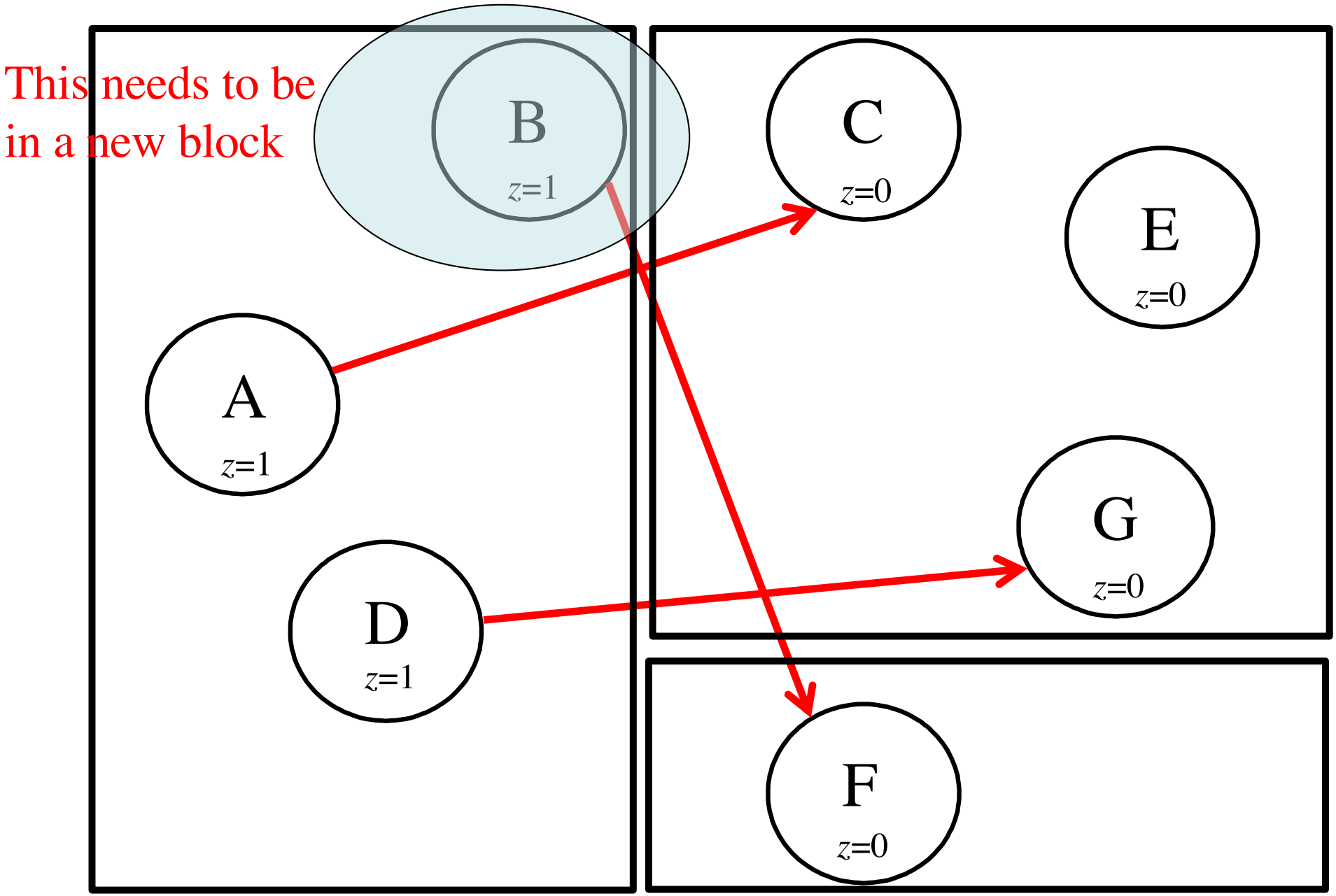
Partition #4.1

(Examine the 1-successors of ABD)



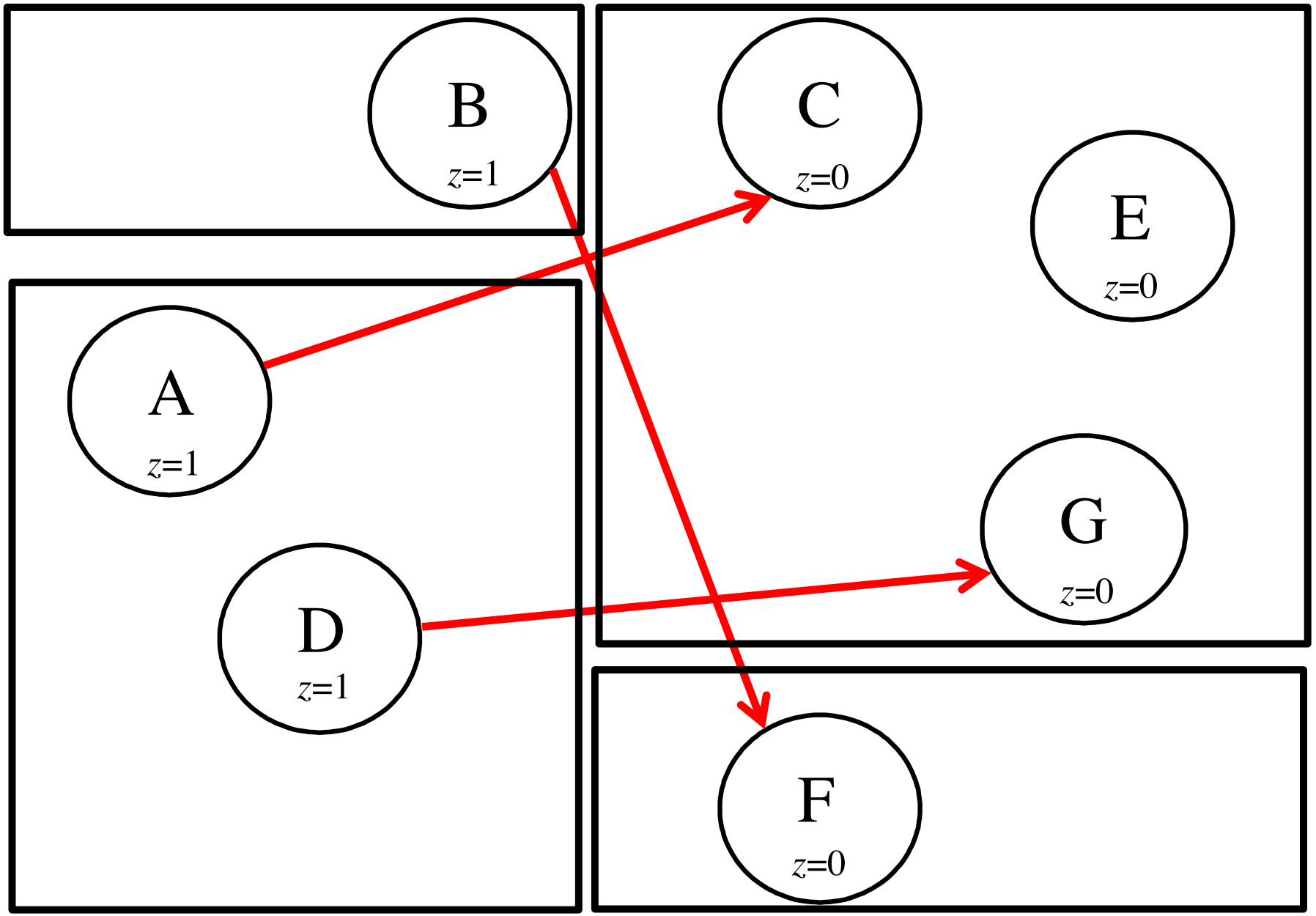
Partition #4.1

(Examine the 1-successors of ABD)



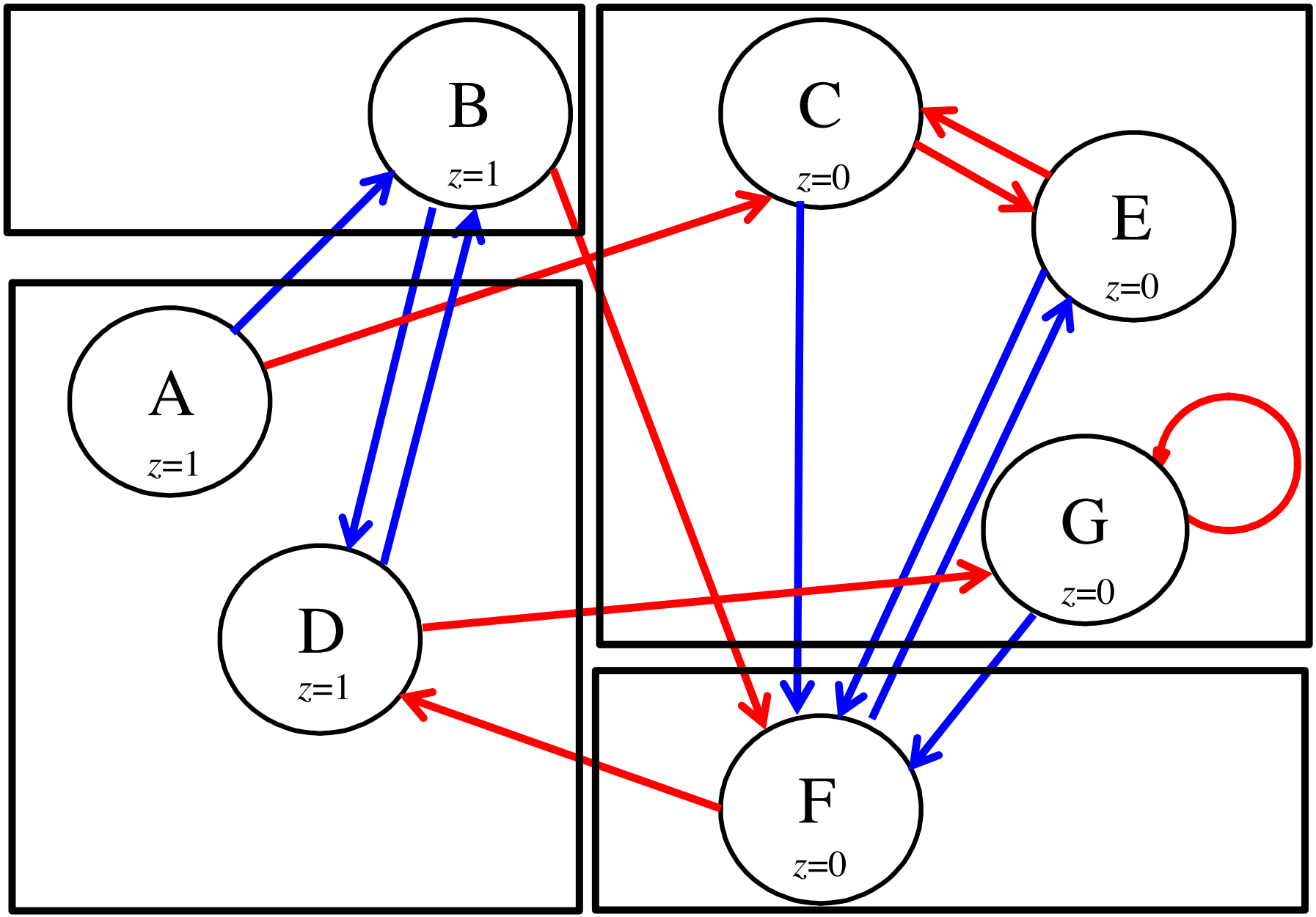
Partition #4

(AD)(B)(CEG)(F)



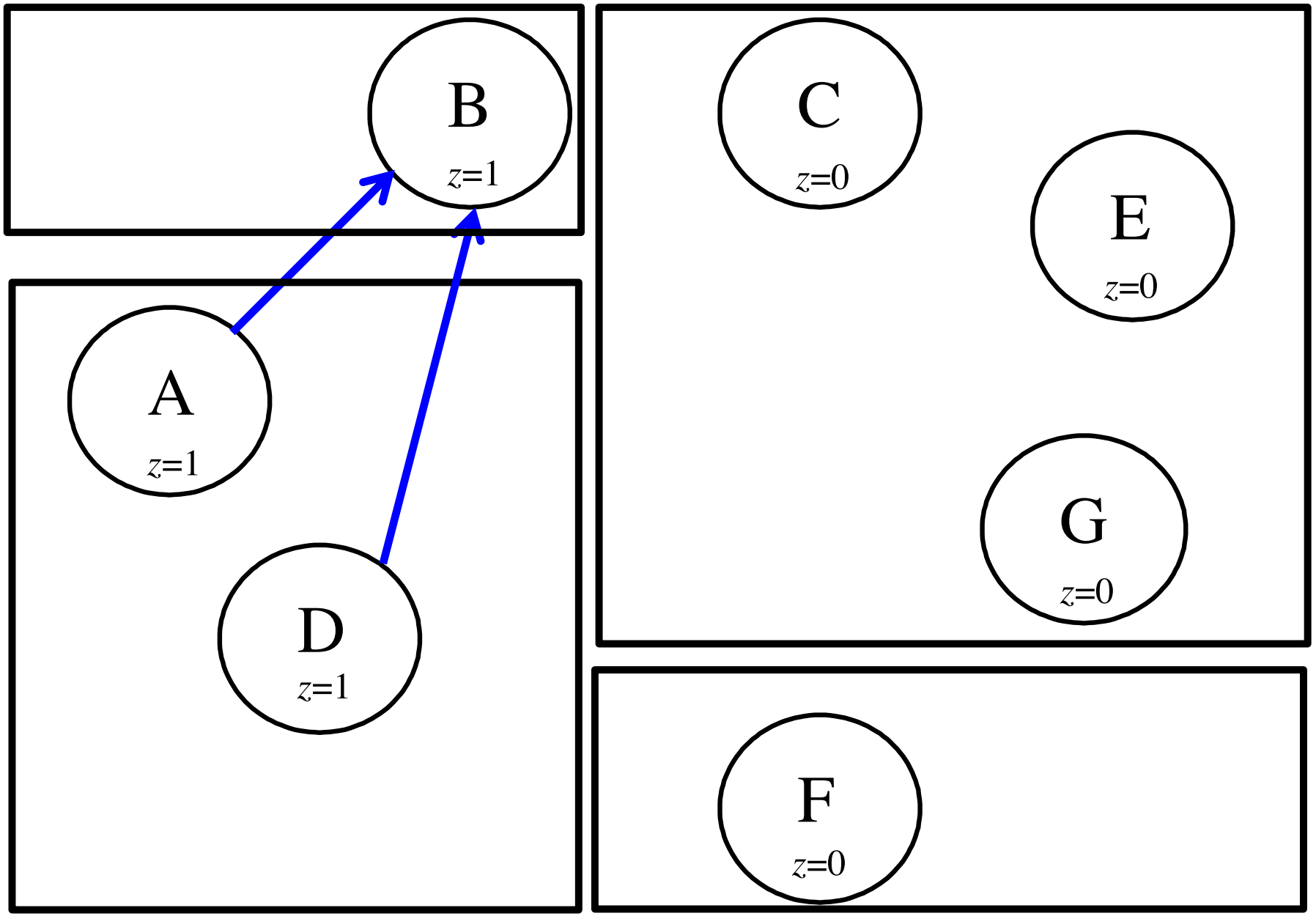
Partition #4

(AD)(B)(CEG)(F)



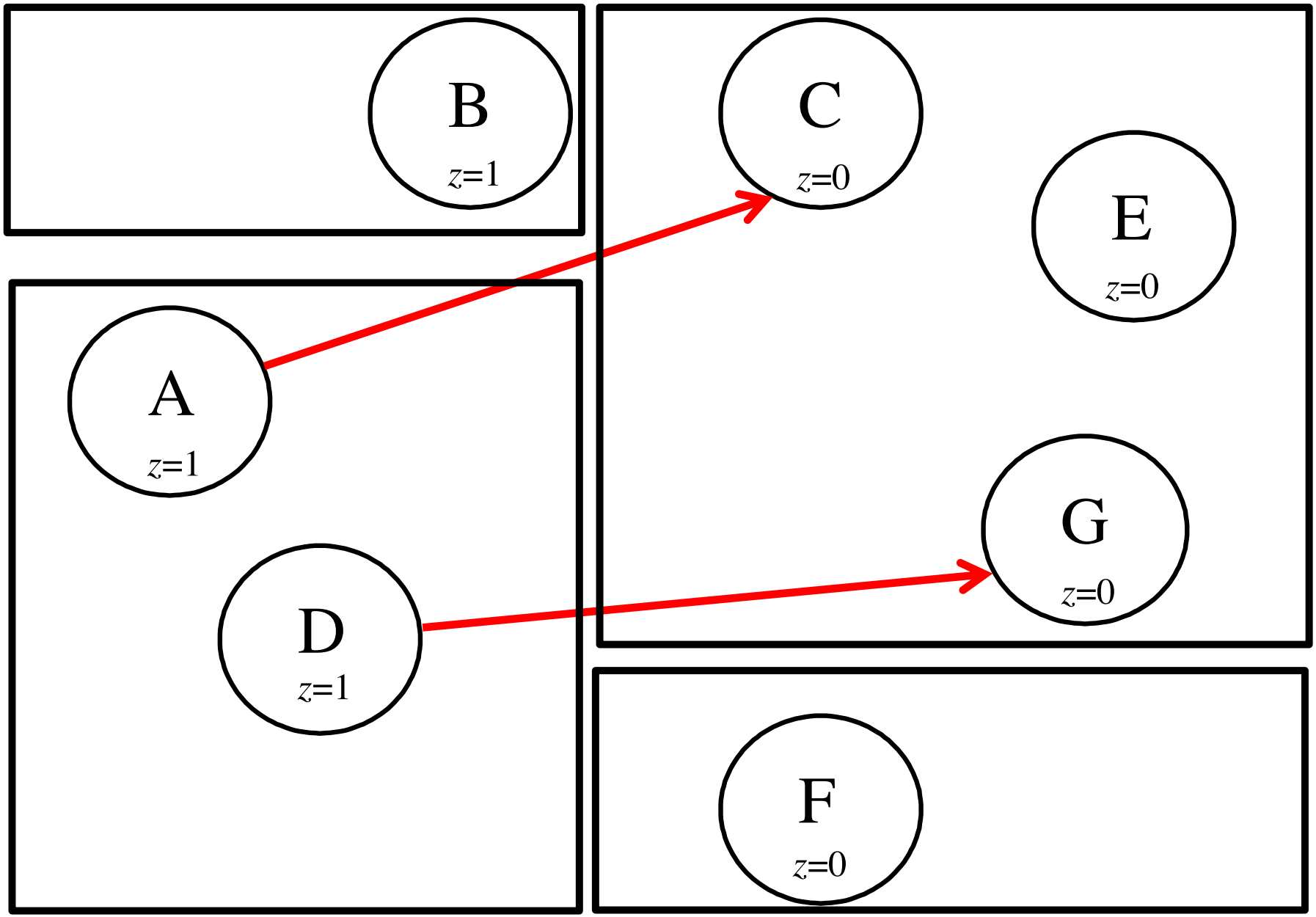
Partition #5.1

(Examine the 0-successors of AD)



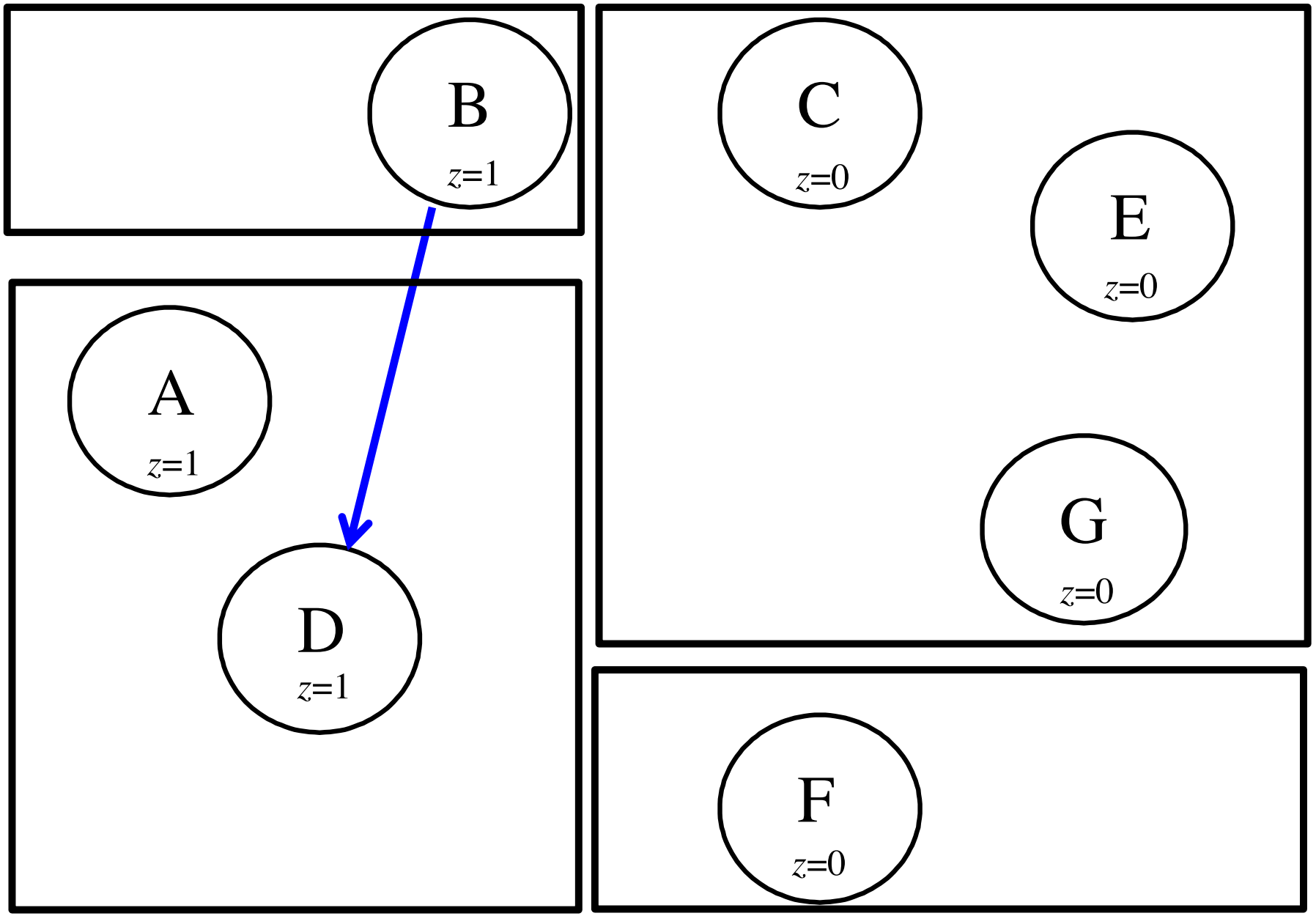
Partition #5.1

(Examine the 1-successors of AD)



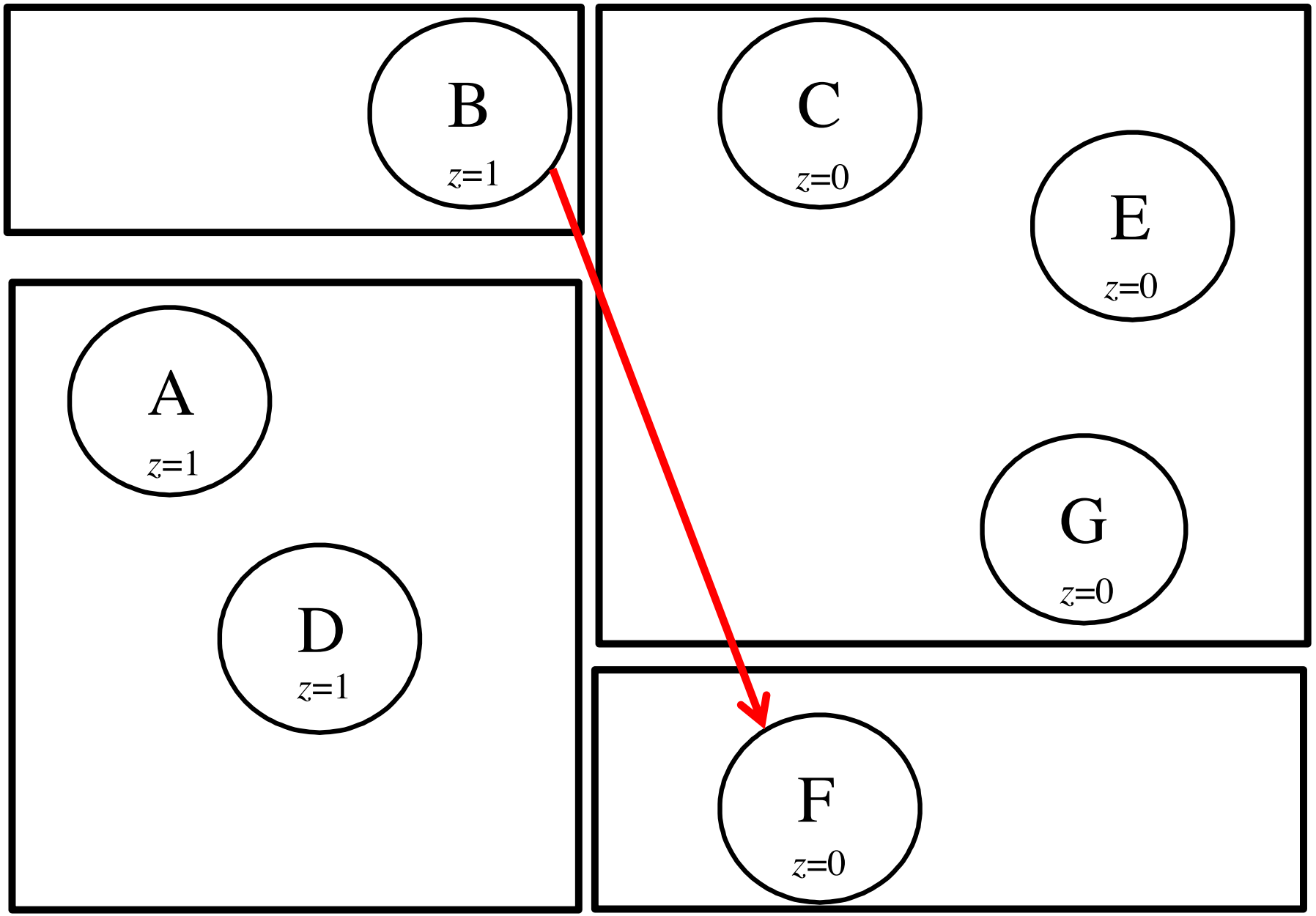
Partition #5.2

(Examine the 0-successors of B)



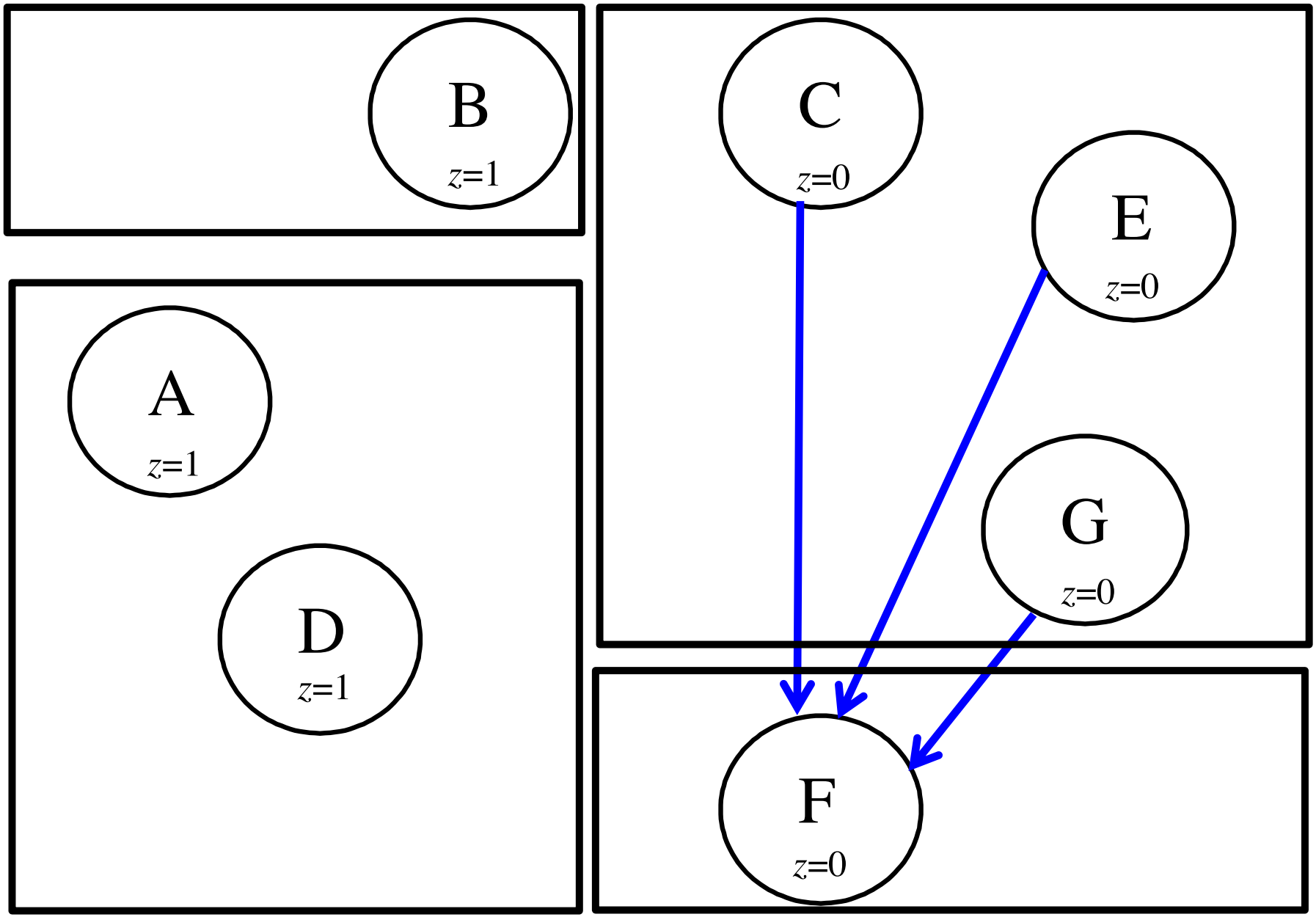
Partition #5.2

(Examine the 1-successors of B)



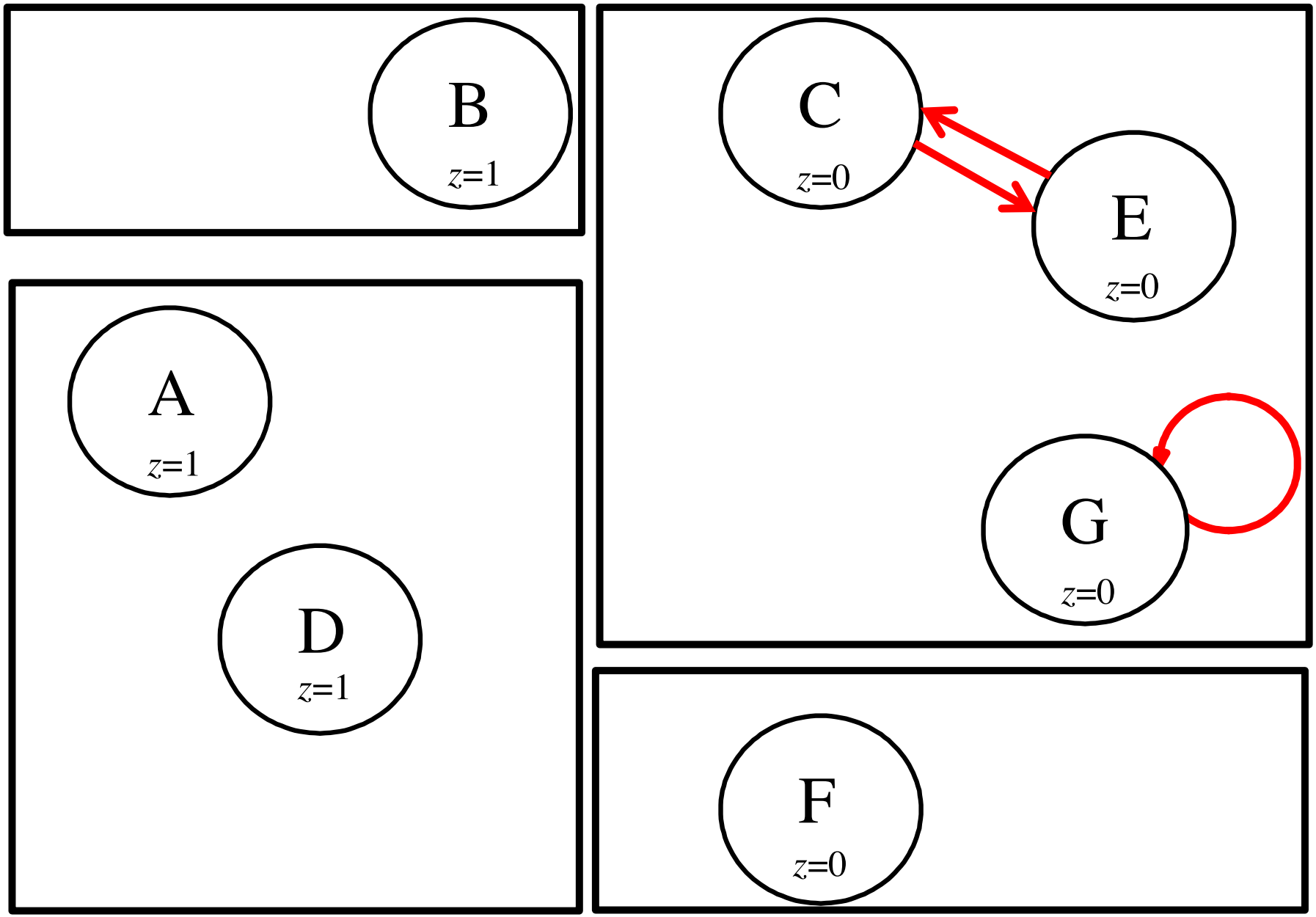
Partition #5.3

(Examine the 0-successors of CEG)



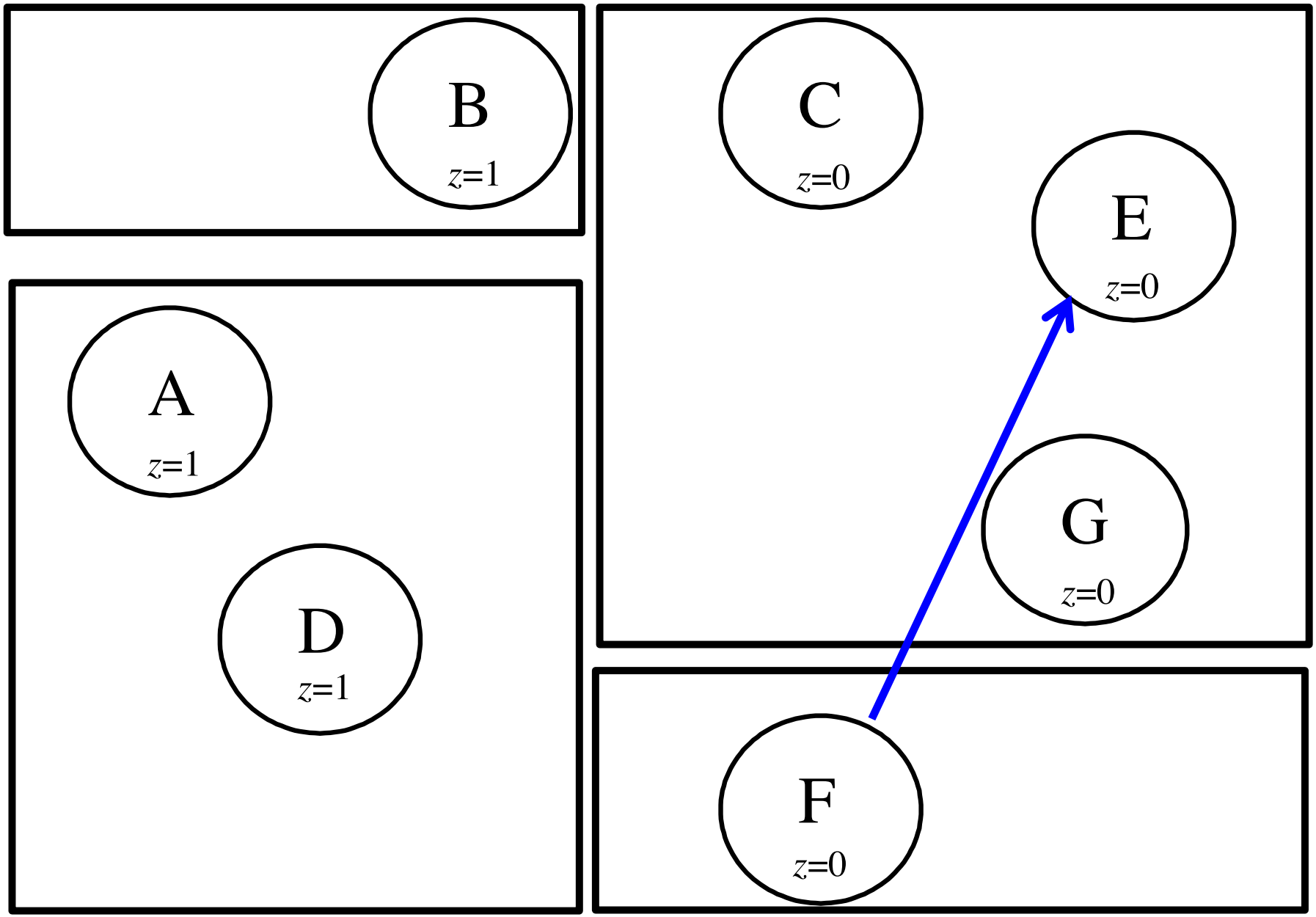
Partition #5.3

(Examine the 1-successors of CEG)



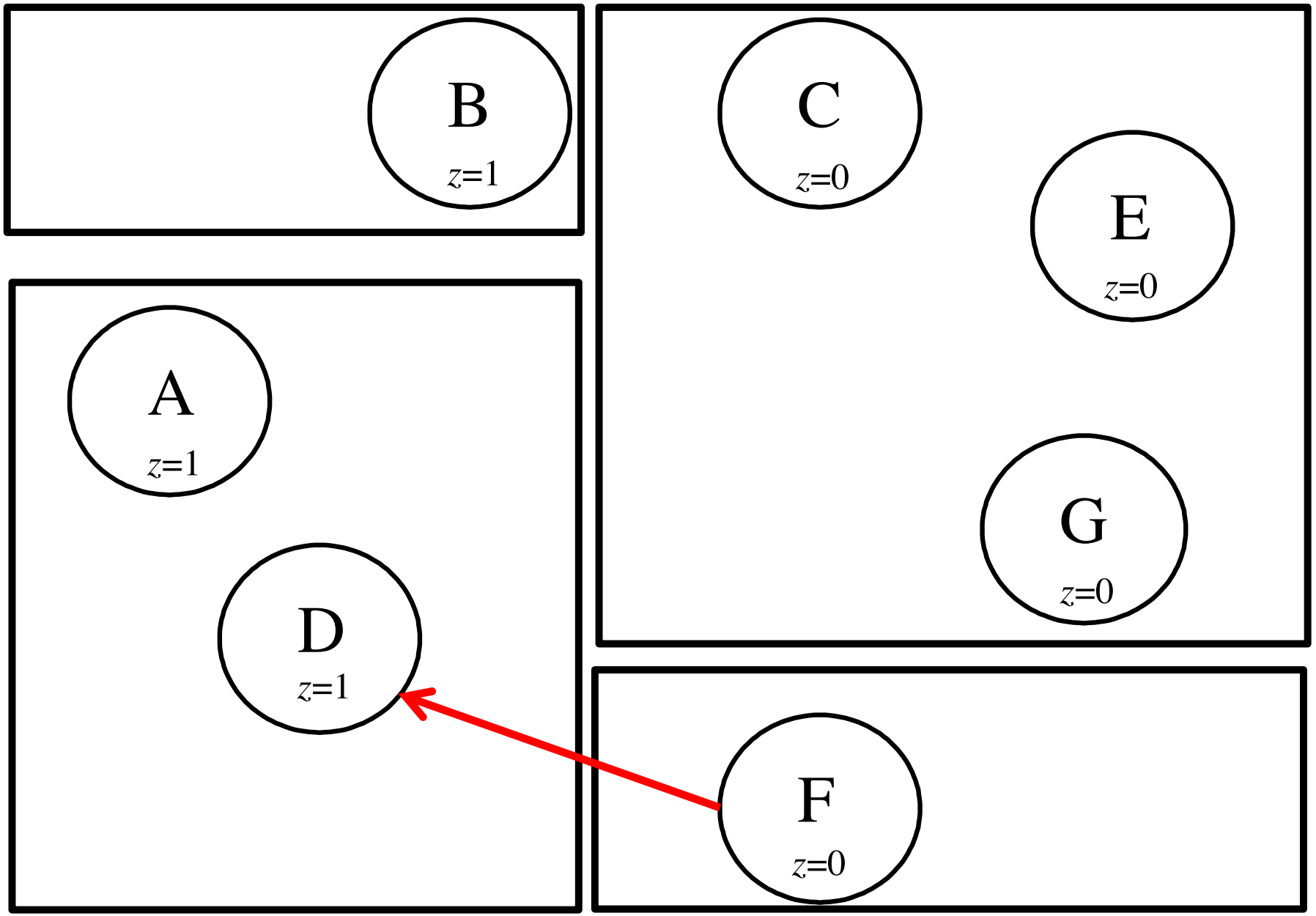
Partition #5.4

(Examine the 0-successors of F)



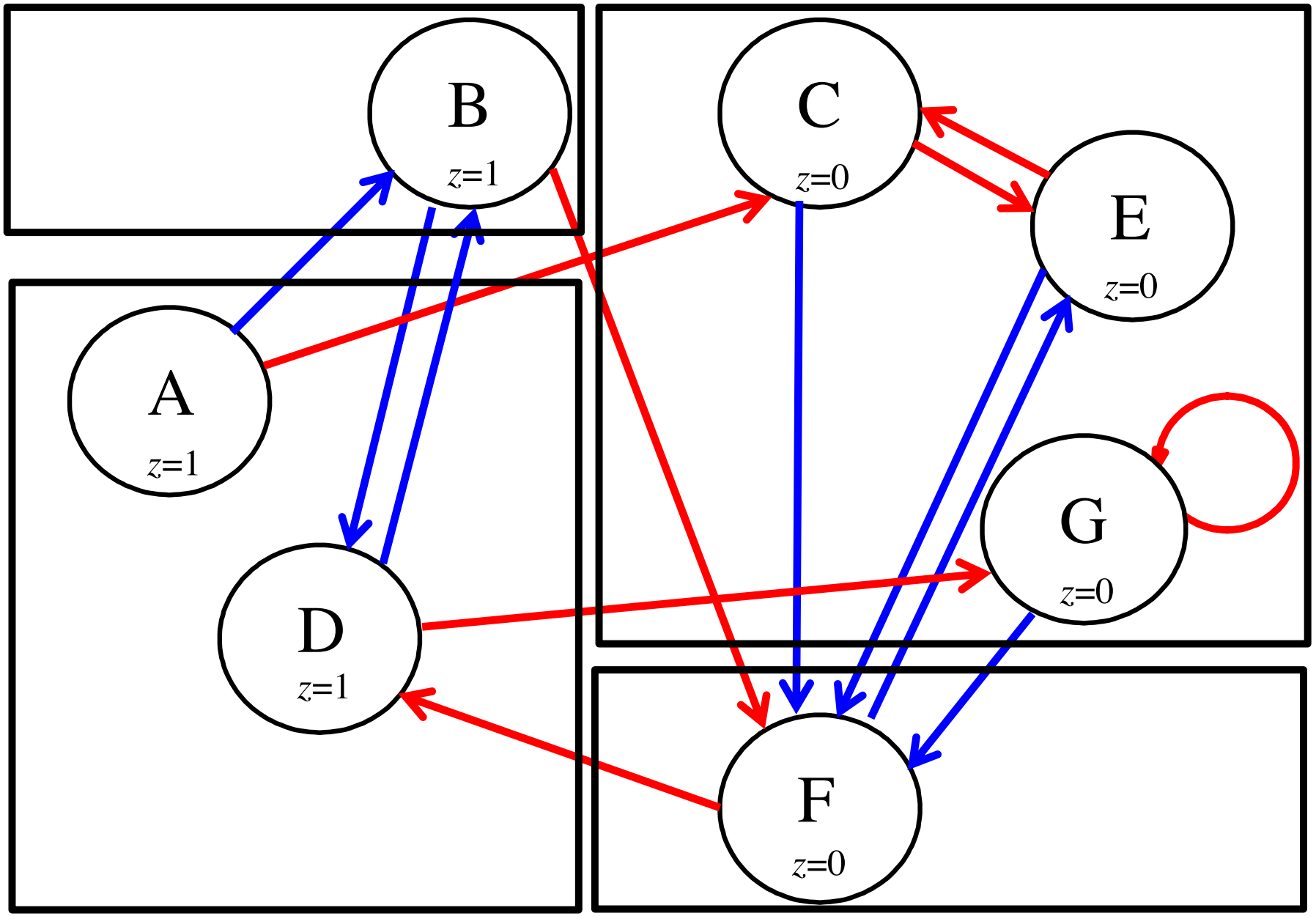
Partition #5.4

(Examine the 1-successors of F)



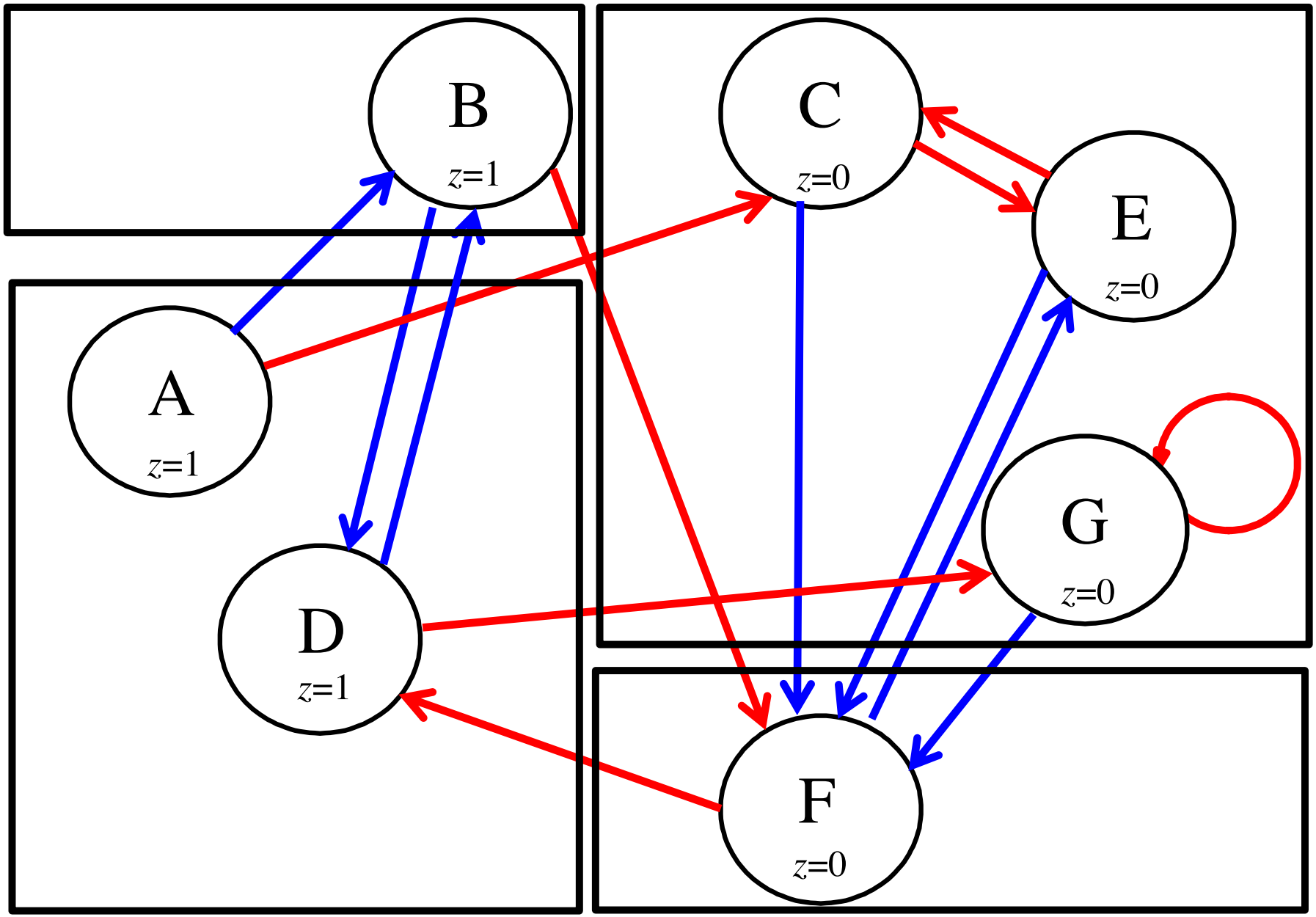
Partition #5

(AD)(B)(CEG)(F)



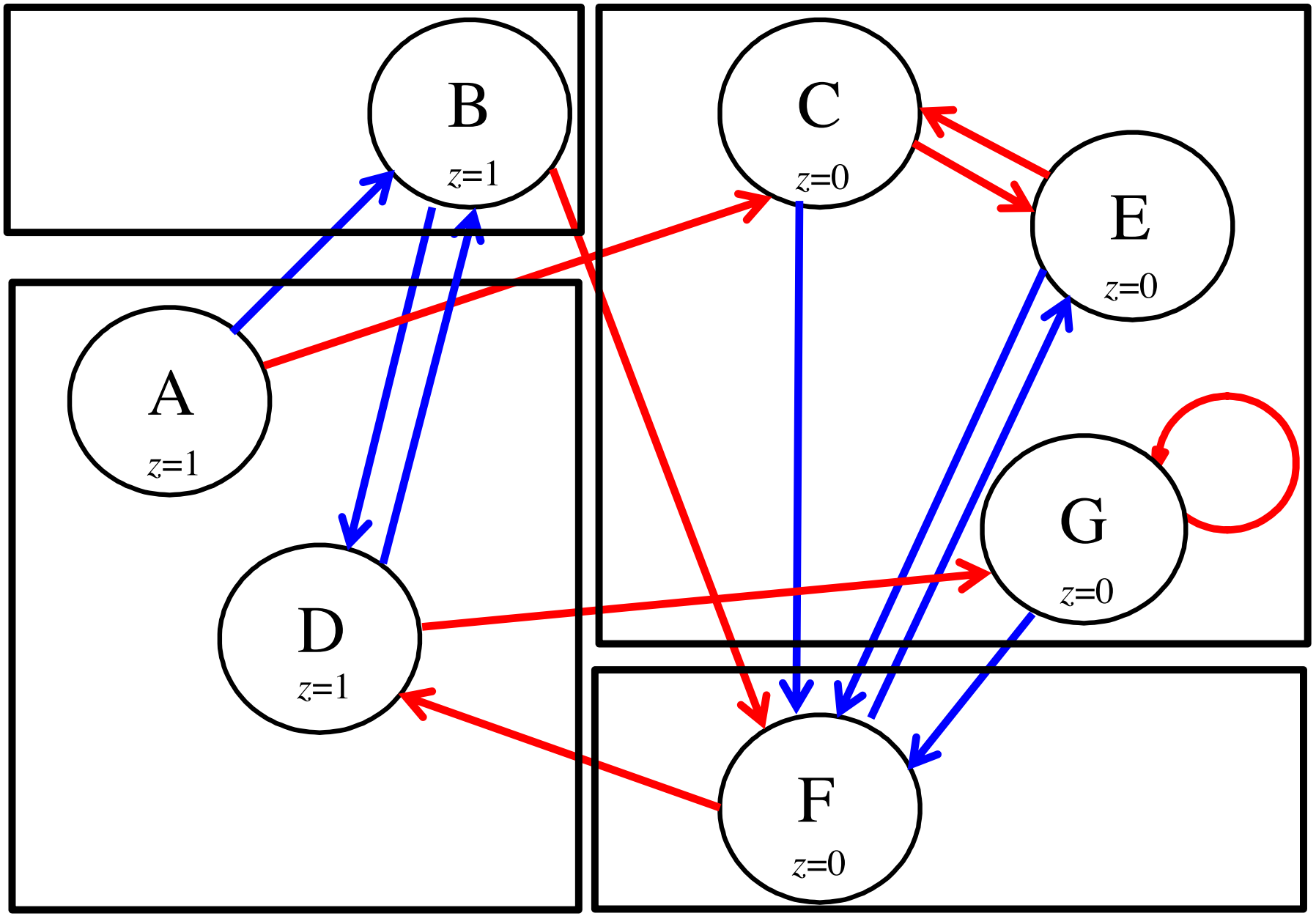
Partition #4

(AD)(B)(CEG)(F)



Partition #5

(This is the same as #4 so we can stop here)



Minimized state table

Present state	Nextstate		Output z
	w = 0	w = 1	
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

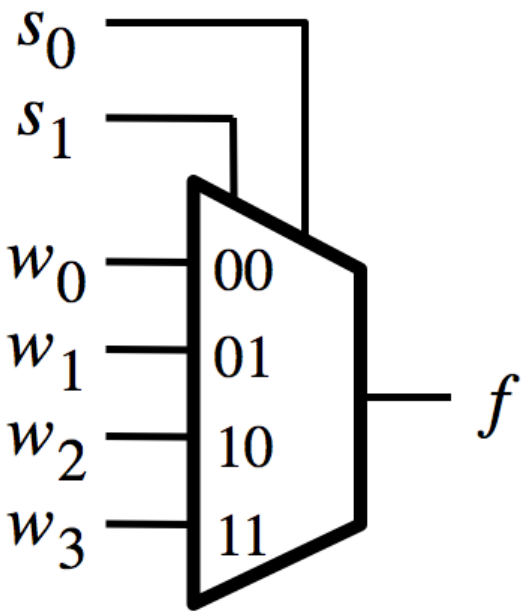
[Figure 6.52 from the textbook]

Multiplexers

4-1 Multiplexer (Definition)

- Has four inputs: w_0 , w_1, w_2, w_3
- Also has two select lines: s_1 and s_0
- If s_1=0 and s_0=0, then the output f is equal to w_0
- If s_1=0 and s_0=1, then the output f is equal to w_1
- If s_1=1 and s_0=0, then the output f is equal to w_2
- If s_1=1 and s_0=1, then the output f is equal to w_3

Graphical Symbol and Truth Table



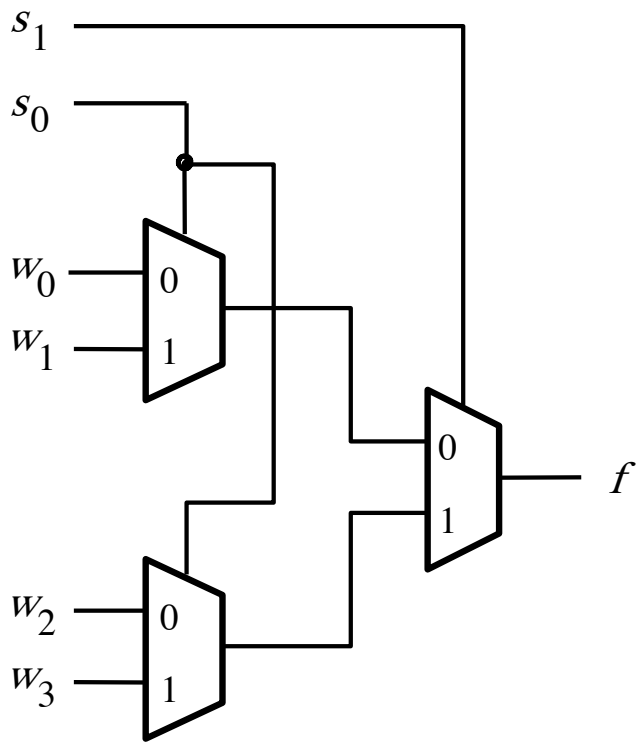
(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

[Figure 4.2a-b from the textbook]

Using three 2-to-1 multiplexers to build one 4-to-1 multiplexer



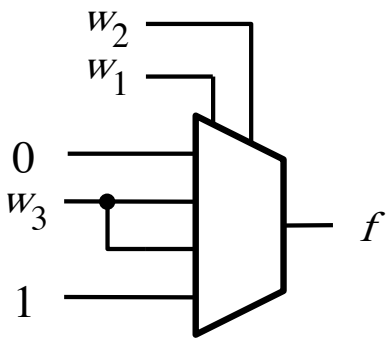
[Figure 4.3 from the textbook]

Implementation of a logic function

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

(a) Modified truth table



(b) Circuit

[Figure 4.7 from the textbook]

Implementation of 3-input XOR with a 4-to-1 Multiplexer

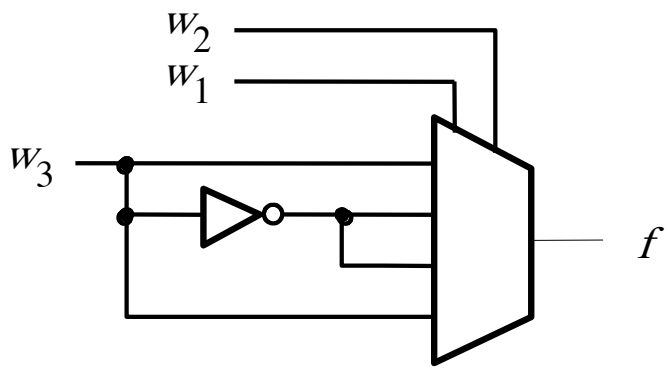
w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	$\bar{w}_3$
0	1	1	0	
1	0	0	1	$\bar{w}_3$
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	

[Figure 4.9a from the textbook]

Implementation of 3-input XOR with a 4-to-1 Multiplexer

w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	$\bar{w}_3$
0	1	1	0	
1	0	0	1	$\bar{w}_3$
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	

(a) Truth table

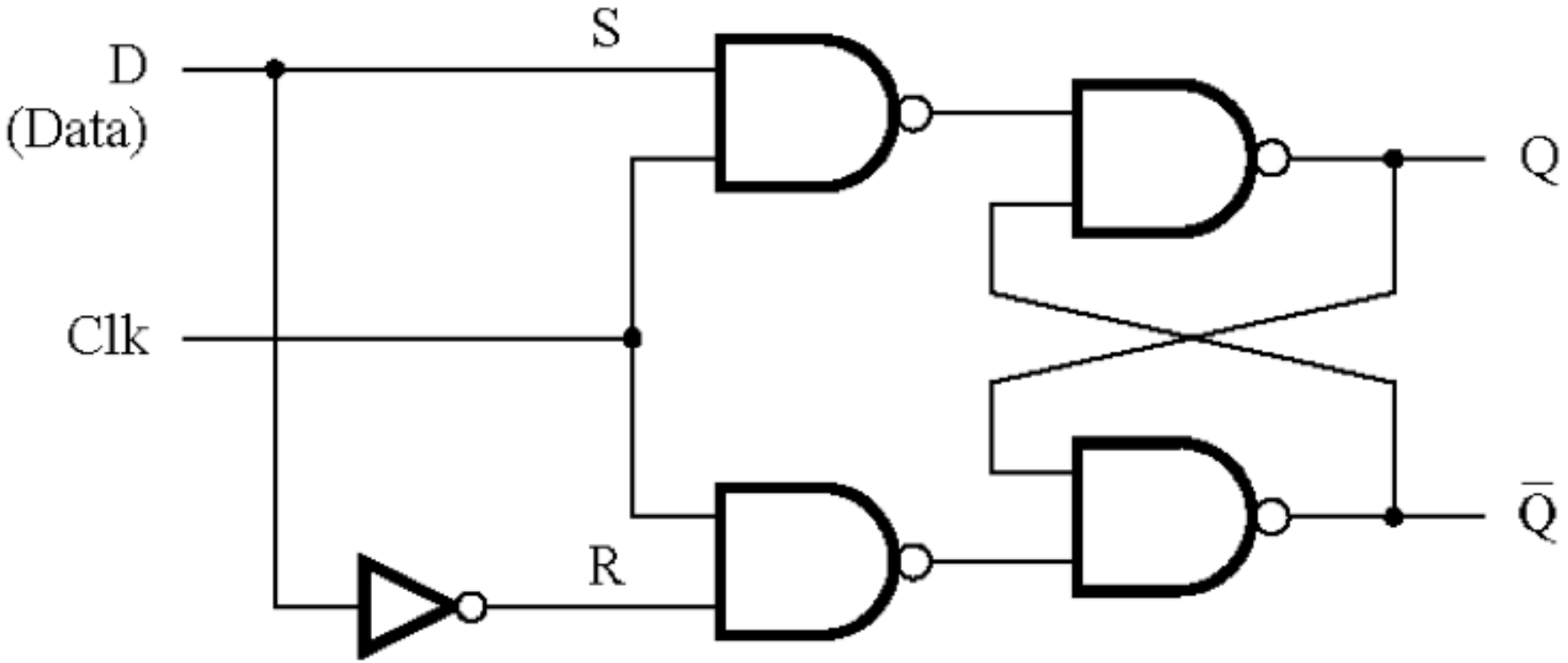


(b) Circuit

[Figure 4.9 from the textbook]

Gated D Latch

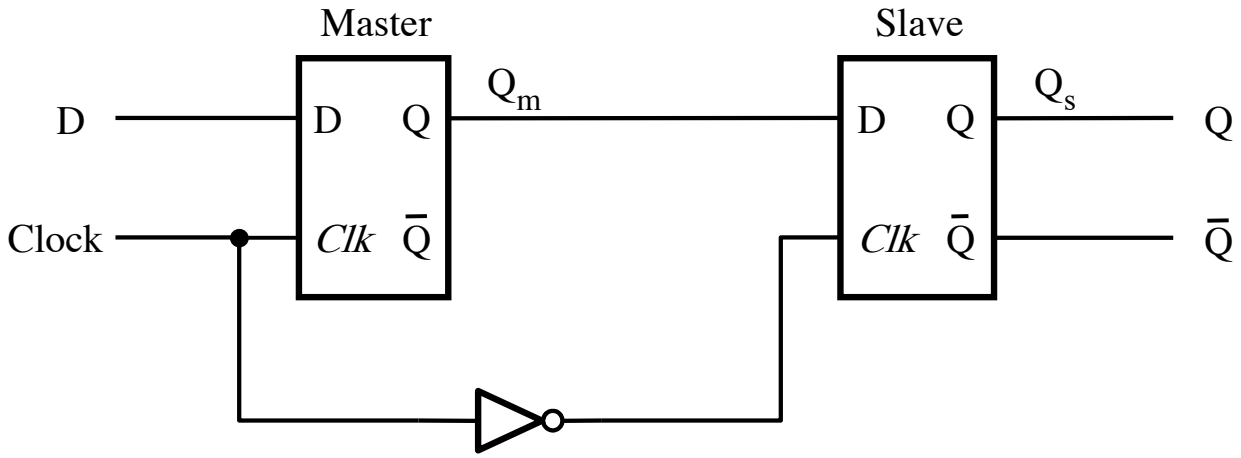
Circuit Diagram for the Gated D Latch



[Figure 5.7a from the textbook]

Edge-Triggered D Flip-Flops

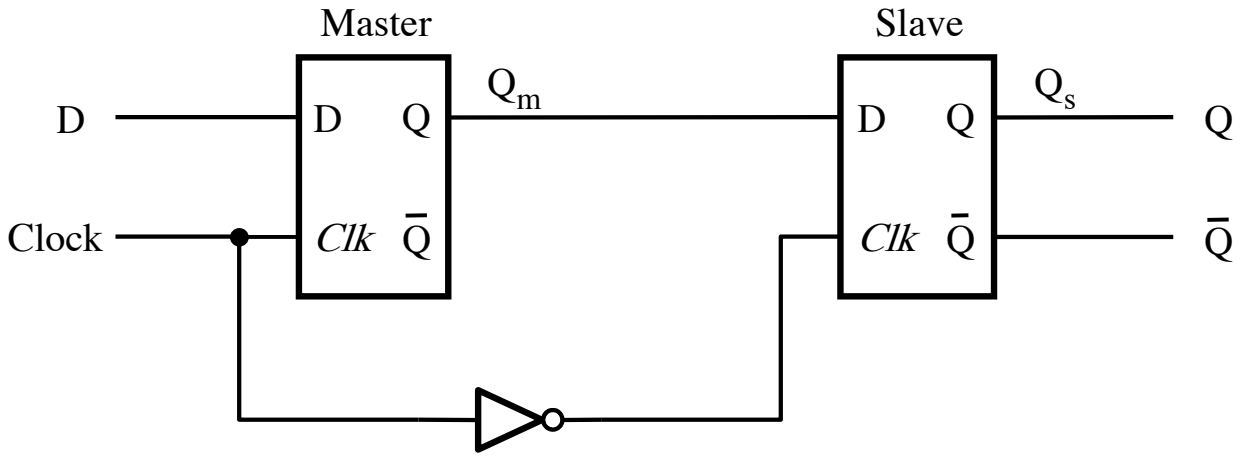
Master-Slave D Flip-Flop



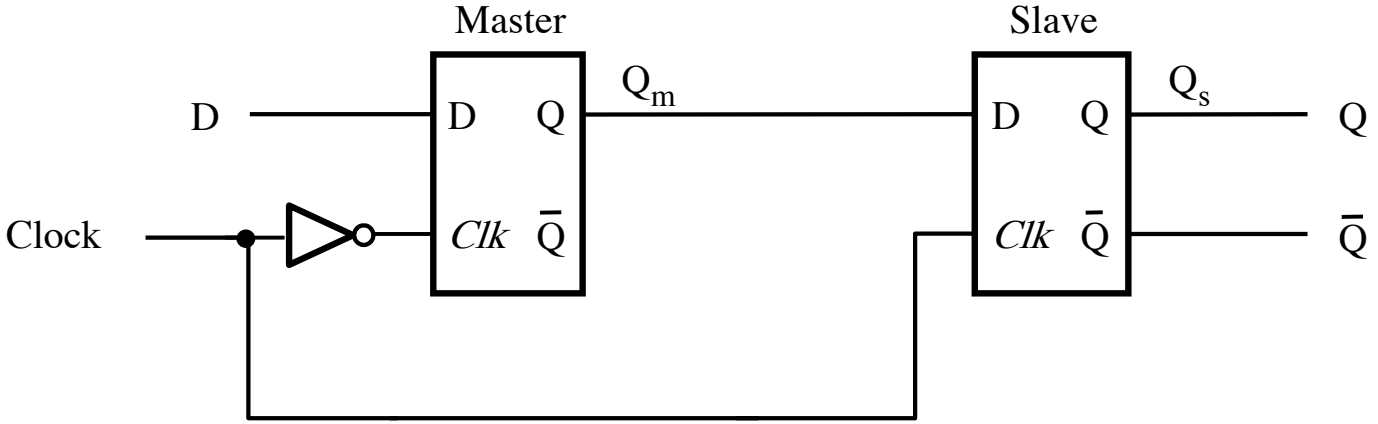
(a) Circuit

[Figure 5.9a from the textbook]

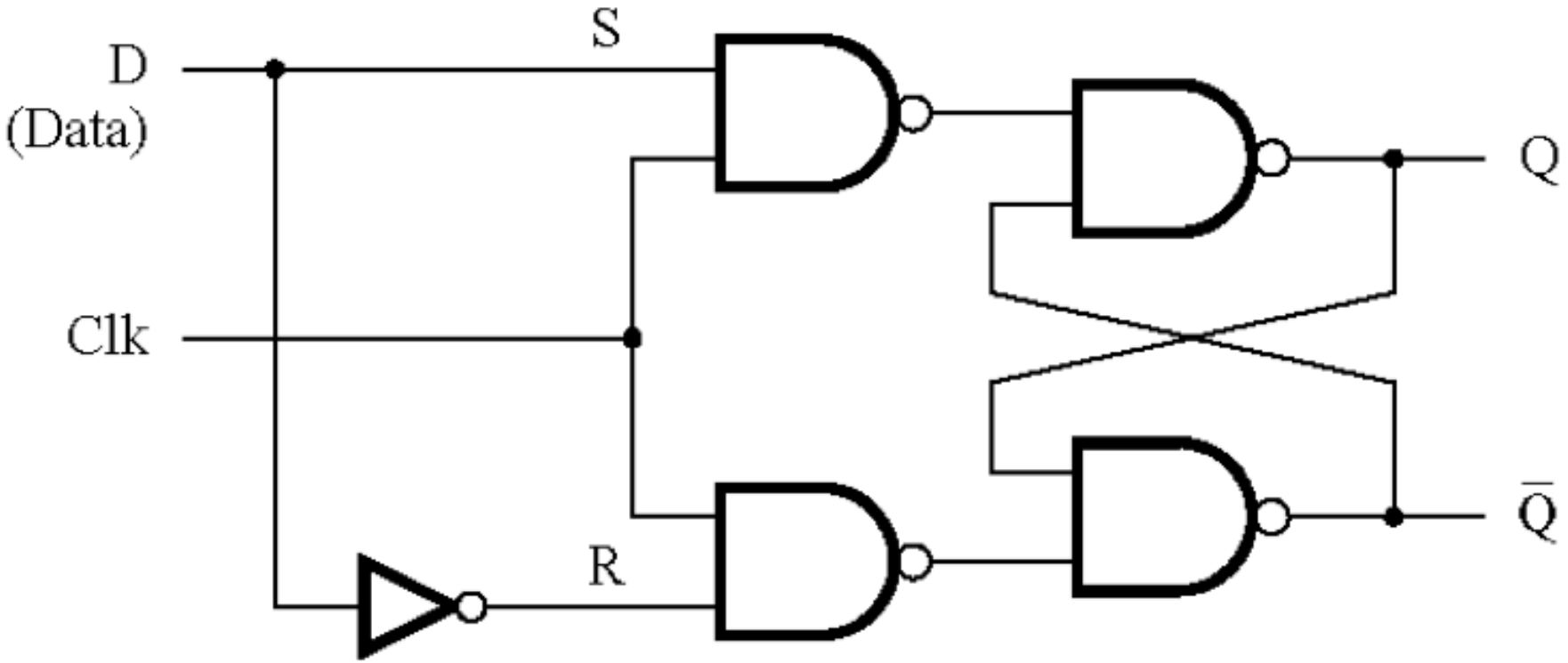
Negative-Edge-Triggered Master-Slave D Flip-Flop



Positive-Edge-Triggered Master-Slave D Flip-Flop

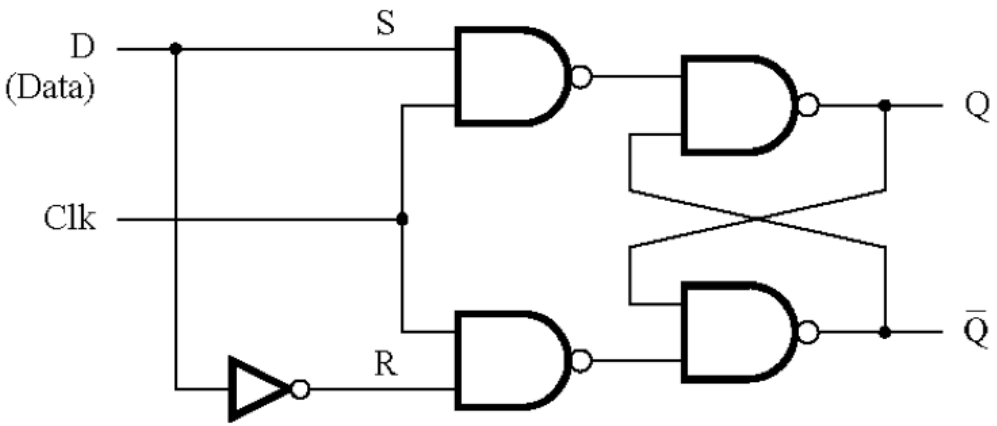


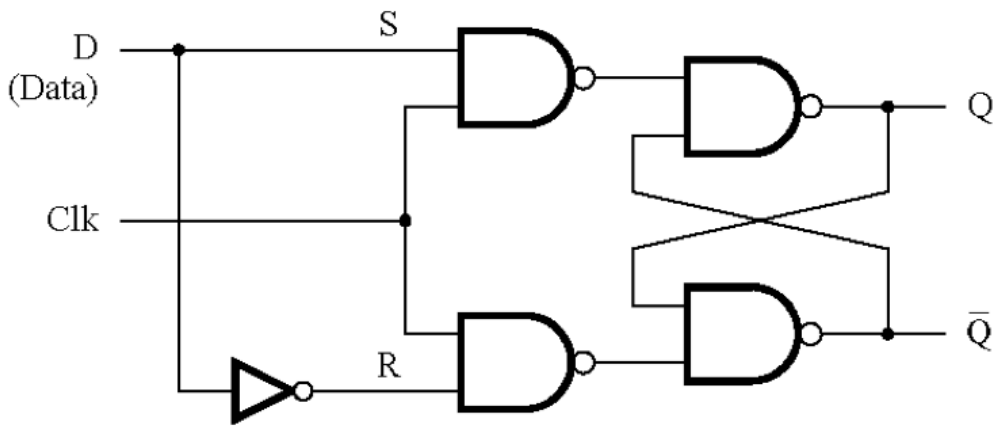
Circuit Diagram for the Gated D Latch



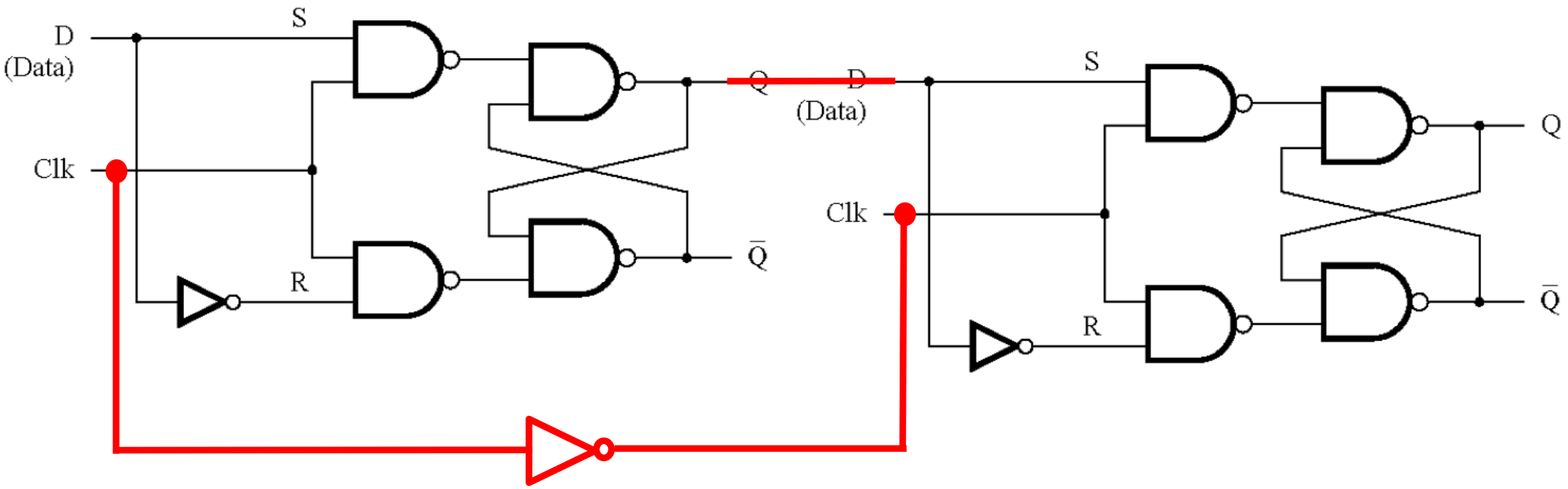
[Figure 5.7a from the textbook]

Constructing a D Flip-Flop

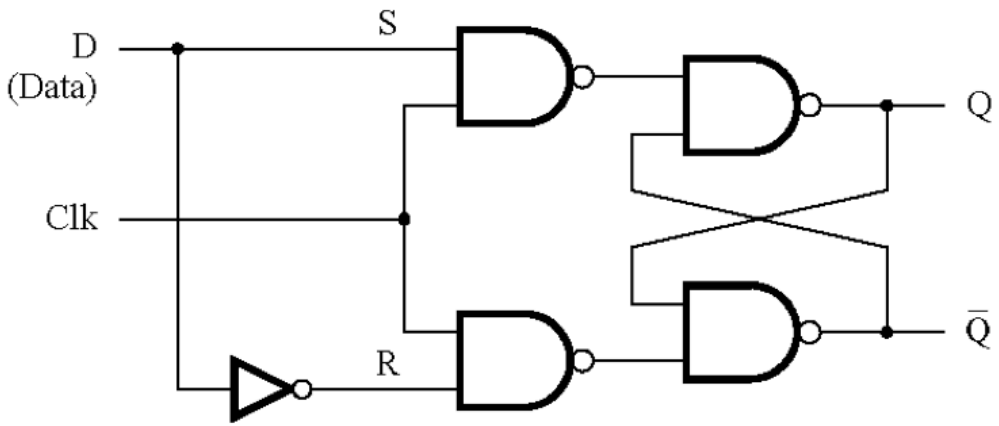


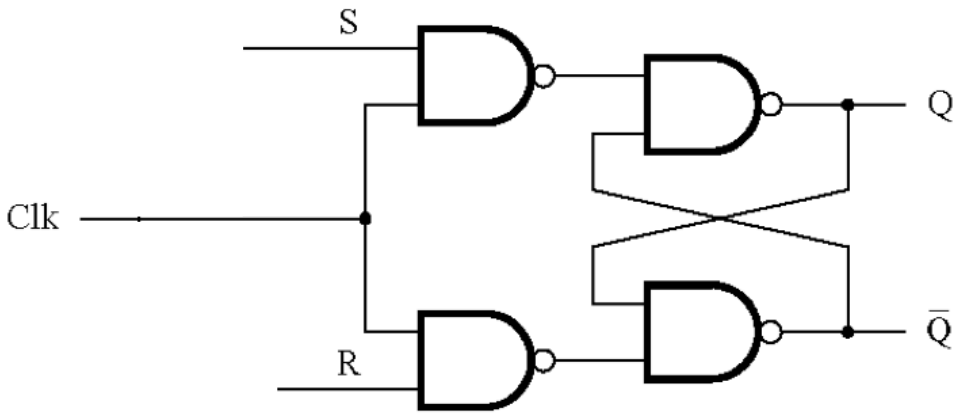


Constructing a D Flip-Flop

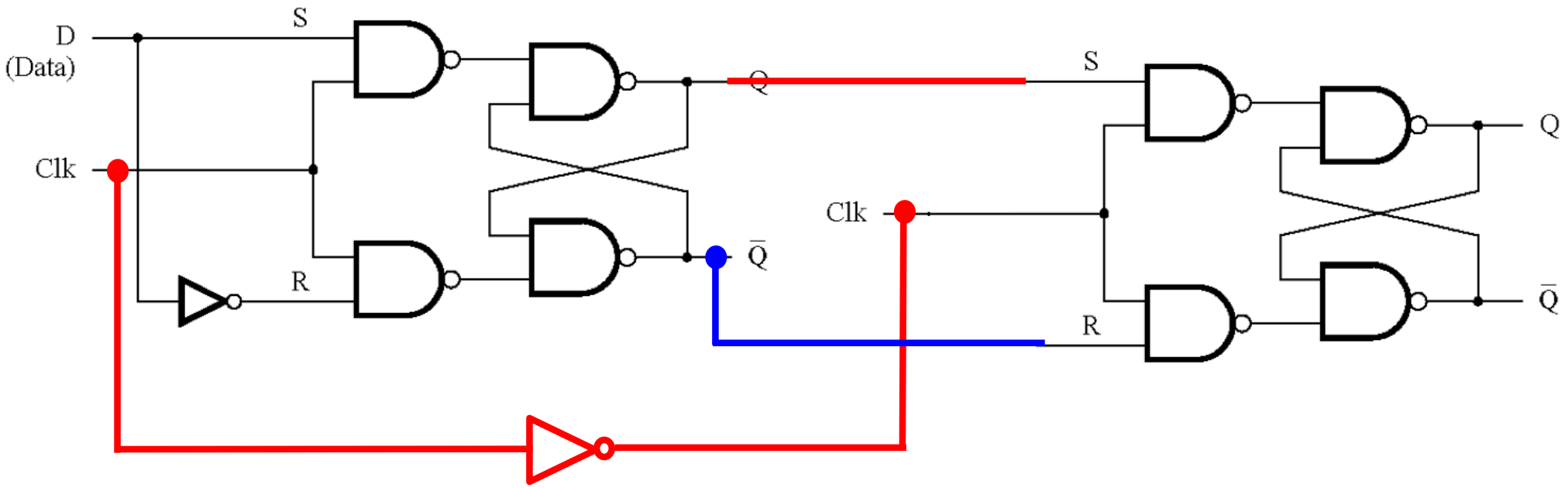


Constructing a D Flip-Flop (with one less NOT gate)



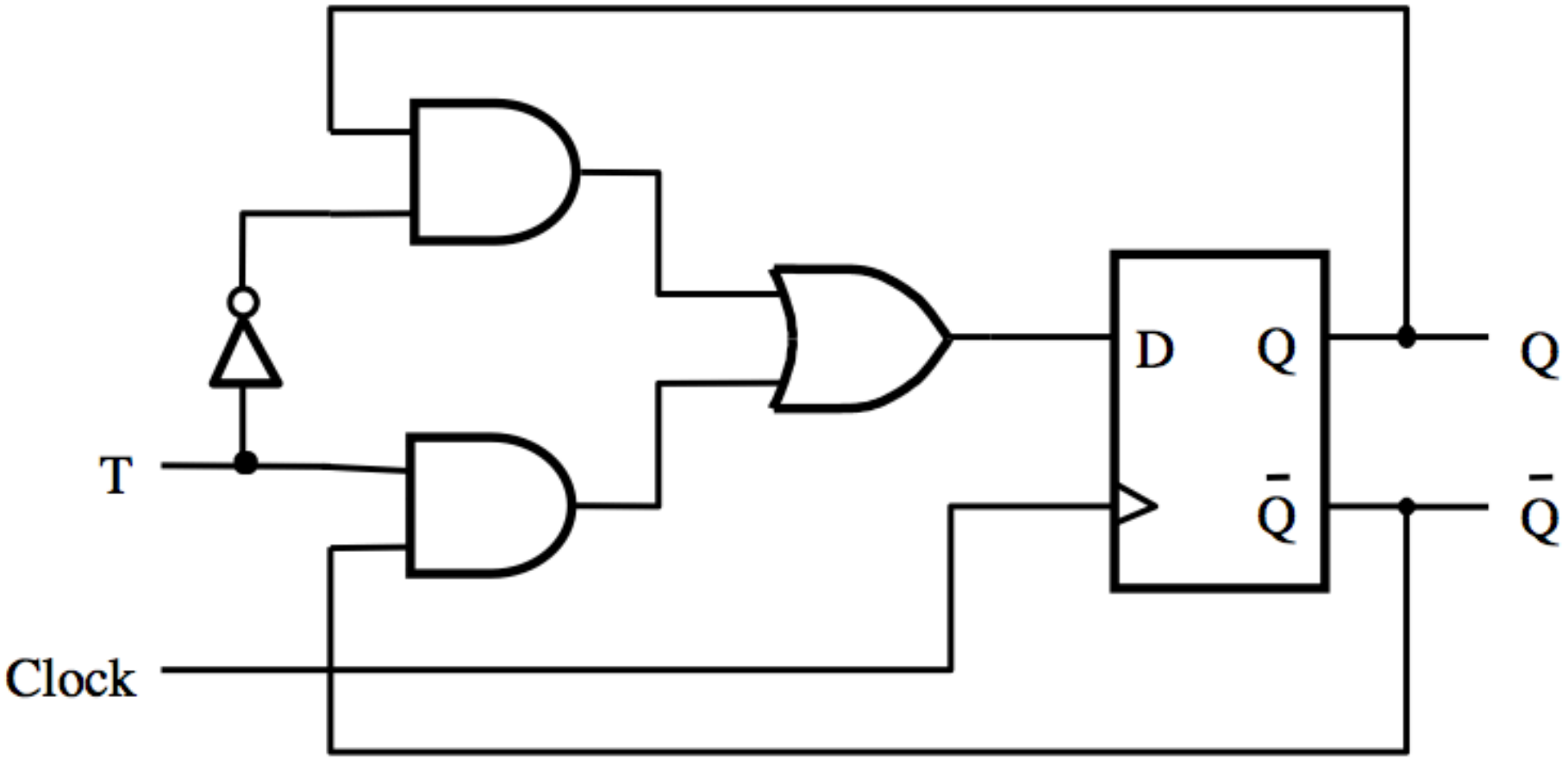


Constructing a D Flip-Flop (with one less NOT gate)



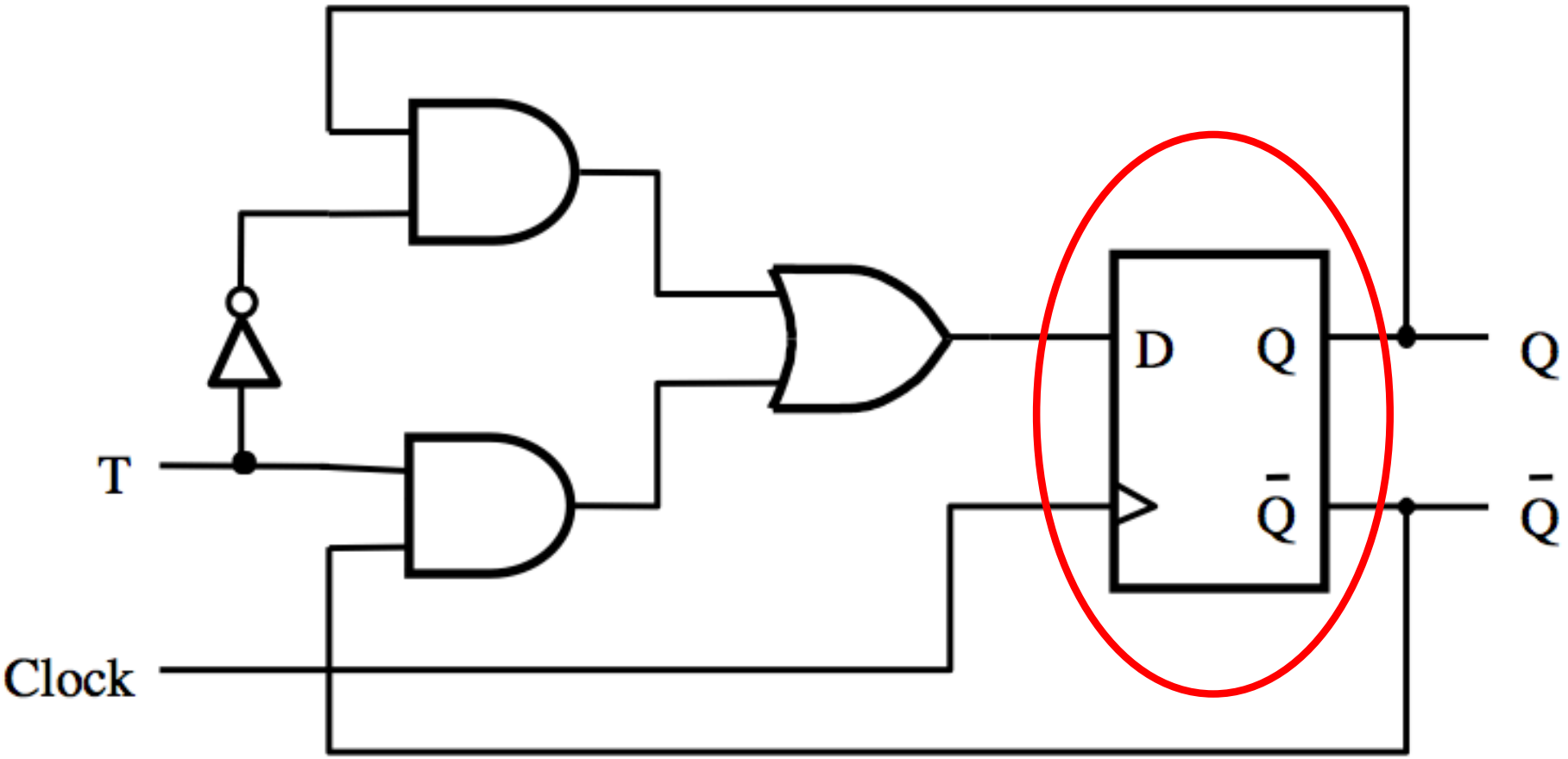
T Flip-Flop

T Flip-Flop



[Figure 5.15a from the textbook]

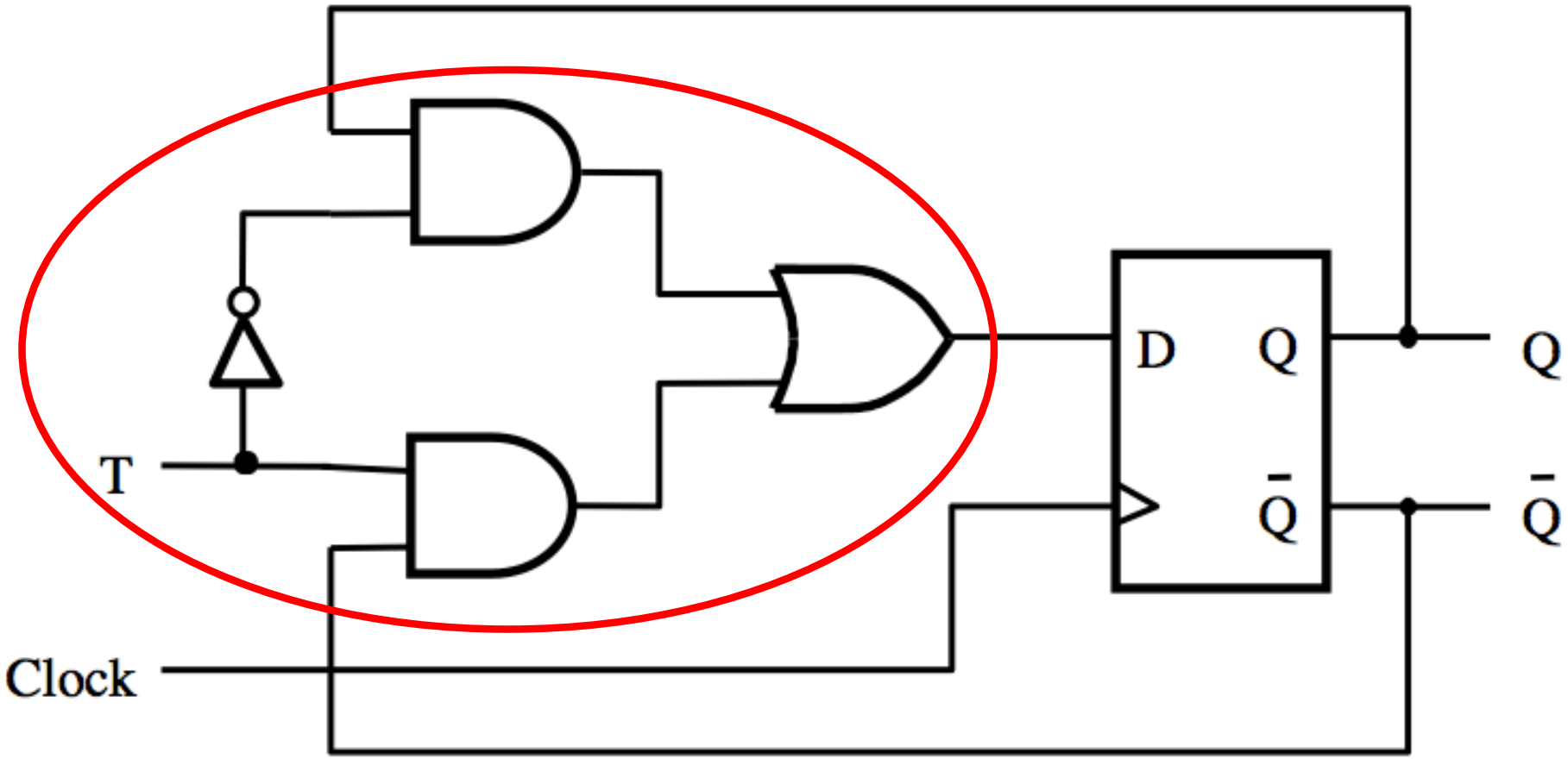
T Flip-Flop



Positive-edge-triggered
D Flip-Flop

[Figure 5.15a from the textbook]

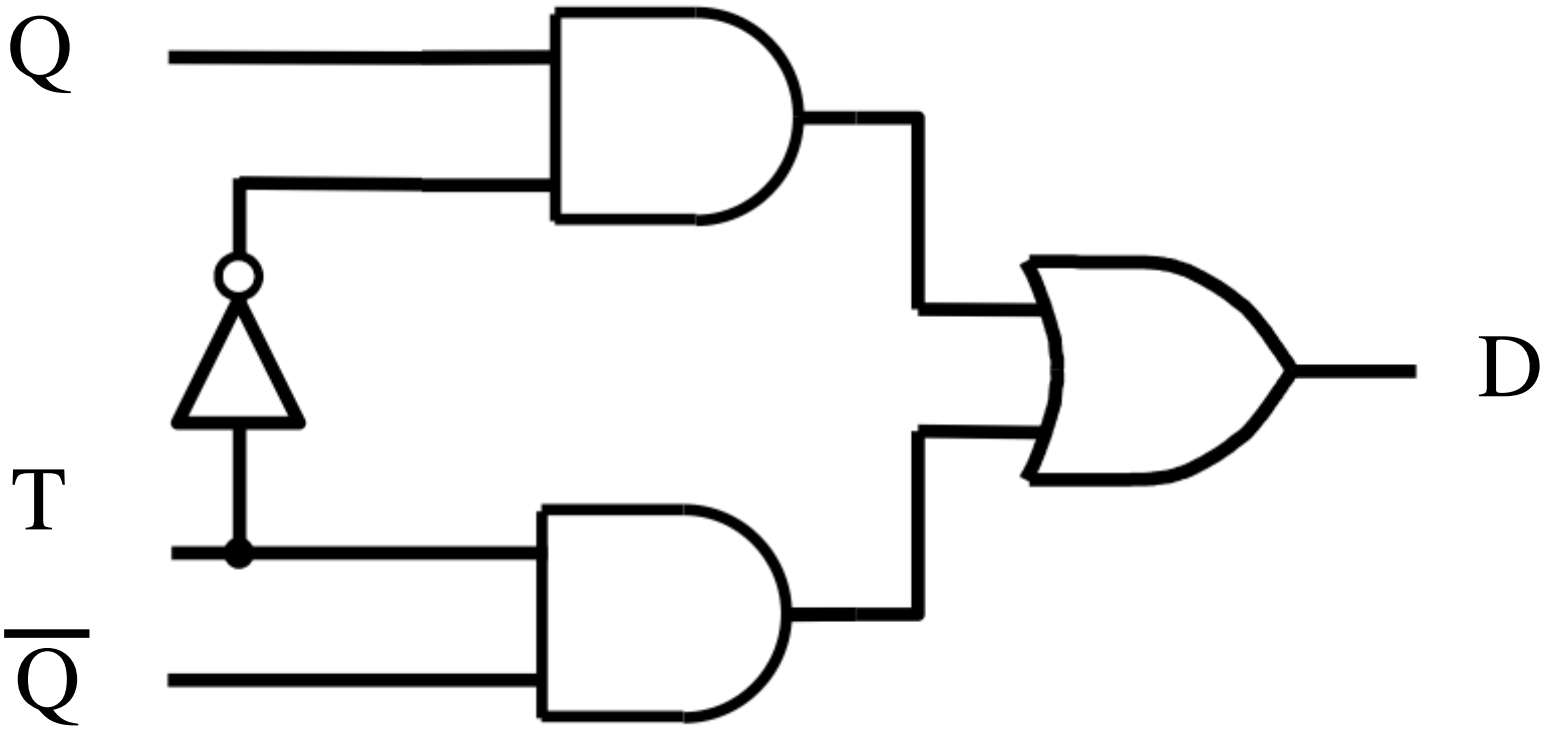
T Flip-Flop



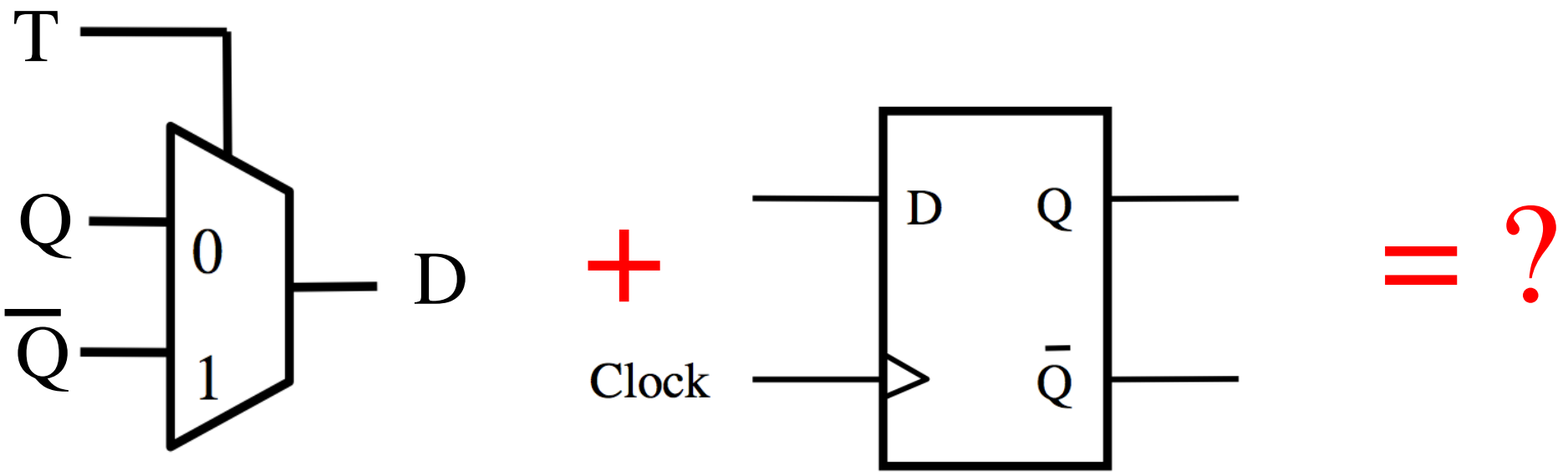
What is this?

[Figure 5.15a from the textbook]

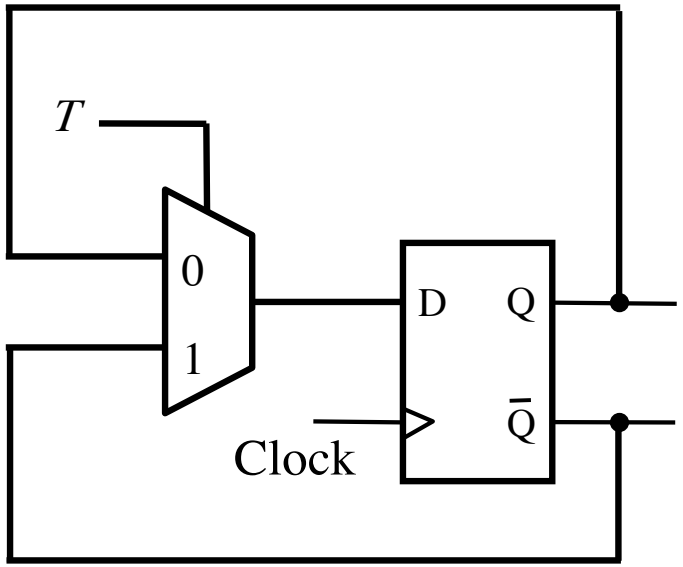
What is this?



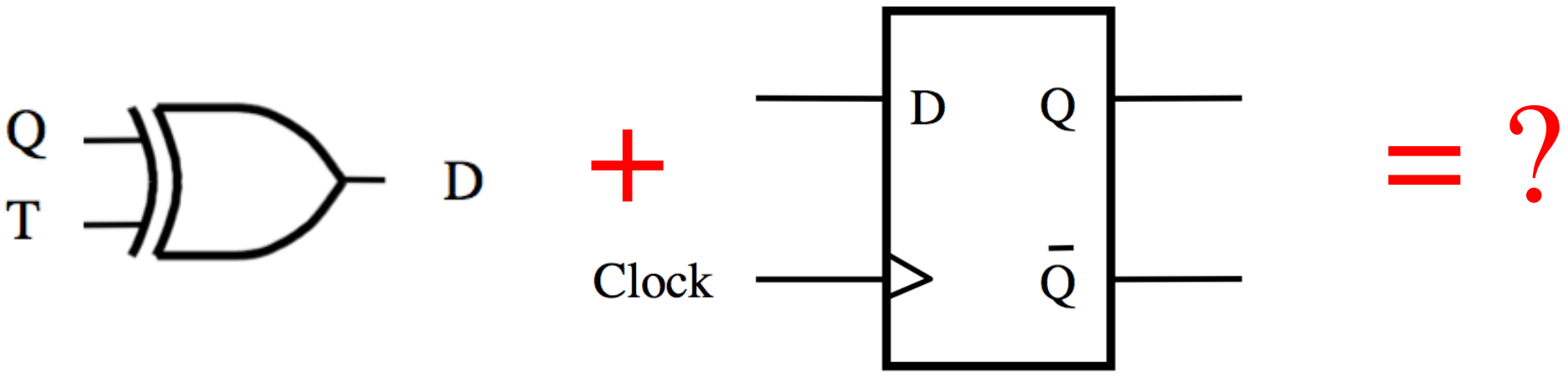
What is this?



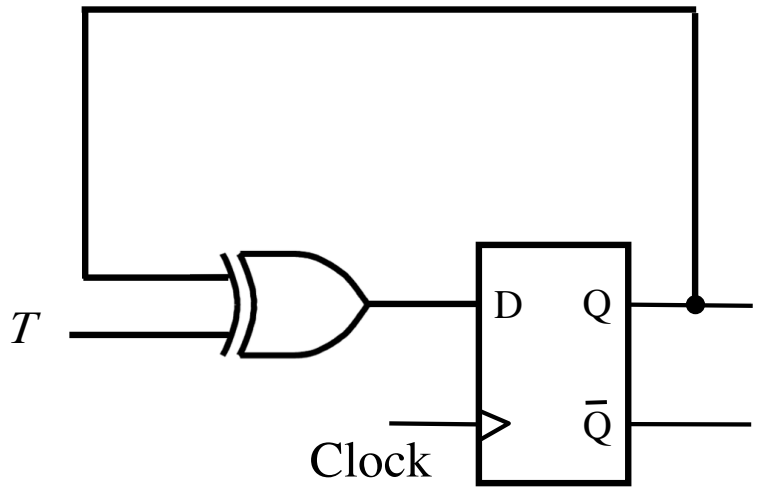
T Flip-Flop



What is this?

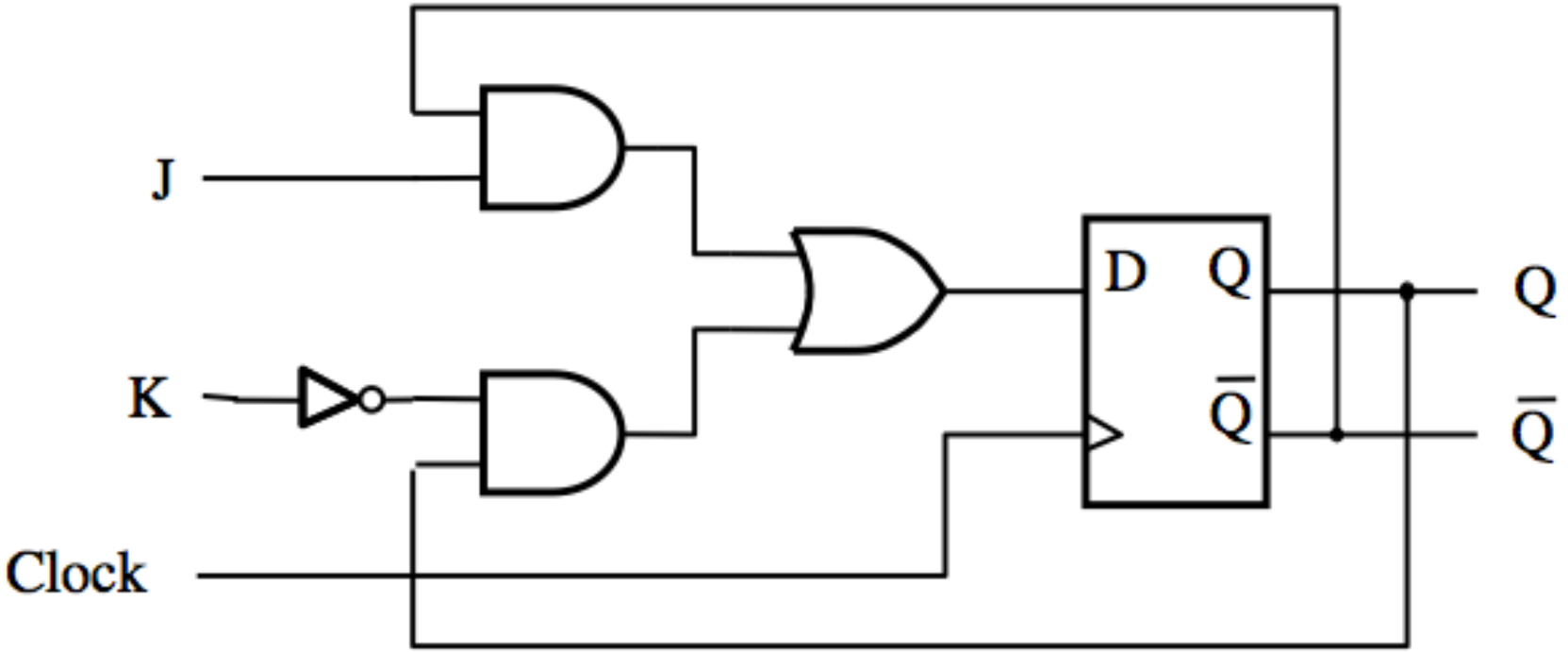


T Flip-Flop



JK Flip-Flop

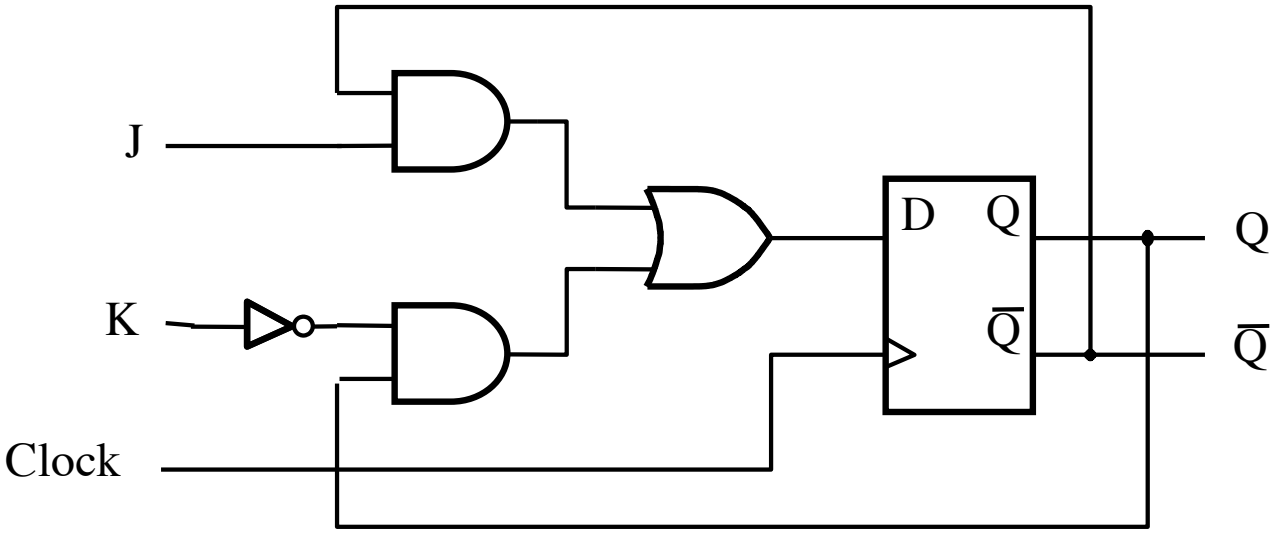
JK Flip-Flop



$$D = J\overline{Q} + \overline{K}Q$$

[Figure 5.16a from the textbook]

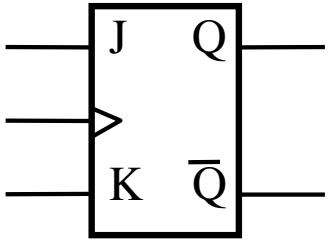
JK Flip-Flop



(a) Circuit

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Truth table



(c) Graphical symbol

[Figure 5.16 from the textbook]

JK Flip-Flop
(How it Works)

A versatile circuit that can be used both as a SR flip-flop and as a T flip flop

If J=0 and S =0 it stays in the same state

Just like SR It can be set and reset

J=S and K=R

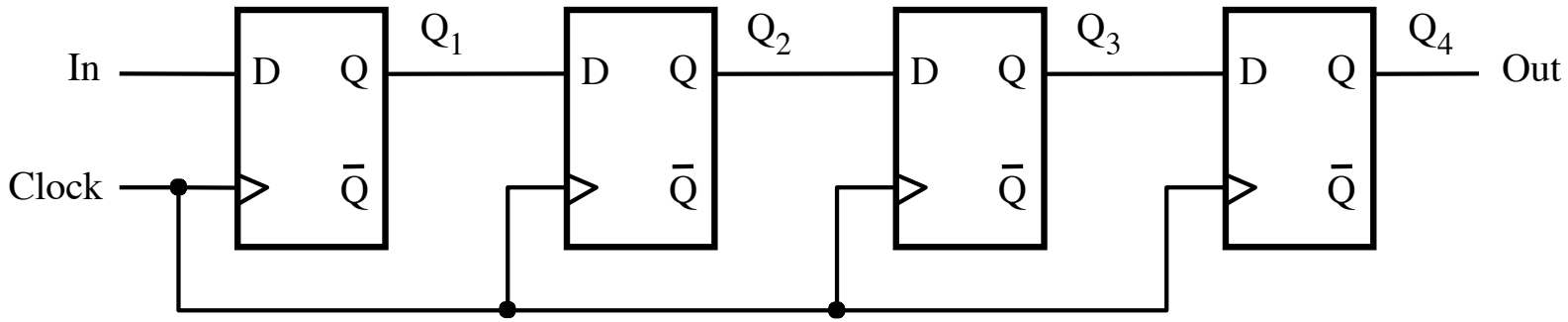
If J=K then it behaves as a T flip-flop

Registers

Register
(Definition)

An n-bit structure consisting of flip-flops

A simple shift register



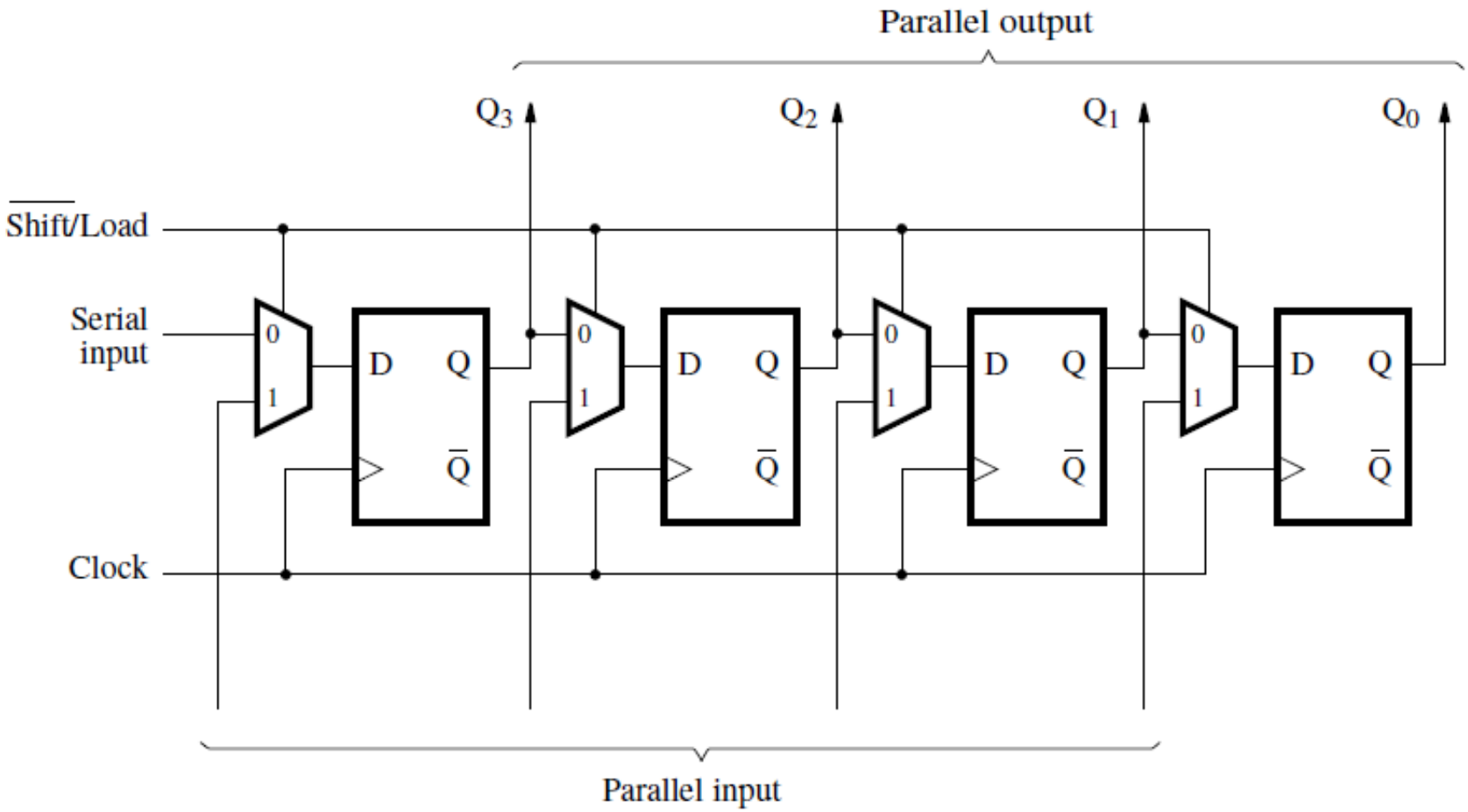
(a) Circuit

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) A sample sequence

[Figure 5.17 from the textbook]

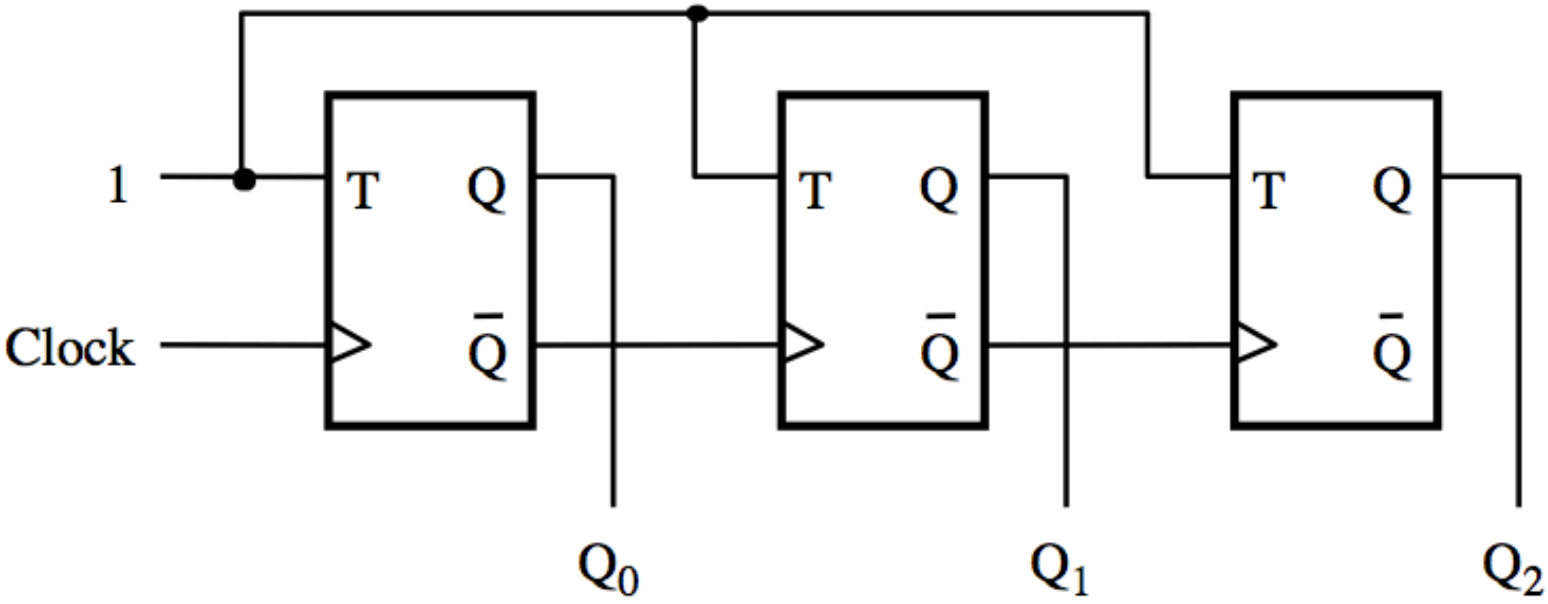
Parallel-access shift register



[Figure 5.18 from the textbook]

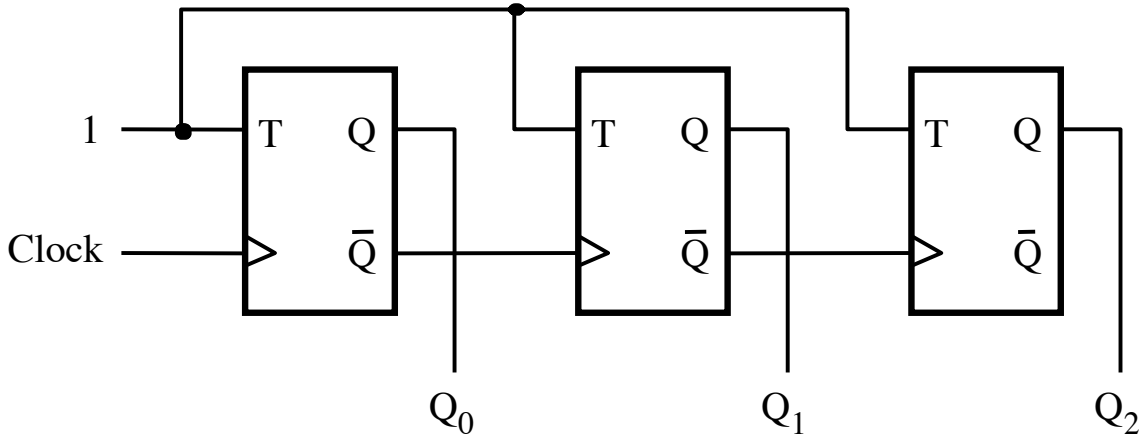
Counters

A three-bit up-counter

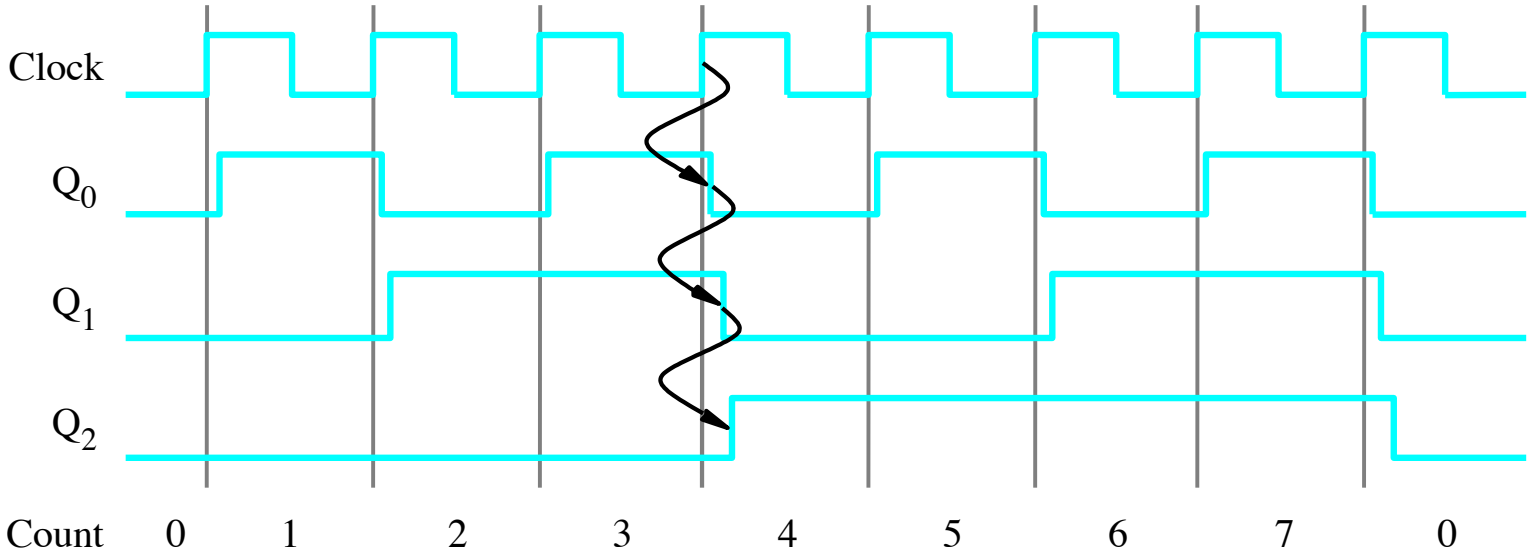


[Figure 5.19 from the textbook]

A three-bit up-counter



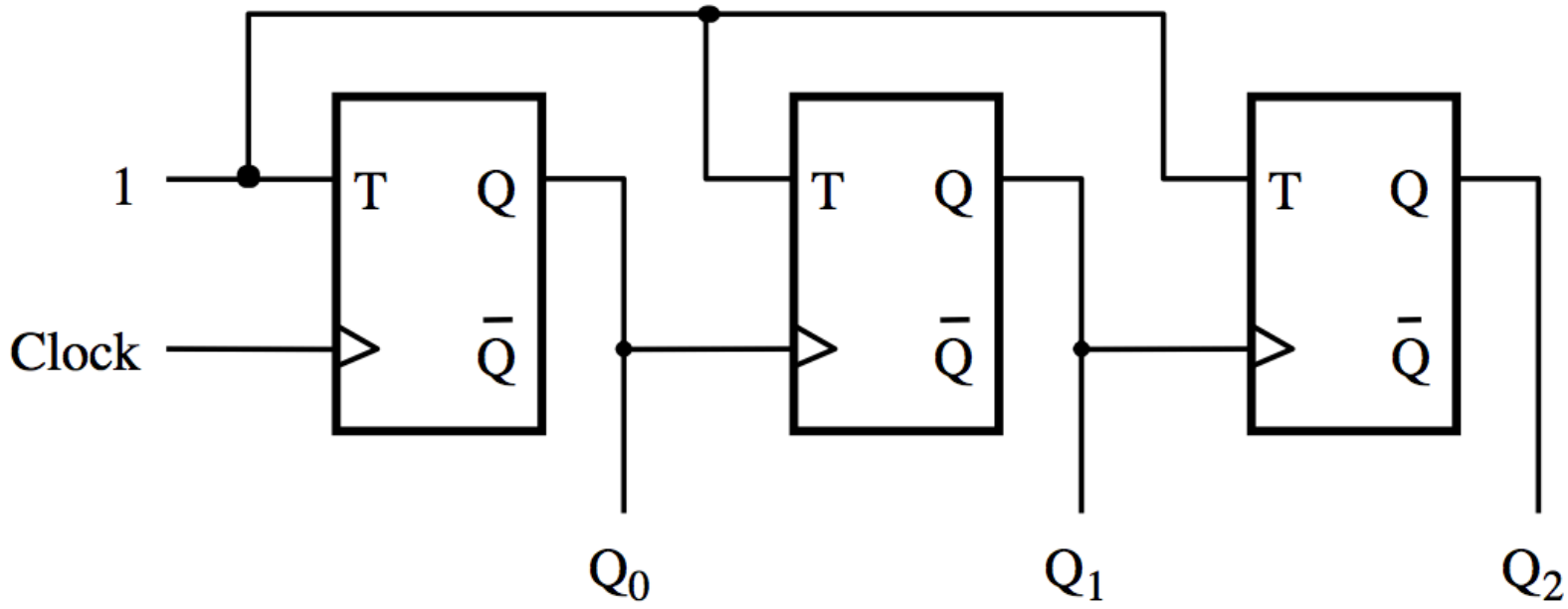
(a) Circuit



(b) Timing diagram

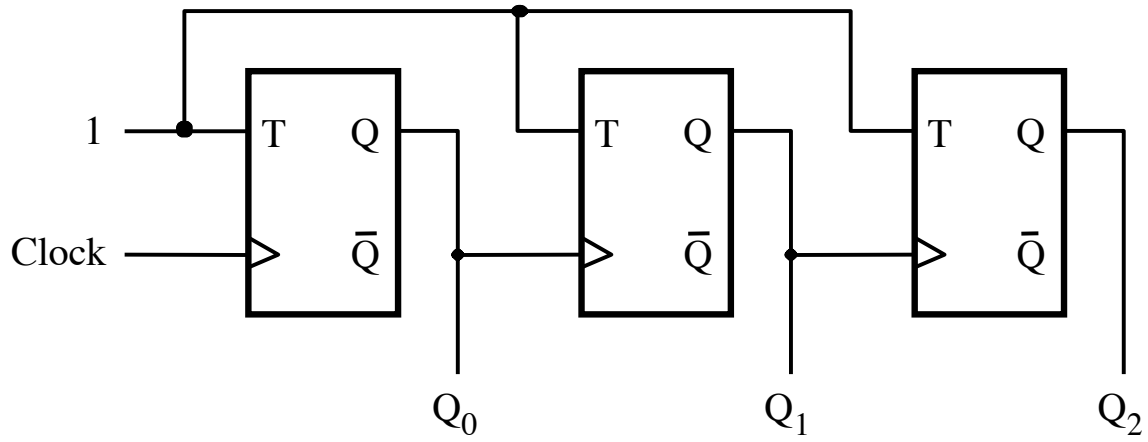
[Figure 5.19 from the textbook]

A three-bit down-counter

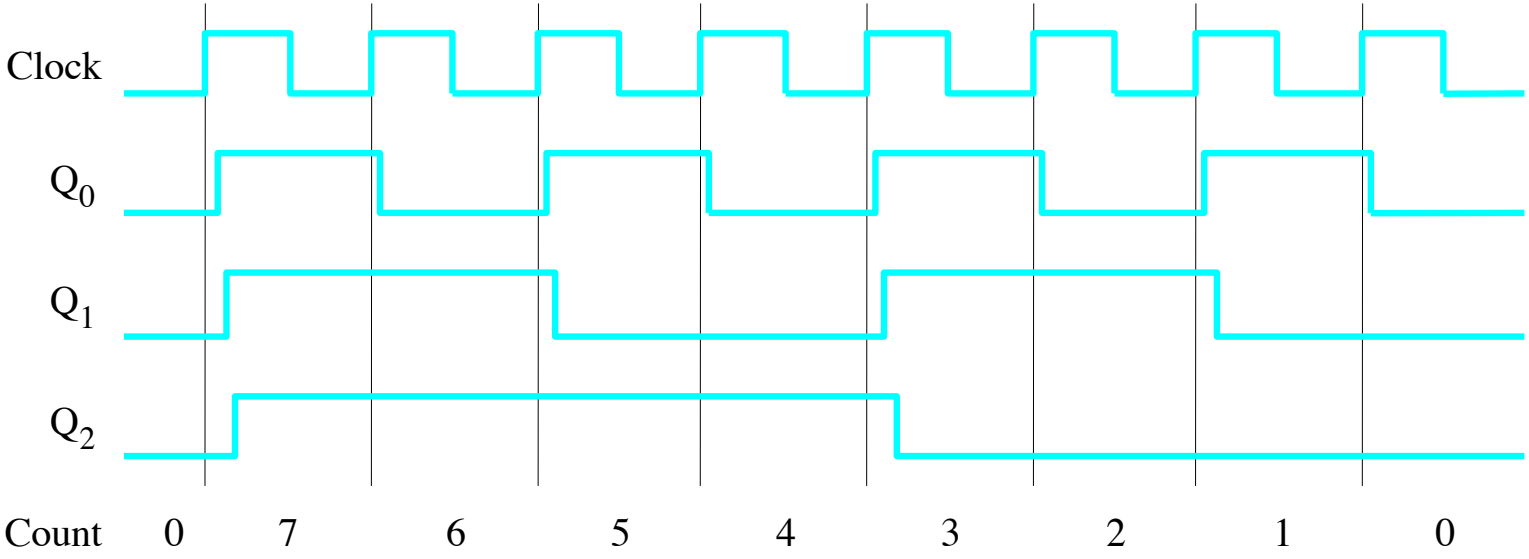


[Figure 5.20 from the textbook]

A three-bit down-counter



(a) Circuit

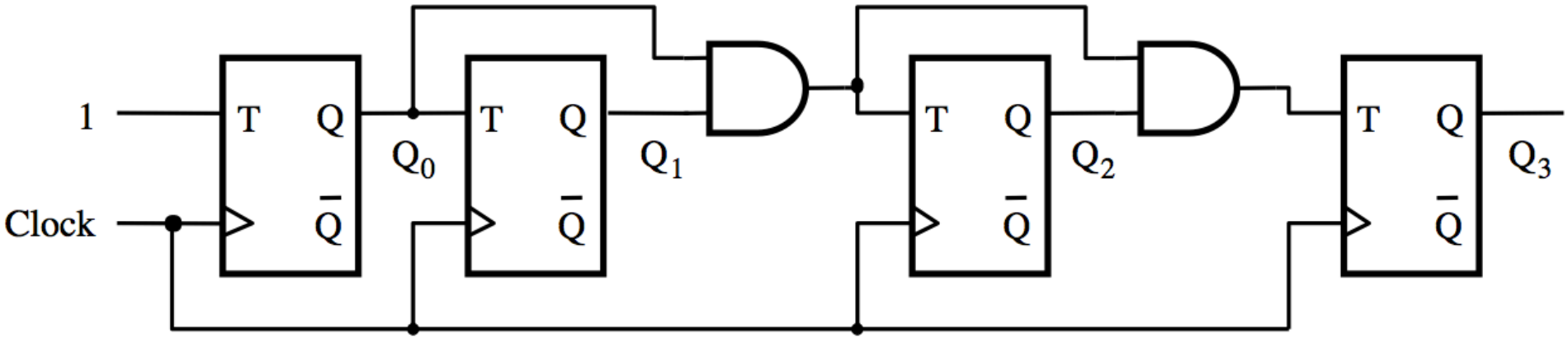


(b) Timing diagram

[Figure 5.20 from the textbook]

Synchronous Counters

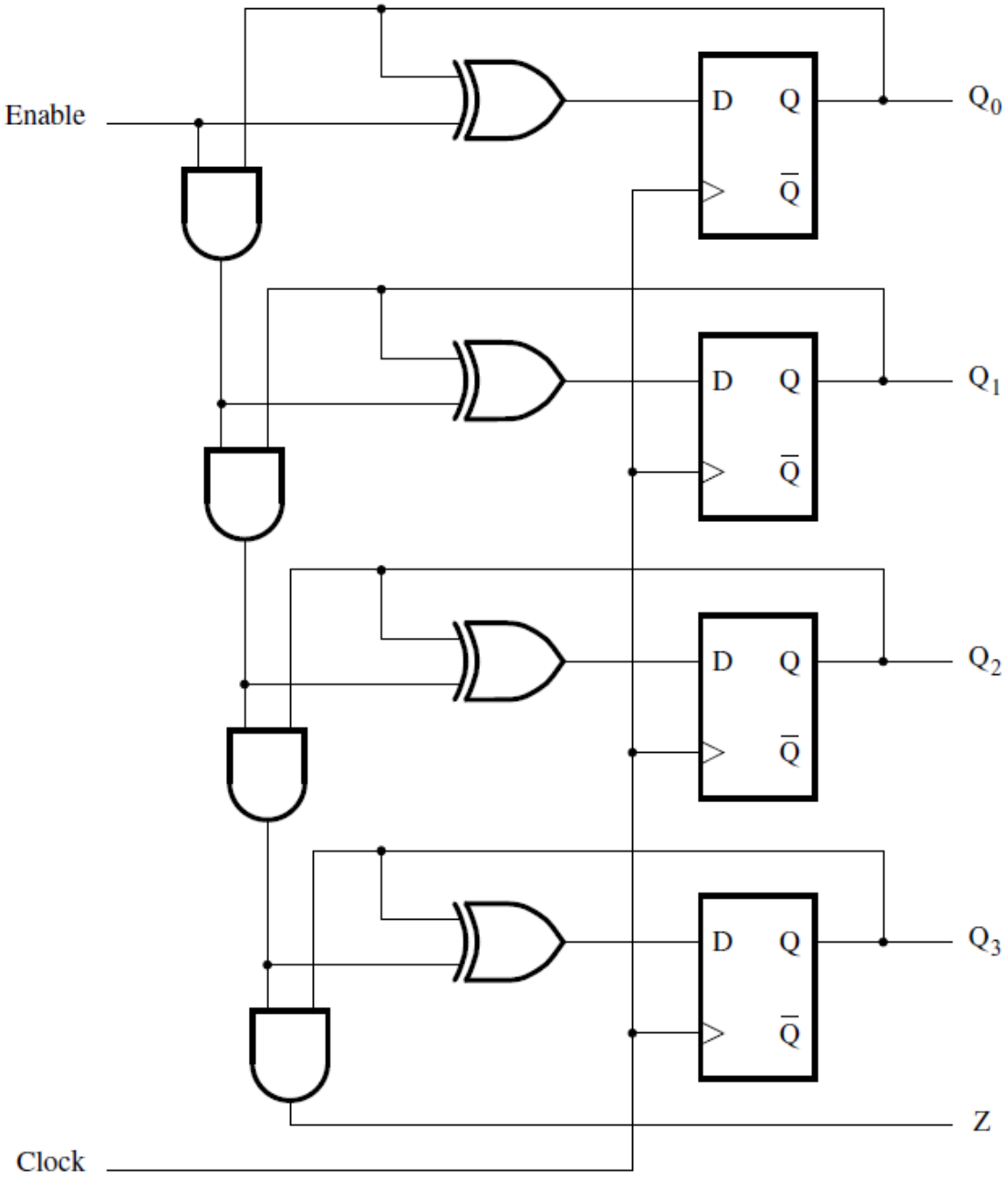
A four-bit synchronous up-counter



[Figure 5.21 from the textbook]

Synchronous Counter with D Flip-Flops

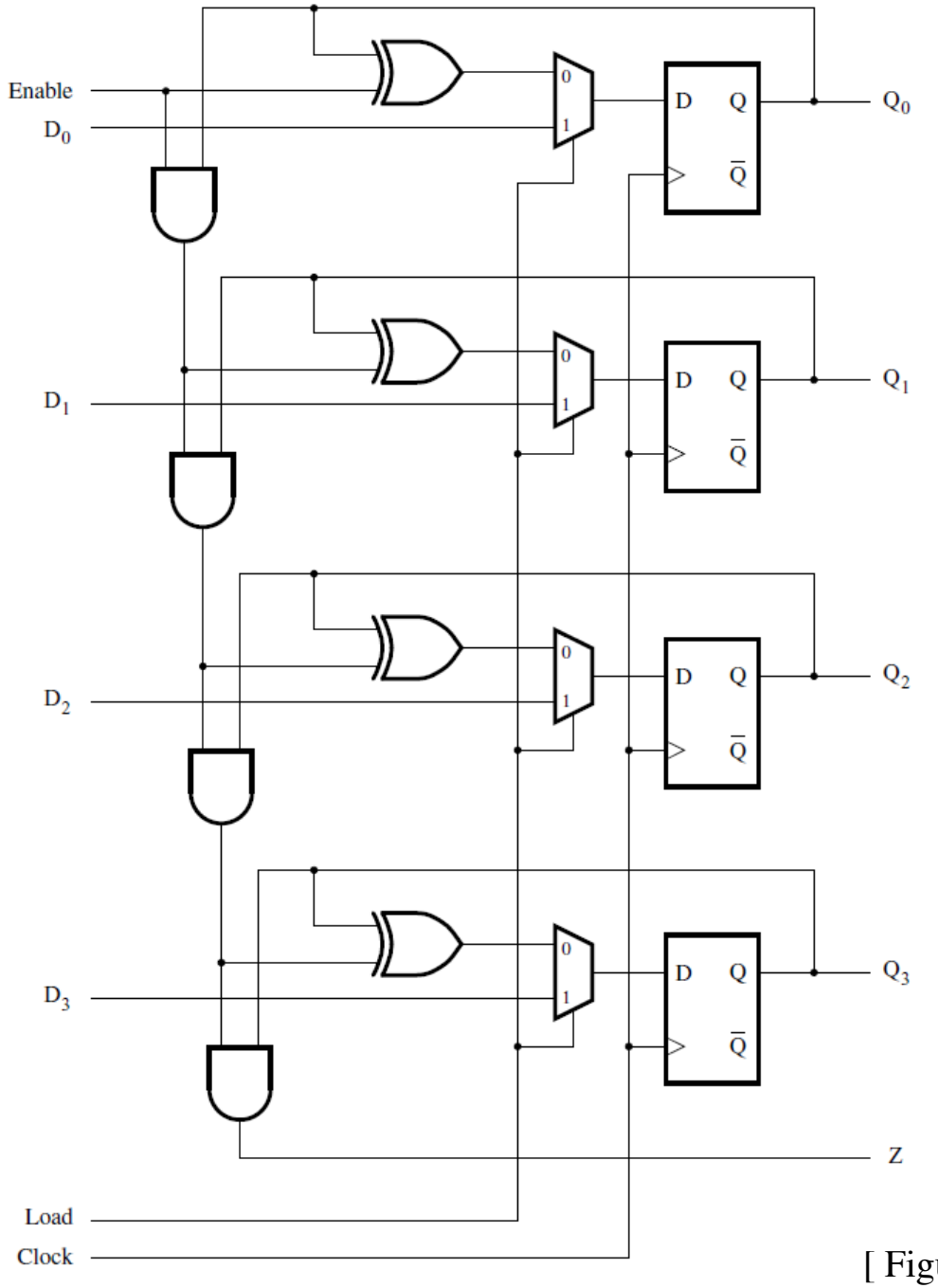
A four-bit counter with D flip-flops



[Figure 5.23 from the textbook]

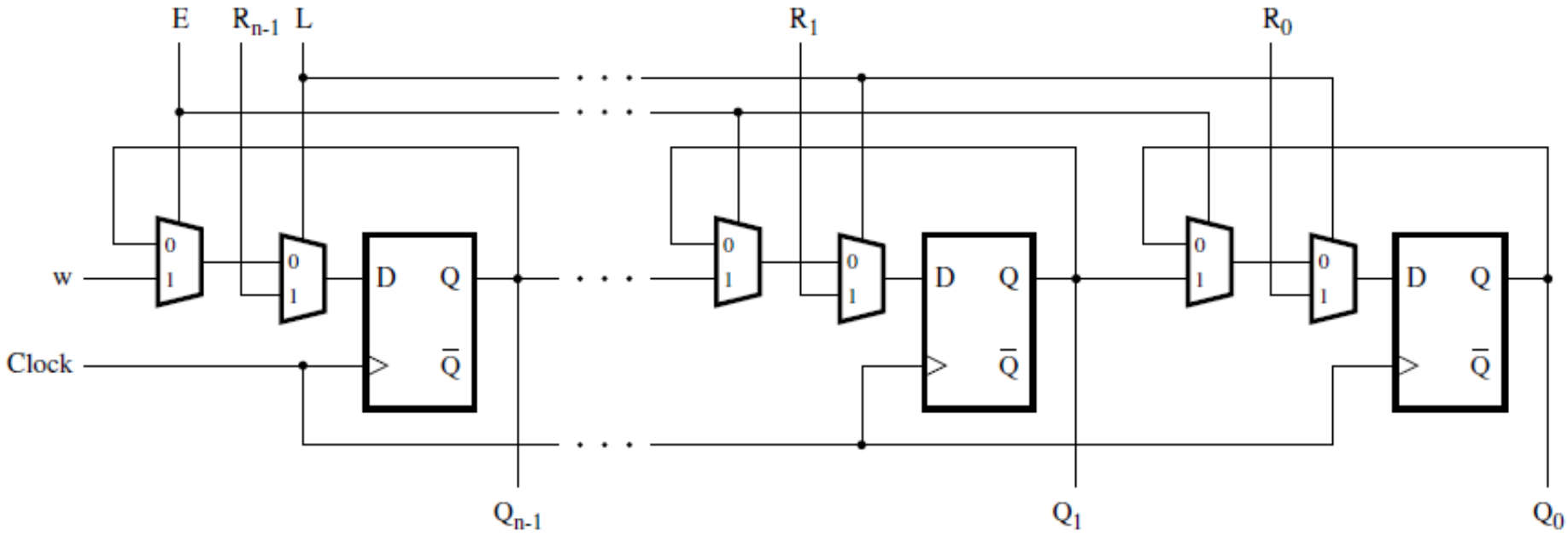
Counters with Parallel Load

A counter with parallel-load capability



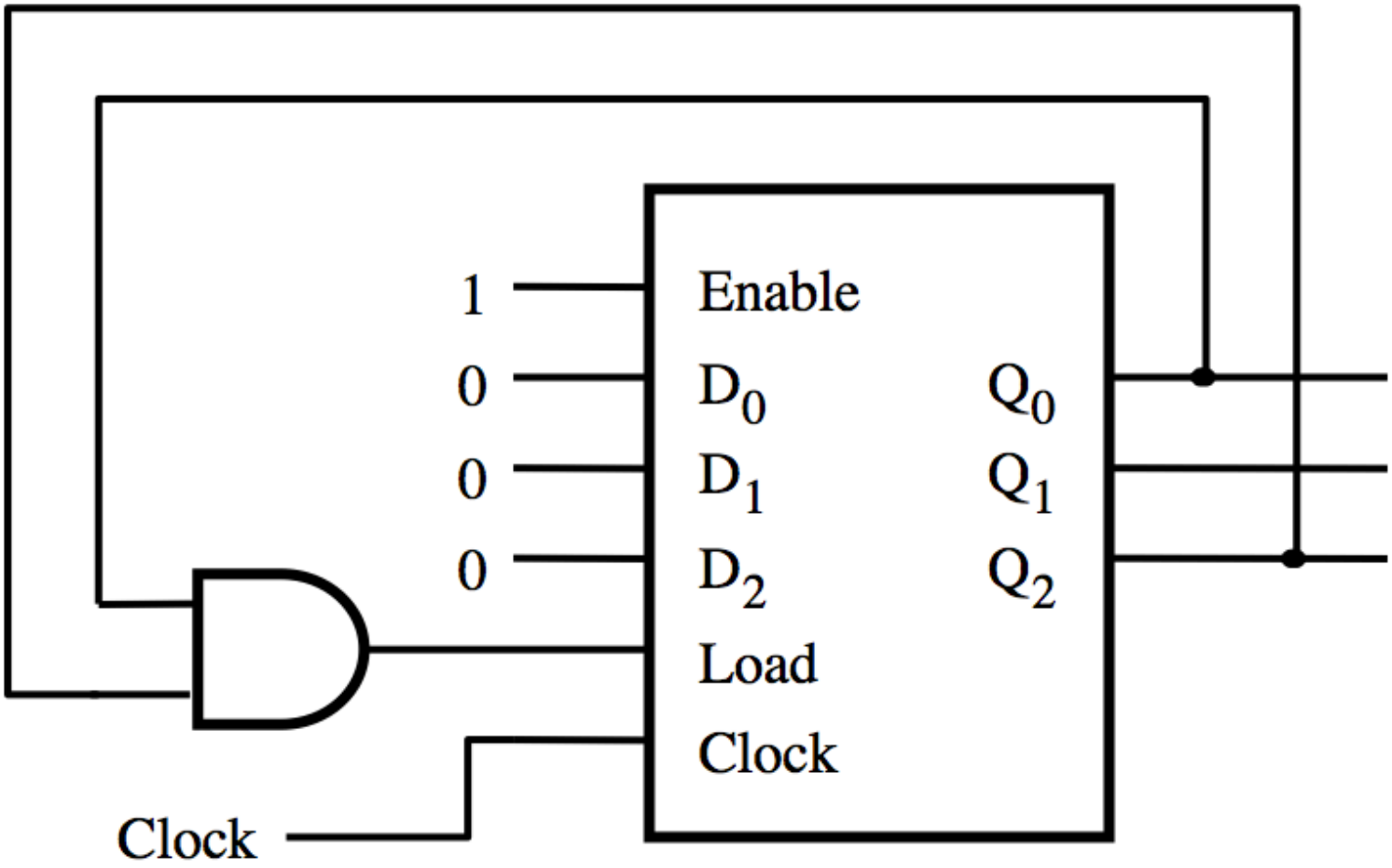
[Figure 5.24 from the textbook]

A shift register with parallel load and enable control inputs



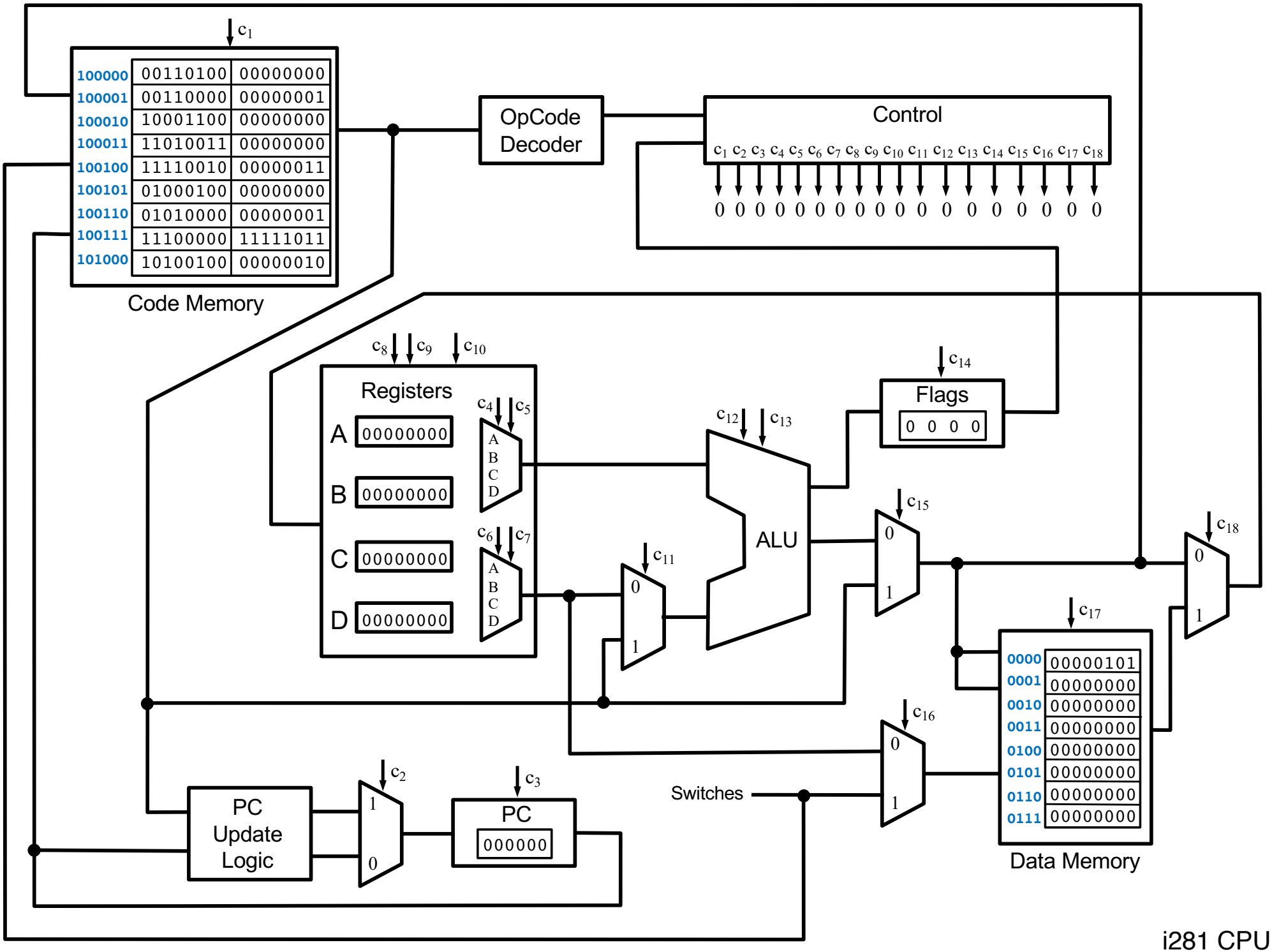
[Figure 5.59 from the textbook]

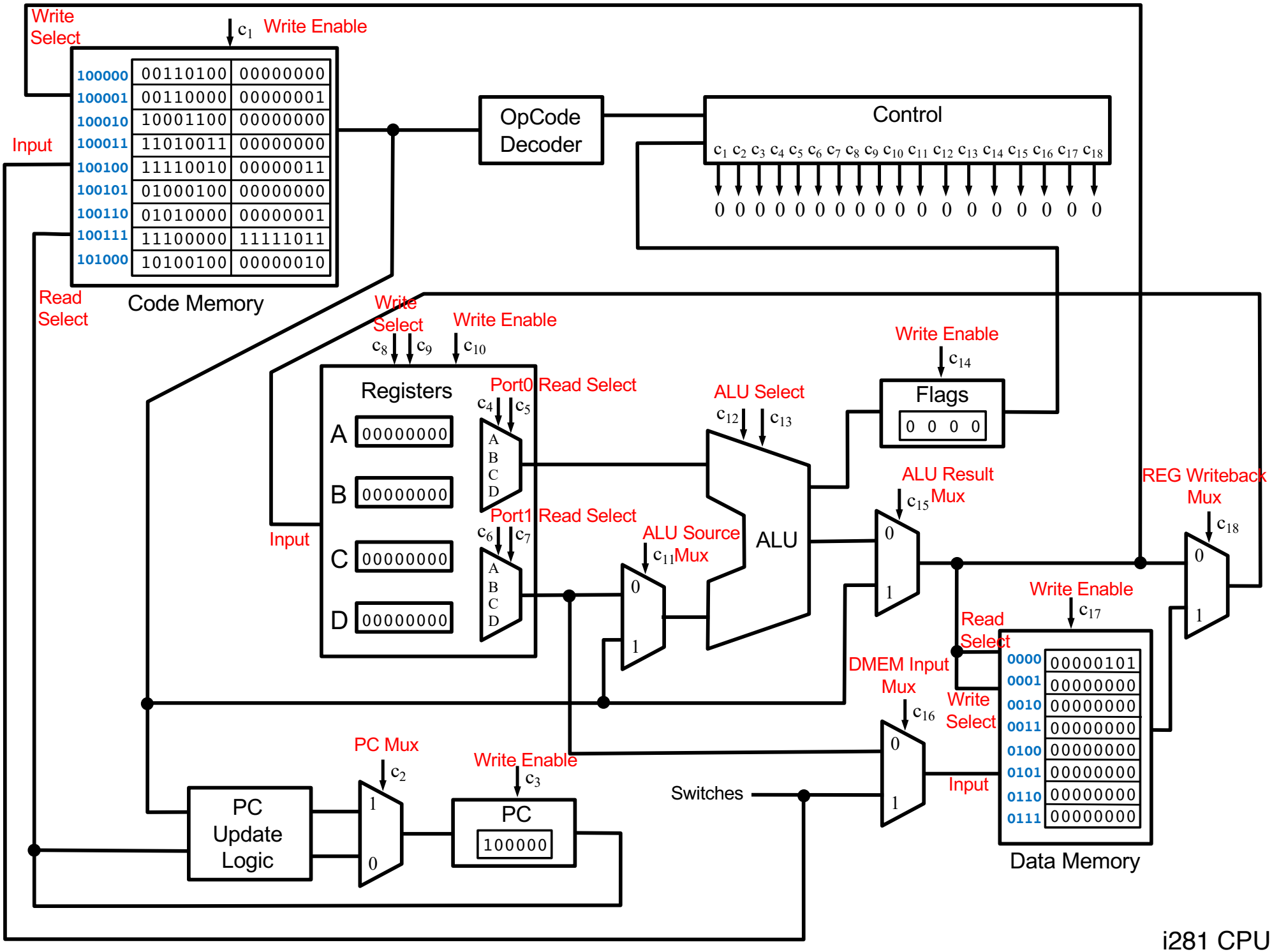
What does this circuit do?

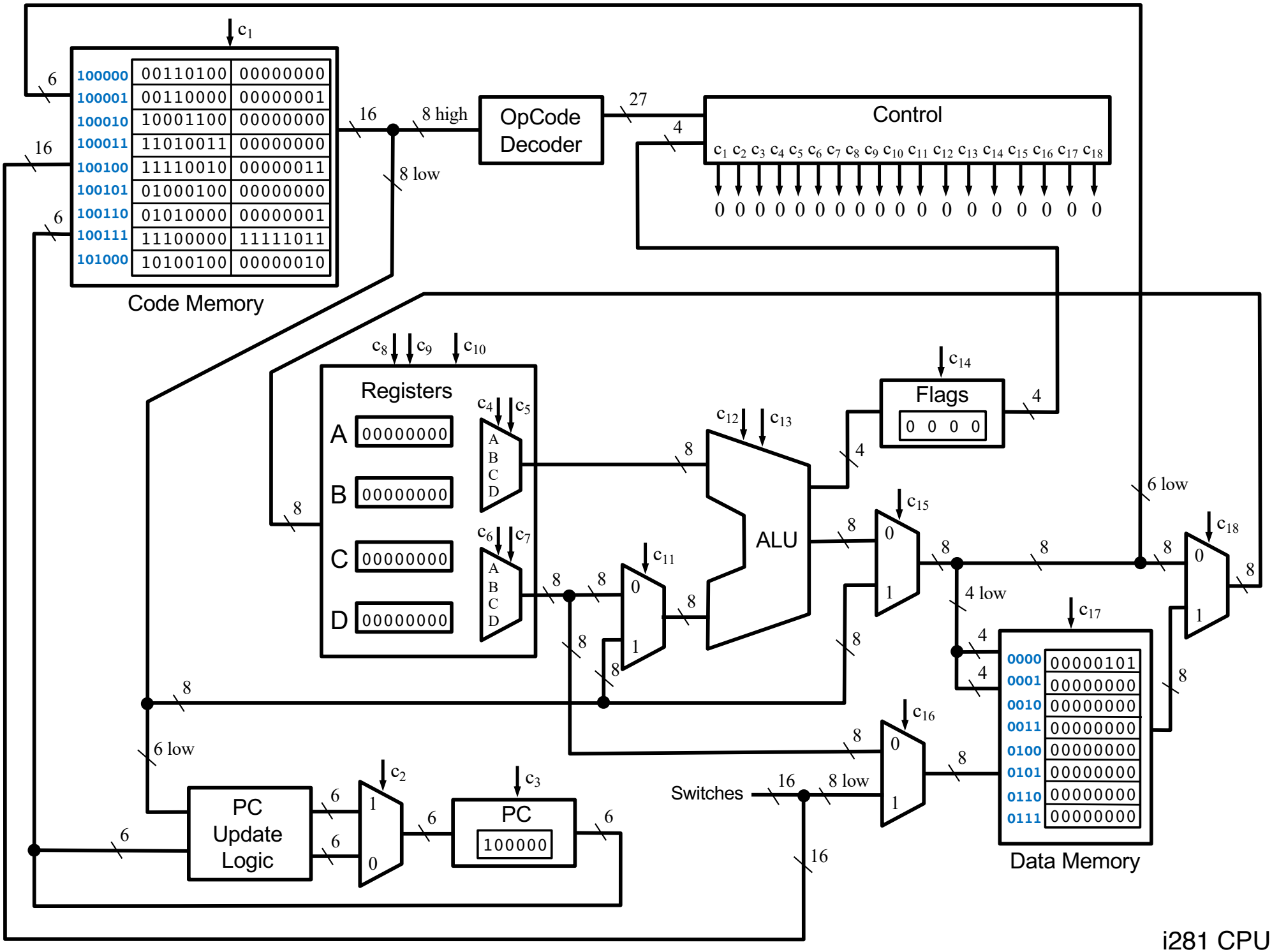


[Figure 5.25a from the textbook]

i281 CPU


c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c8 c9 c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
c4 c5
c6 c7
A B C D
c14
Flags
0 0 0 0
c12 c13
ALU
c15
c11
0 1
c18
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
Switches
c2
c3
PC Update Logic
PC
000000
i281 CPU


Write Select
c1 Write Enable
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
Input
Read Select
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Write Select
c8 c9
Write Enable
c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
Port0 Read Select
c4 c5
Port1 Read Select
c6 c7
A B C D
Input
ALU Select
c12 c13
ALU
ALU Source
c11 Mux
Write Enable
c14
Flags
0 0 0 0
ALU Result
c15 Mux
REG Writeback
Mux
c18
Write Enable
c17
Read Select
Write Select
DMEM Input
Mux
c16
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
Input
Switches
PC Mux
c2
Write Enable
c3
PC
100000
PC Update Logic
i281 CPU


c1
6
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
16
8 high
8 low
OpCode Decoder
27
4
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c8 c9 c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
c4 c5
c6 c7
A B C D
8
c14
Flags
0 0 0 0
4
c12 c13
ALU
c11
c15
c16
c18
6 low
4 low
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
Switches
8 low
c2
c3
PC
100000
PC Update Logic
6 low
i281 CPU

Add the numbers from 1 to 5

```c
// C Version
// using a for loop




int main()
{
        int N=5;
        int i, sum;

        sum=0;
        for(i=1; i<=N; i++) {
           sum+=i;
        }

        // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N         BYTE    5
i         BYTE    ?
sum       BYTE    ?

.code
          LOADI   B, 0        ; sum=0
          LOADI   A, 1        ; i=1
          LOAD    D, [N]      ; register_D=N
Loop:     CMP     A, D        ; i<=N ?
          BRG     End         ; exit if i>N
Add:      ADD     B, A        ; sum+=i
          ADDI    A, 1        ; i++
          JUMP    Loop        ; next iteration
End:      STORE   [sum], B    ; write B to sum
```

Mapping Assembly to Machine Code

```
.data
N         BYTE    5
i         BYTE    ?
sum       BYTE    ?

.code
          LOADI   B, 0
          LOADI   A, 1
          LOAD    D, [N]
Loop:     CMP     A, D
          BRG     End
Add:      ADD     B, A
          ADDI    A, 1
          JUMP    Loop
End:      STORE   [sum], B
```

Assembly Language

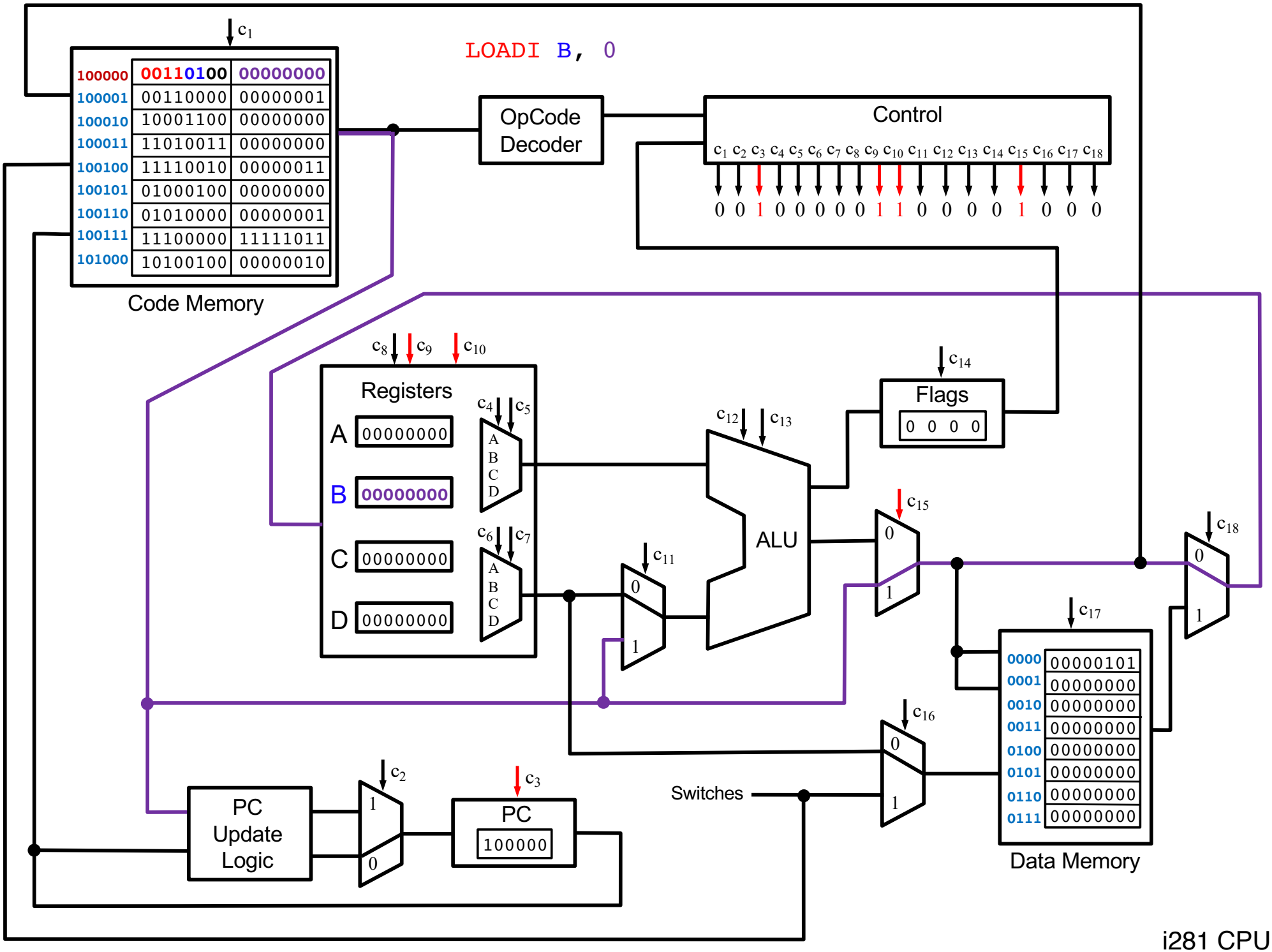
Data Memory:

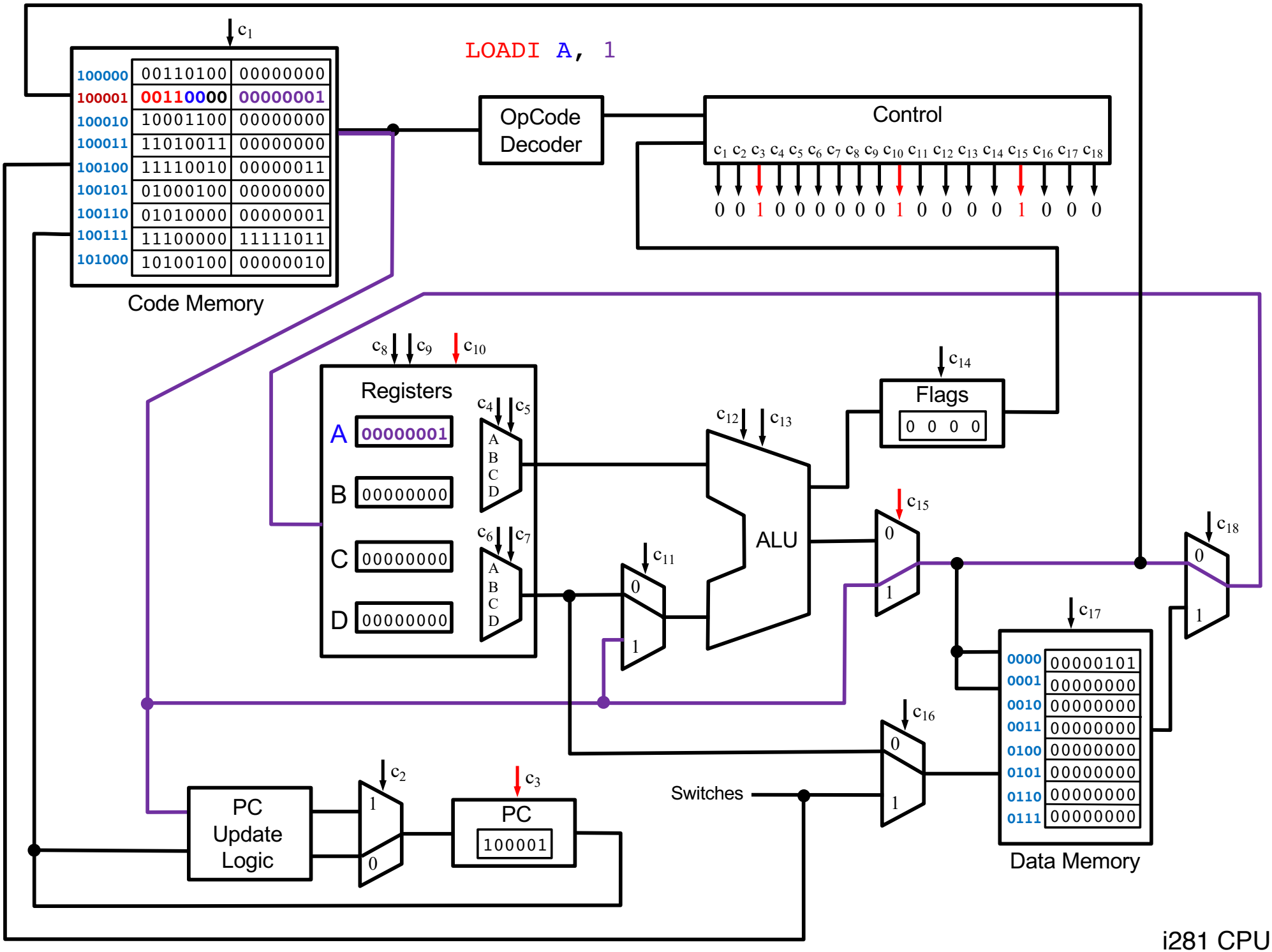
```
00000101
00000000
00000000
```

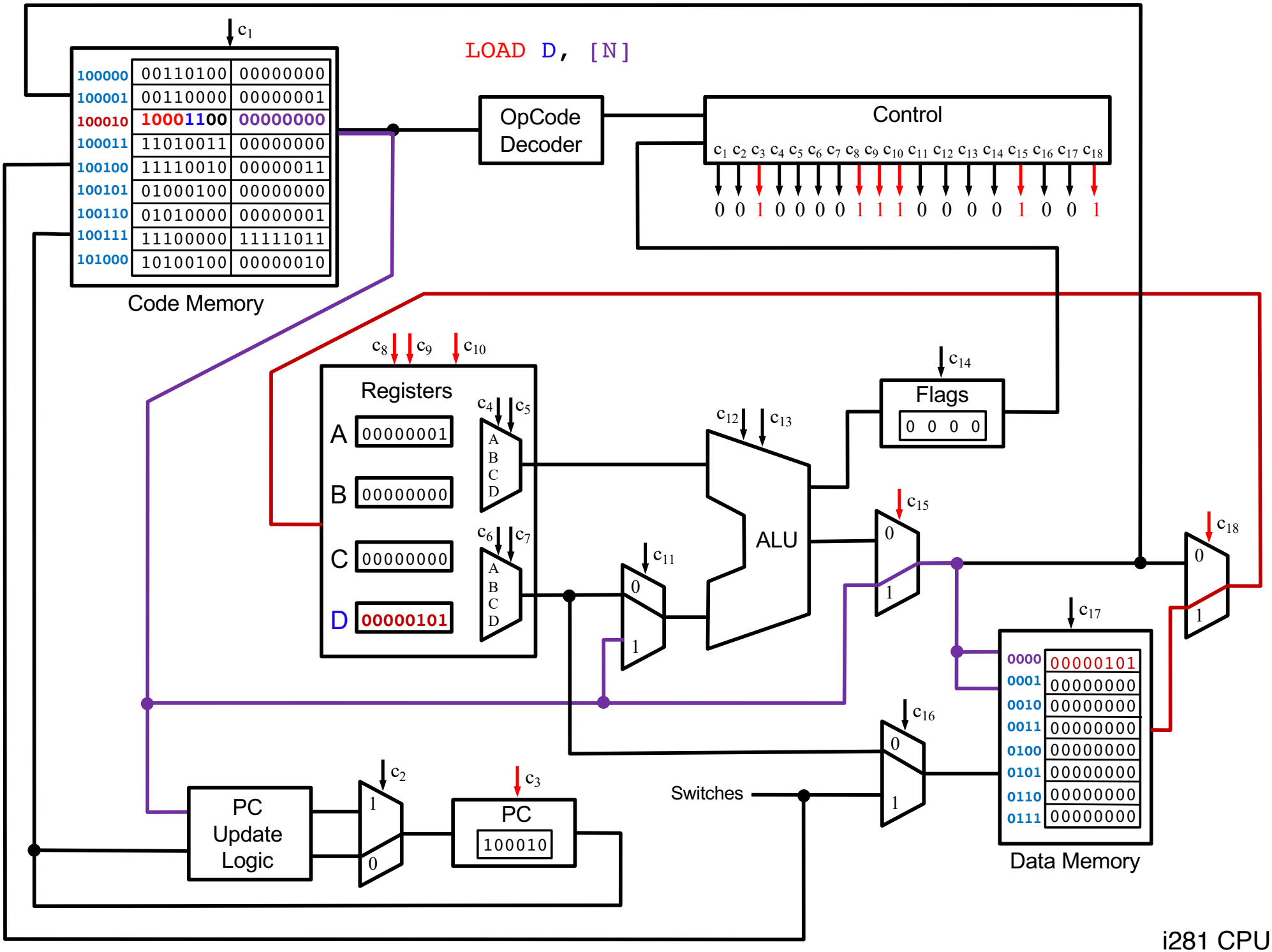
Code Memory:

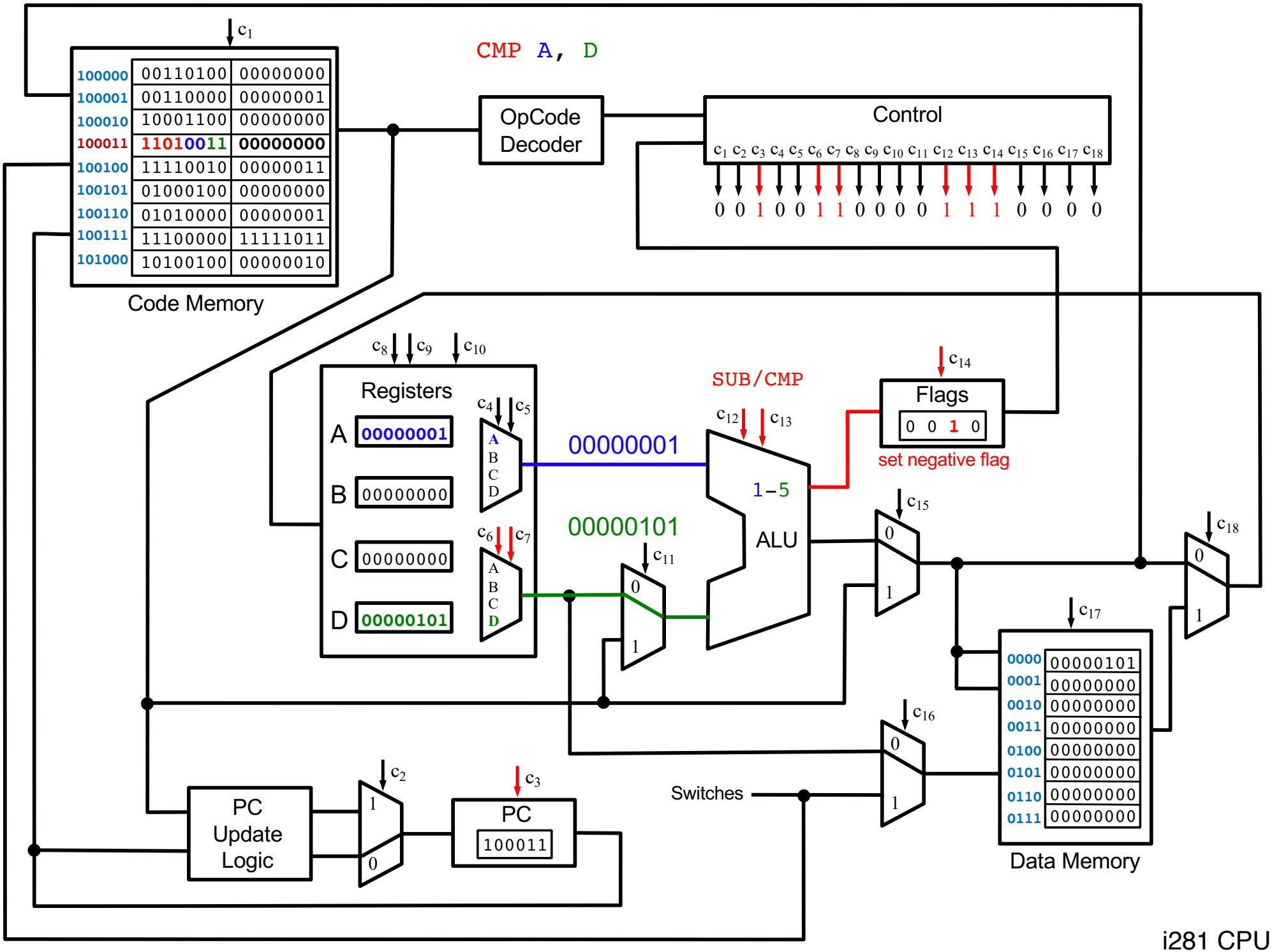
```
0011010000000000
0011000000000001
1000110000000000
1101001100000000
1111001000000011
0100010000000000
0101000000000001
1110000011111011
1010010000000010
```

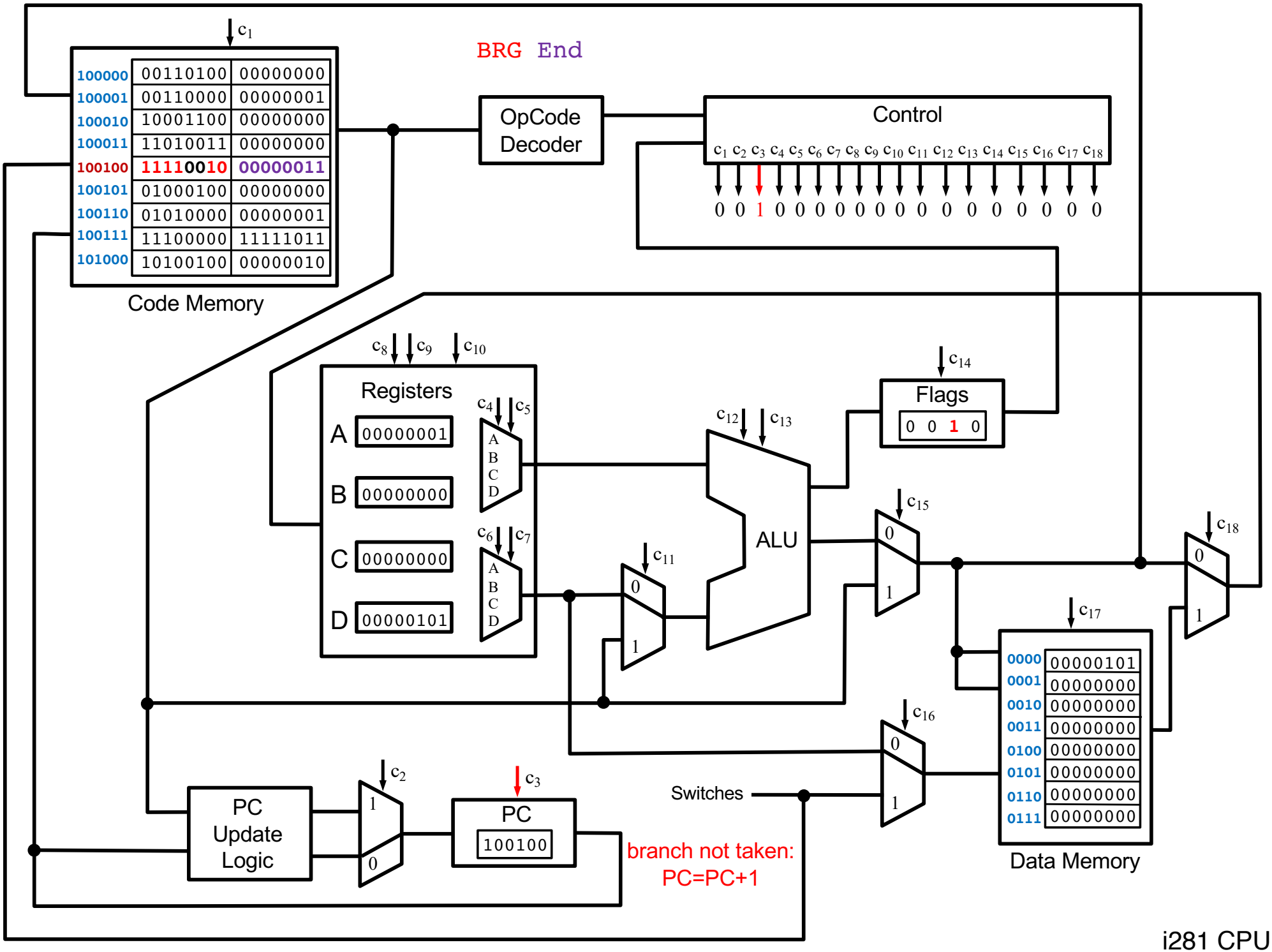
Machine Language

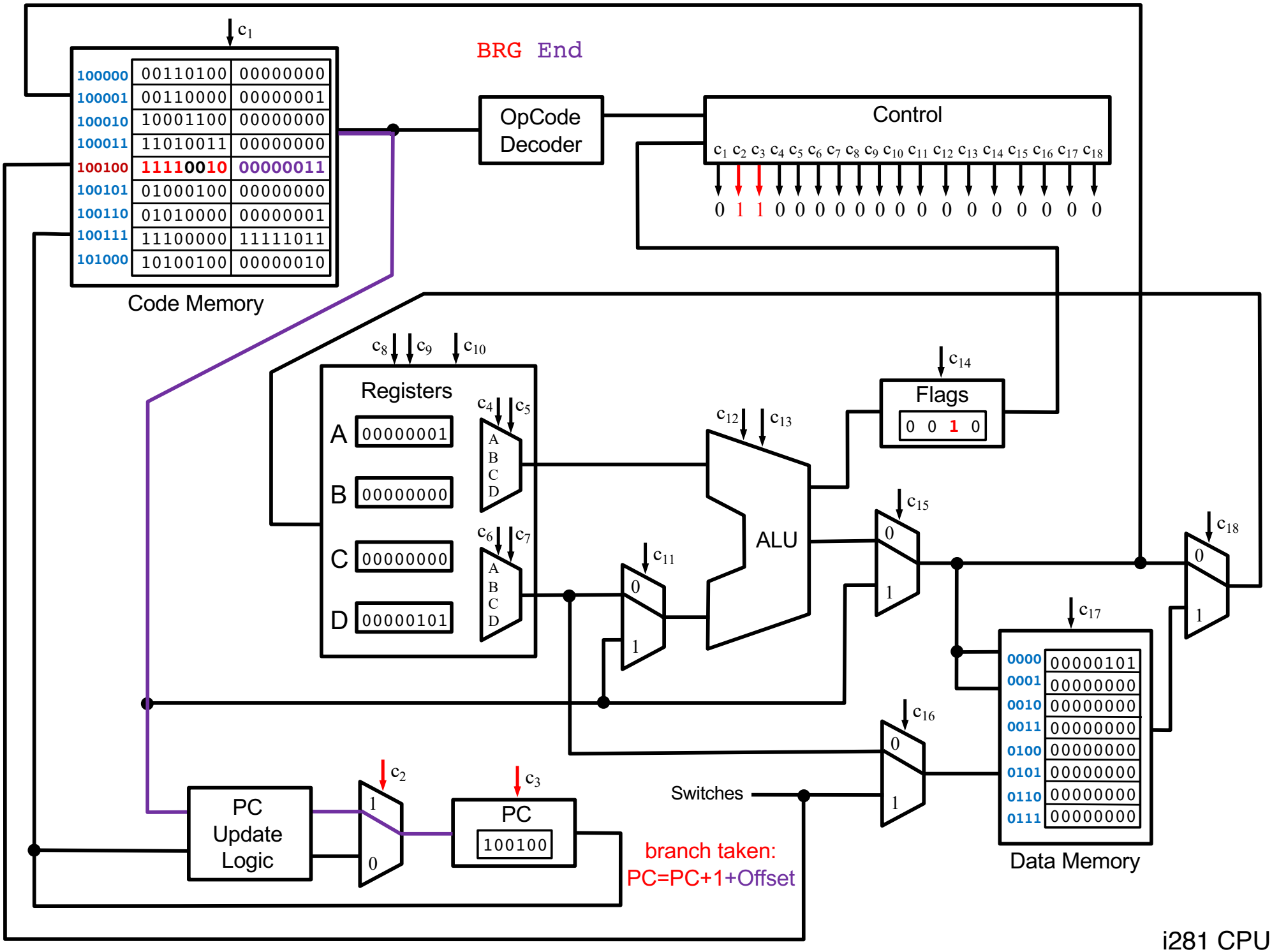

LOADI B, 0
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0
c8 c9 c10
Registers
A 00000000
B 00000000
C 00000000
D 00000000
c4 c5
c6 c7
A B C D
c11
c12 c13
ALU
c14
Flags
0 0 0 0
c15
c16
c17
c18
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
Switches
PC Update Logic
c2
c3
PC
100000
i281 CPU


LOADI A, 1
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000000
c4 c5
c6 c7
A B C D
c11
c12 c13
ALU
c14
Flags
0 0 0 0
c15
c18
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c16
Switches
c2
c3
PC Update Logic
PC
100001
i281 CPU


LOAD D, [N]
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 0 0 1
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
c6 c7
A B C D
c11
c12 c13
ALU
c14
Flags
0 0 0 0
c15
c16
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c18
Switches
c2
c3
PC Update Logic
PC
100010
i281 CPU


CMP A, D
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
c6 c7
A B C D
00000001
00000101
c11
SUB/CMP
c12 c13
1-5
ALU
c14
Flags
0 0 1 0
set negative flag
c15
c16
c17
c18
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
Switches
c2
c3
PC Update Logic
PC
100011
i281 CPU


BRG End
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
c6 c7
A B C D
c11
c12 c13
ALU
c14
Flags
0 0 1 0
c15
c16
c17
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
c18
Switches
c2
c3
PC Update Logic
PC
100100
branch not taken:
PC=PC+1
i281 CPU


BRG End
c1
100000 00110100 00000000
100001 00110000 00000001
100010 10001100 00000000
100011 11010011 00000000
100100 11110010 00000011
100101 01000100 00000000
100110 01010000 00000001
100111 11100000 11111011
101000 10100100 00000010
Code Memory
OpCode Decoder
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c8 c9 c10
Registers
A 00000001
B 00000000
C 00000000
D 00000101
c4 c5
c6 c7
A B C D
c11
c12 c13
ALU
c14
Flags
0 0 1 0
c15
c16
c17
c18
0000 00000101
0001 00000000
0010 00000000
0011 00000000
0100 00000000
0101 00000000
0110 00000000
0111 00000000
Data Memory
Switches
c2
c3
PC Update Logic
PC
100100
branch taken:
PC=PC+1+Offset
i281 CPU

Bubble Sort

C Version

```
int array[] = {7, 3, 2, 1, 6, 4, 5, 8};
int last = 7; // last valid index in the array
int temp;
int i, j;

int main()
{
   for (i = 0; i < last; i++)
        for (j = 0; j < last-i; j++)
               if (array[j] > array[j+1]){
                      temp = array[j];
                      array[j] = array[j+1];
                      array[j+1] = temp;
               }

   //for(i = 0; i < N; i++){
   //    printf("%d, ", array[i]);
   //}
}
```

Assembly Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0                 ; i = 0;
Outer:  LOAD   D, [last]            ; Load last into D
        LOADI  B, 0                 ; j = 0;
        CMP    A, D                 ; i < last
        BRGE   End                  ; If i >= last break out of the outer loop
Inner:  LOAD   D, [last]            ; Re-Load last into D (this register is shared)
        SUB    D, A                 ; D = D - A  (i.e., D = last - i)
        CMP    B, D                 ; j < last - i
        BRGE   Iinc                 ; If j >= last-i  branch to Iinc
If:     LOADF  C, [array+B]         ; C = array[j]
        LOADF  D, [array+B+1]       ; D = array[j+1] (compiler adds 1 to addr. of array)
        CMP    D, C                 ; if array[j+1] < array[j]  (switched direction)
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1                 ; j++
        JUMP   Inner
Iinc:   ADDI   A, 1                 ; i++
        JUMP   Outer
End:    NOOP                        ; Do nothing

; Register allocation:
; A: i
; B: j
; C: array[j]
; D: last, array[j+1]

; Notes: i and j are optimized away. They exist only in registers, not in the main memory.
```

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Data Memory:

```
00000111
00000011
00000010
00000001
00000110
00000100
00000101
00001000
00000111
00000000
```

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Data Memory:

```
00000111 //array[0]
00000011 //array[1]
00000010 //array[2]
00000001 //array[3]
00000110 //array[4]
00000100 //array[5]
00000101 //array[6]
00001000 //array[7]
00000111 //last
00000000 //temp
```

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Address	Data Memory:
0000	00000111 //array[0]
0001	00000011 //array[1]
0010	00000010 //array[2]
0011	00000001 //array[3]
0100	00000110 //array[4]
0101	00000100 //array[5]
0110	00000101 //array[6]
0111	00001000 //array[7]
1000	00000111 //last
1001	00000000 //temp

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Address	Data Memory:
0000	00000111 //array[0]
0001	00000011 //array[1]
0010	00000010 //array[2]
0011	00000001 //array[3]
0100	00000110 //array[4]
0101	00000100 //array[5]
0110	00000101 //array[6]
0111	00001000 //array[7]
1000	00000111 //last
1001	00000000 //temp
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Code Memory:

```
0011000000000000
1000110000001000
0011010000000000
1101001100000000
1111001100001110
1000110000001000
0110110000000000
1101011100000000
1111001100001000
1001100100000000
1001110100000001
1101111000000000
1111001100000010
1011110100000000
1011100100000001
0101010000000001
1110000011110100
0101000000000001
1110000011101110
0000000000000000
```

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Address	Code Memory:
100000	0011000000000000
100001	1000110000001000
100010	0011010000000000
100011	1101001100000000
100100	1111001100001110
100101	1000110000001000
100110	0110110000000000
100111	1101011100000000
101000	1111001100001000
101001	1001100100000000
101010	1001110100000001
101011	1101111000000000
101100	1111001100000010
101101	1011110100000000
101110	1011100100000001
101111	0101010000000001
110000	1110000011110100
110001	0101000000000001
110010	1110000011101110
110011	0000000000000000

Machine Code Version

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE   End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE   Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP   Inner
Iinc:   ADDI   A, 1
        JUMP   Outer
End:    NOOP
```

Address	Code Memory:
100000	0011000000000000
100001	1000110000001000
100010	0011010000000000
100011	1101001100000000
100100	1111001100001110
100101	1000110000001000
100110	0110110000000000
100111	1101011100000000
101000	1111001100001000
101001	1001100100000000
101010	1001110100000001
101011	1101111000000000
101100	1111001100000010
101101	1011110100000000
101110	1011100100000001
101111	0101010000000001
110000	1110000011110100
110001	0101000000000001
110010	1110000011101110
110011	0000000000000000
110100	0000000000000000
. . .	. . .
111110	0000000000000000
111111	0000000000000000

Questions?

THE END