

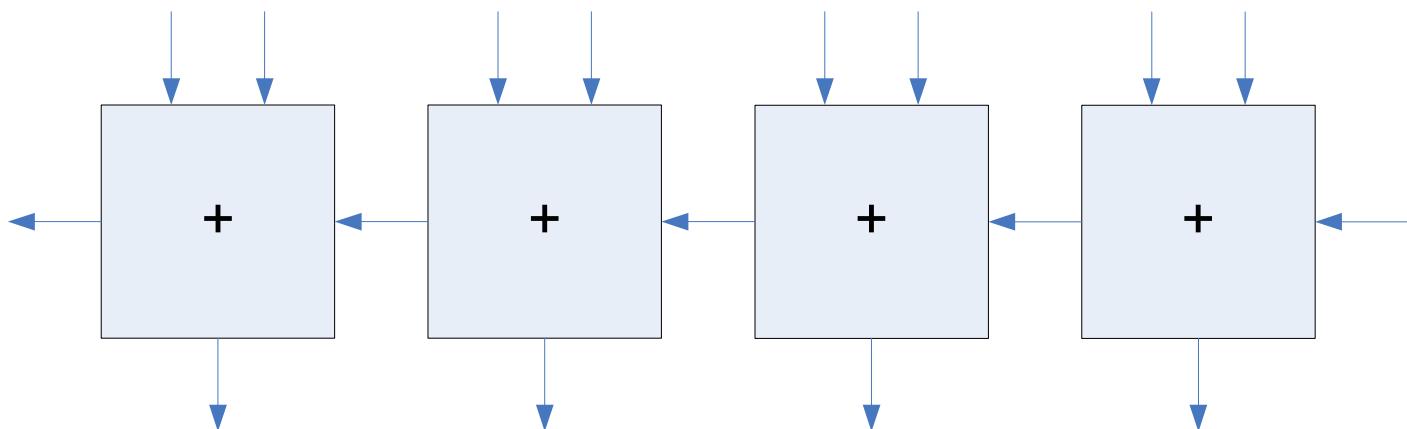
# **Computer Architecture**

Lecture 2

# **Carry-Lookahead Adder and Memory Elements (Appendix A.6 & A-8)**

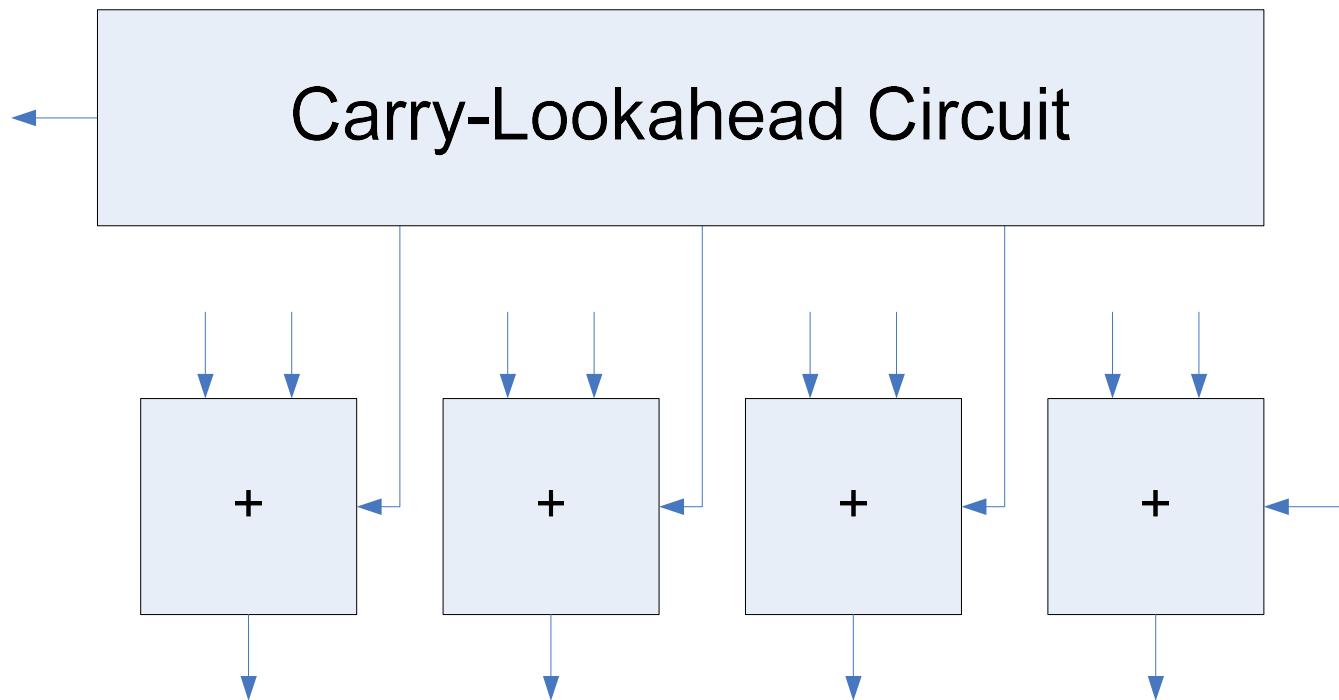
# Improving Addition Performance

- ❑ The ripple-carry adder is slow



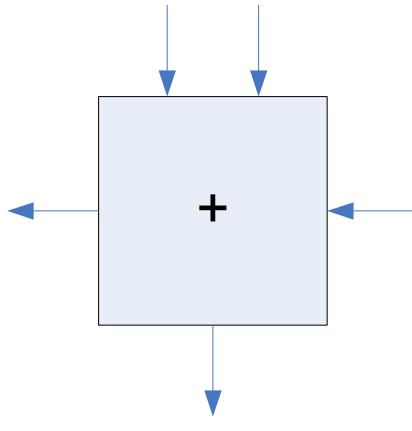
# Carry-Lookahead Adder

- ❑ Need fast way to find the carry



# Carry-Lookahead Adder

- ❑ Carry generate and carry propagate



$a_i$	$b_i$	$g_i$	$p_i$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

- ❑  $g_i = a_i \cdot b_i$
- ❑  $p_i = a_i + b_i$

# Carry-Lookahead Adder

Carry Equations:

$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 c_1$$

$$= g_1 + p_1 g_0 + p_1 p_0 c_0$$

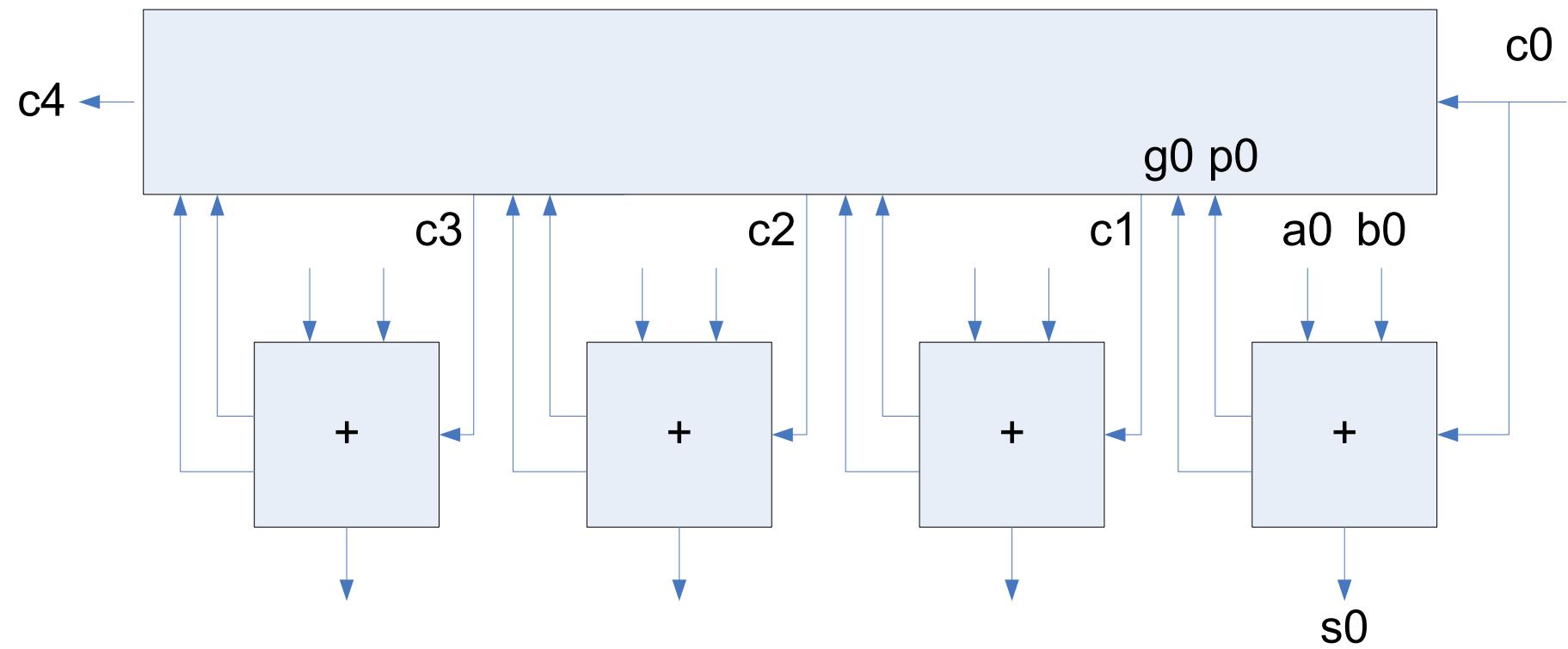
$$c_3 = g_2 + p_2 c_2$$

$$= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$c_4 = g_3 + p_3 c_3$$

$$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$

# 4-bit Carry-Lookahead Adder

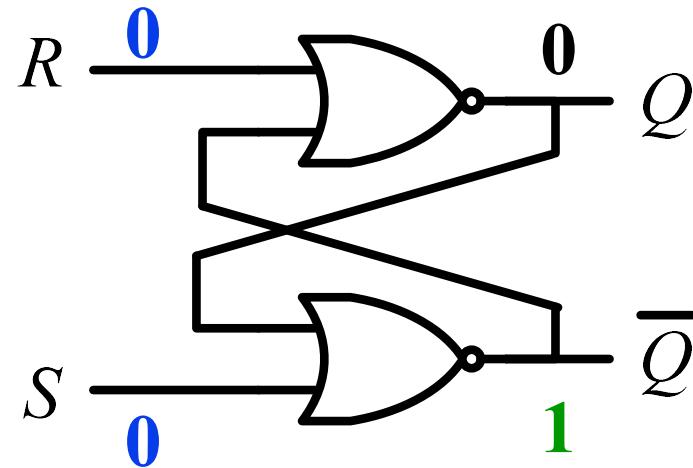


# Memory Elements

- ❑ Latches
- ❑ Flip flops
- ❑ Registers

# Latches

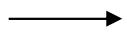
## □ SR Latch



Initial Value

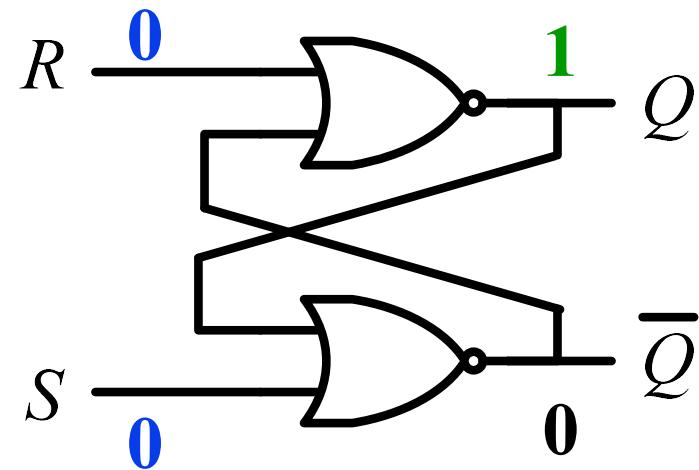
$S$	$R$	$Q_0$	$Q$	$Q'$
0	0	0	0	1

$$Q = Q_0$$



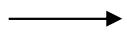
# Latches

## □ SR Latch



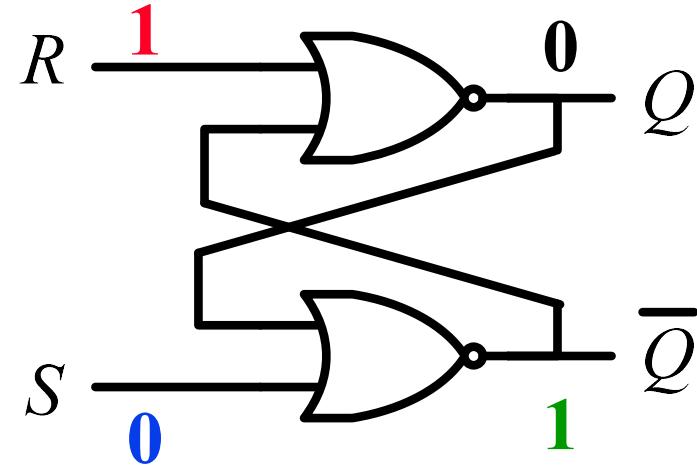
$S$	$R$	$Q_0$	$Q$	$Q'$
0	0	0	0	1
0	0	1	1	0

$$\begin{aligned} Q &= Q_0 \\ \bar{Q} &= \bar{Q}_0 \end{aligned}$$



# Latches

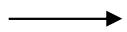
## □ SR Latch



$S$	$R$	$Q_0$	$Q$	$Q'$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1

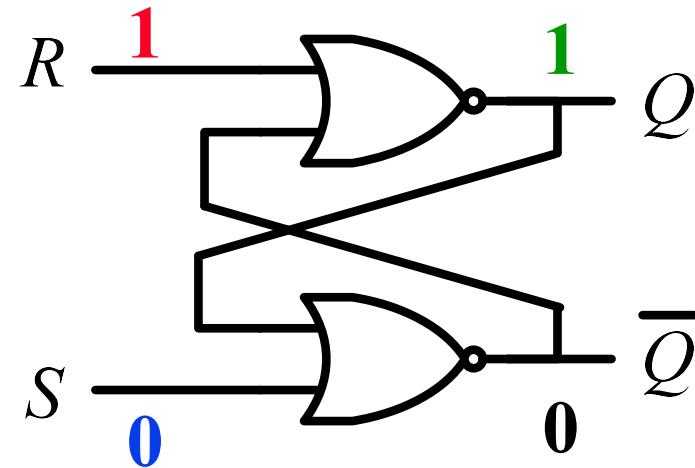
Annotations on the right side of the table indicate the state of  $Q$  based on the initial value  $Q_0$ :

- $Q = Q_0$  (highlighted in red) applies to the first three rows where  $Q_0$  is 0 or 1.
- $Q = 0$  (highlighted in red) applies to the last five rows where  $Q_0$  is 0.



# Latches

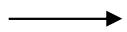
## □ SR Latch



$S$	$R$	$Q_0$	$Q$	$Q'$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1

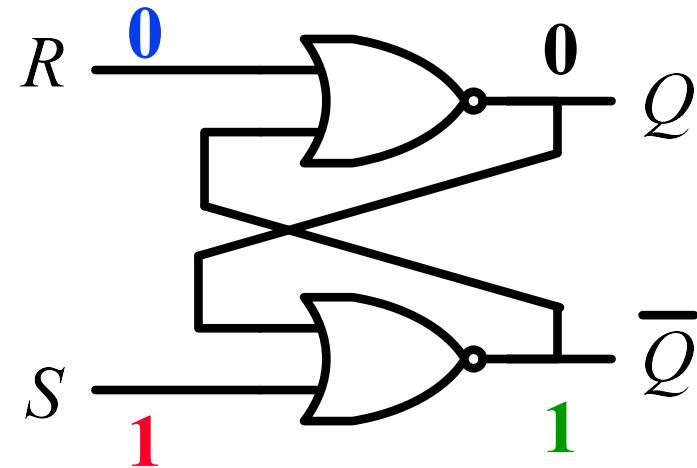
Annotations on the right side of the table:

- A brace groups the first two rows:  $Q = Q_0$
- $Q = 0$  is written next to the third row.
- $Q = 0$  is written next to the fourth row.



# Latches

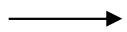
## □ SR Latch



$S$	$R$	$Q_0$	$Q$	$Q'$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0

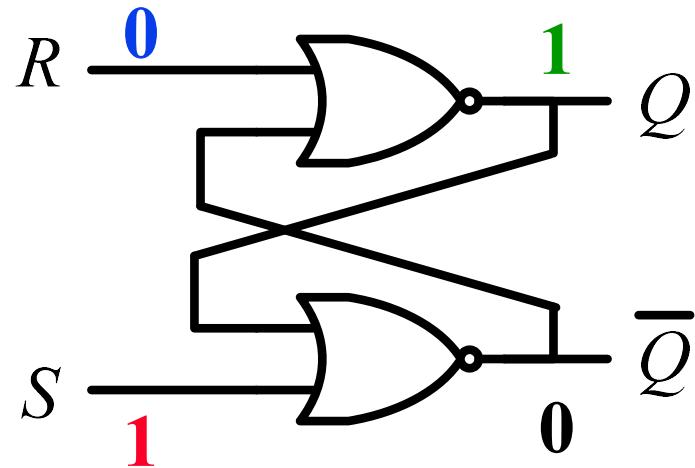
Annotations on the right side of the table indicate the state of  $Q$  based on the initial value  $Q_0$ :

- $Q = Q_0$  (for rows 1, 2, 4, and 5)
- $Q = 0$  (for rows 3 and 6)
- $Q = 1$  (for rows 7, 8, and 9)



# Latches

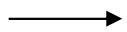
## □ SR Latch



S	R	$Q_0$	$Q$	$Q'$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0

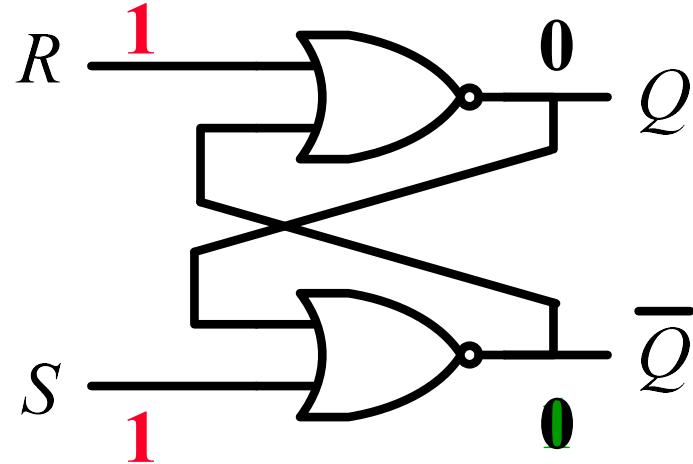
Annotations on the right side of the table:

- $Q = Q_0$  (with curly braces) covers rows 1, 2, and 3.
- $Q = 0$  (with curly braces) covers rows 4, 5, and 6.
- $Q = 1$  (with curly braces) covers rows 7 and 8.
- $Q = 1$  (with curly braces) covers rows 7 and 8.



# Latches

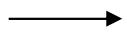
## □ SR Latch



$S$	$R$	$Q_0$	$Q$	$Q'$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0

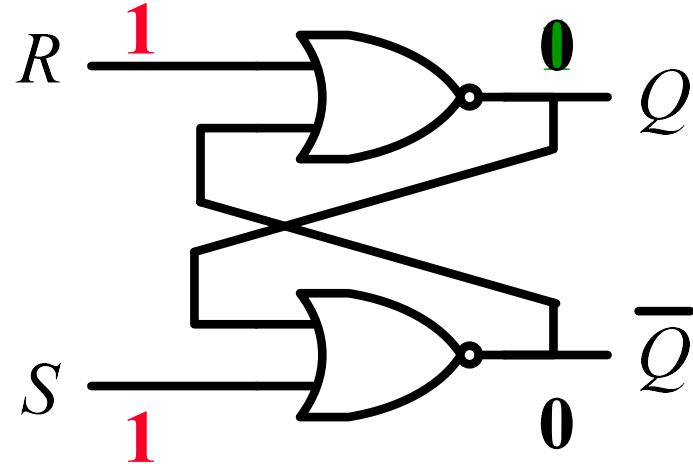
Annotations on the right side of the table indicate the state of  $Q$  based on the initial value  $Q_0$ :

- $Q = Q_0$  (for rows 1, 2, 4, 5, 7)
- $Q = 0$  (for rows 3, 6)
- $Q = 1$  (for row 8)
- $Q = Q'$  (for row 0)



# Latches

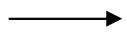
## □ SR Latch



$S$	$R$	$Q_0$	$Q$	$Q'$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0

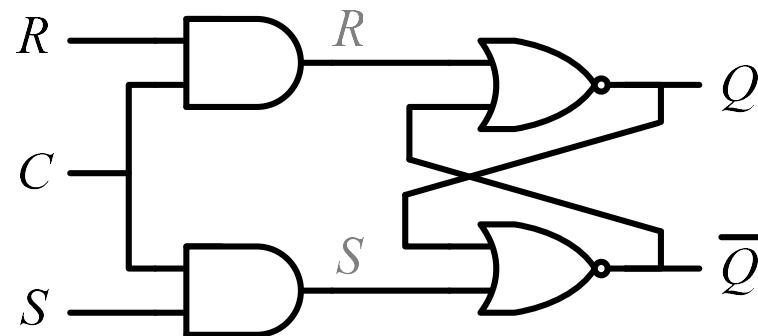
Annotations on the right side of the table:

- $Q = Q_0$  (for rows 1, 2, 3, 4)
- $Q = 0$  (for rows 5, 6)
- $Q = 1$  (for rows 7, 8)
- $Q = Q'$  (for rows 1, 2, 3, 4)
- $Q = \bar{Q}'$  (for rows 5, 6)

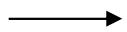


# Controlled Latches

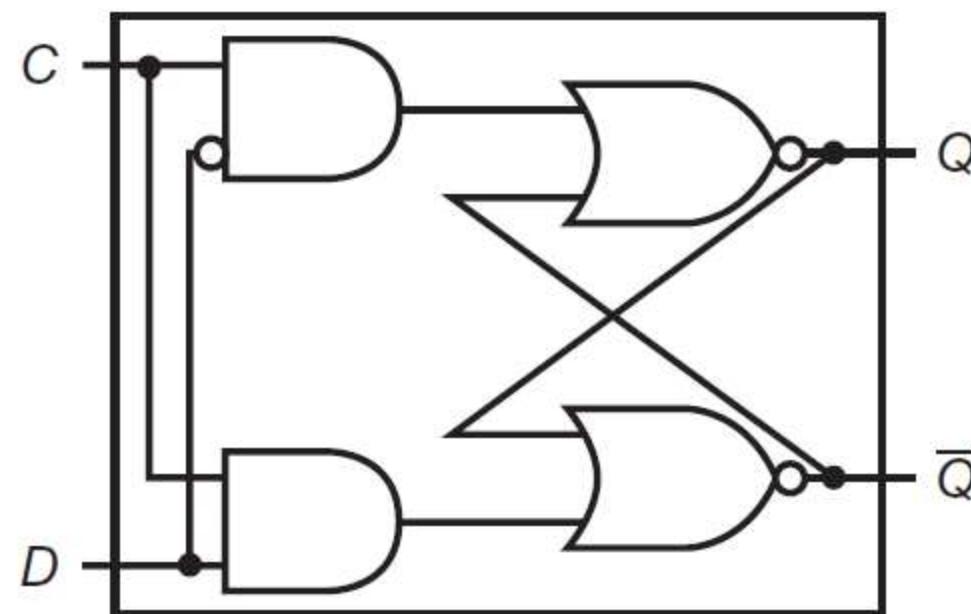
- SR Latch with Control Input



$C$	$S$	$R$	$Q$	
0	x	x	$Q_0$	No change
1	0	0	$Q_0$	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	$Q=Q'$	Invalid

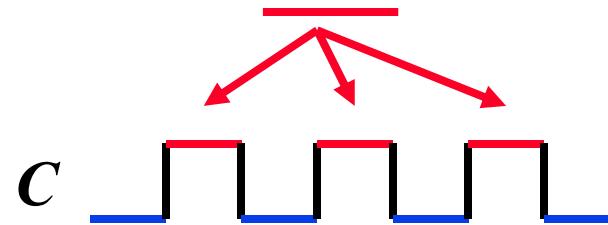


## A D latch implemented with NOR gates

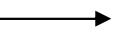
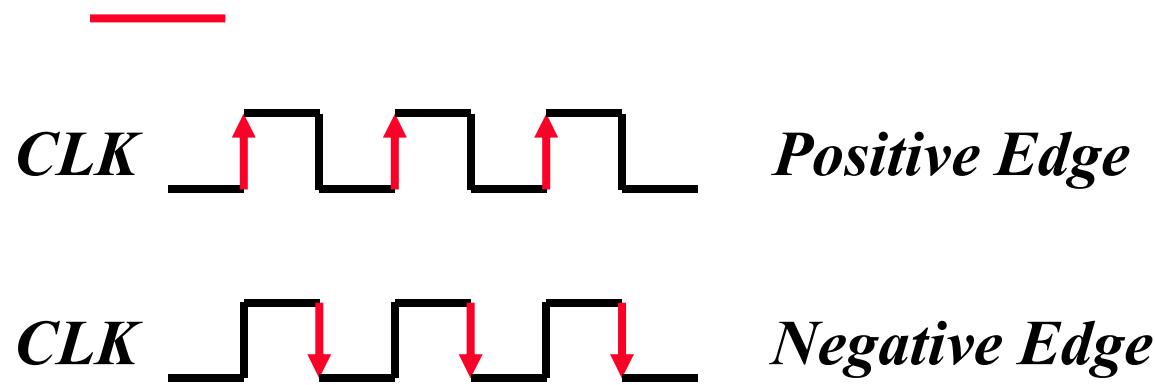


# Flip-Flops

- Controlled latches are **level**-triggered

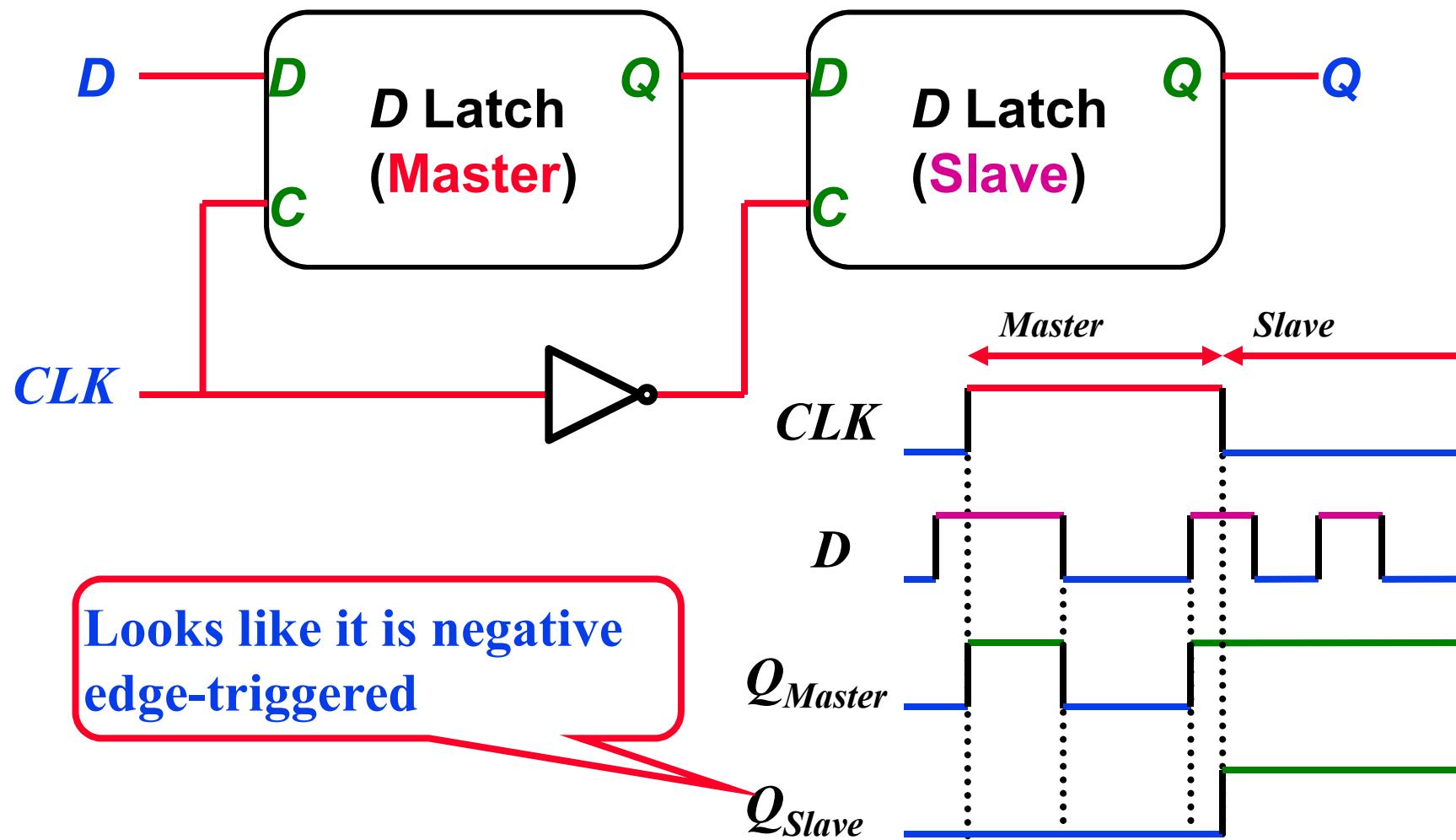


- Flip-Flops are **edge**-triggered

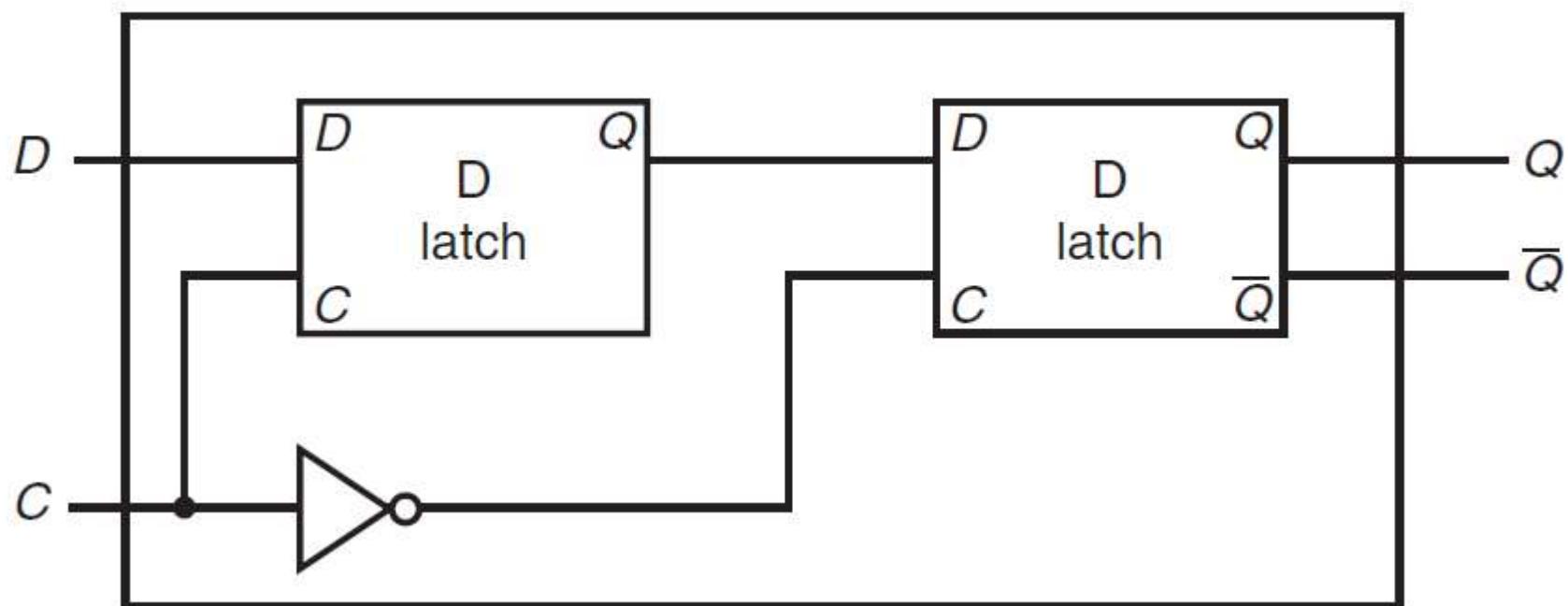


# Flip-Flops

## □ Master-Slave $D$ Flip-Flop

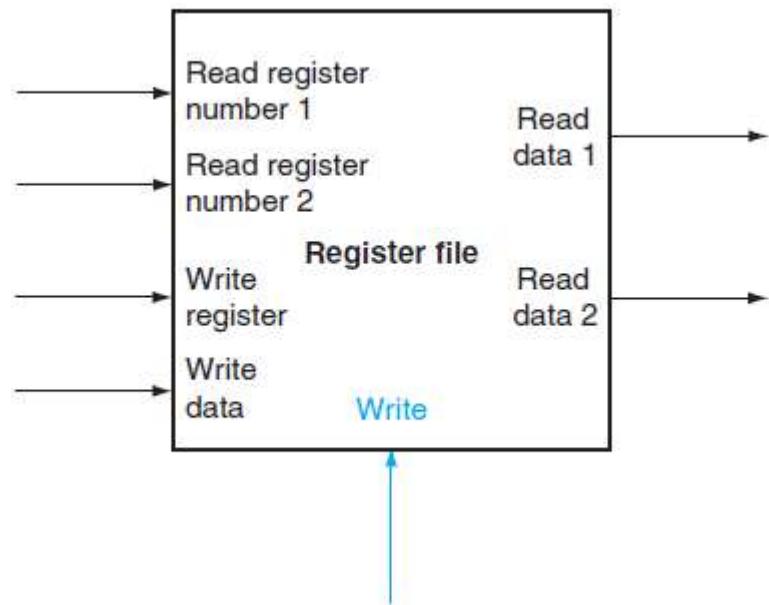


## A D flip-flop implemented with a falling-edge trigger



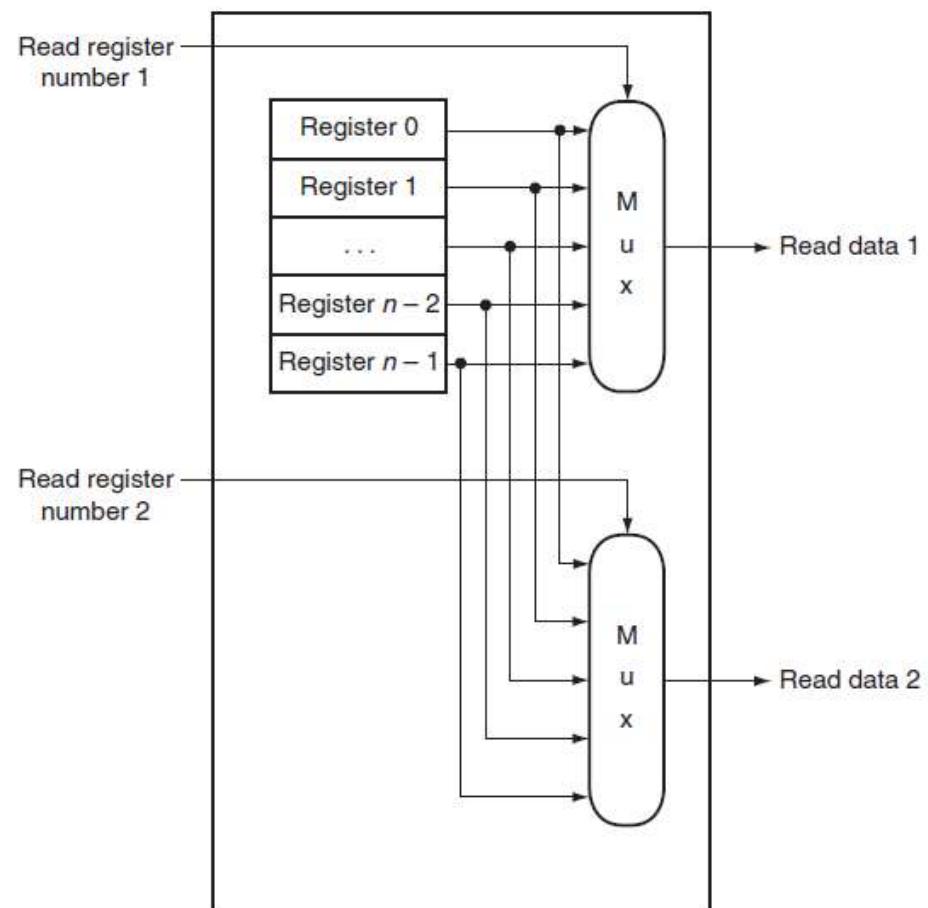
# Register files

- A register file with
  - Two read ports
  - One write port
  - Five inputs
  - Two outputs



# Read ports of the register file

- ❑ An implementation of two register read ports for a 64-bit-wide register file.



# Write ports of the register file

- Write port implementation for the register file.

