# Datapath Components-Part II

Jia Chen
jiac@ucr.edu

# Datapath components

| | |
|---|---|
| Multiplexer | Decoder |
| Adder | Subtractor |
| Register | Encoder |
| Shifter | Counter |
| Multiplier | Divider |
| Comparator | ... |

# Registers

# Registers

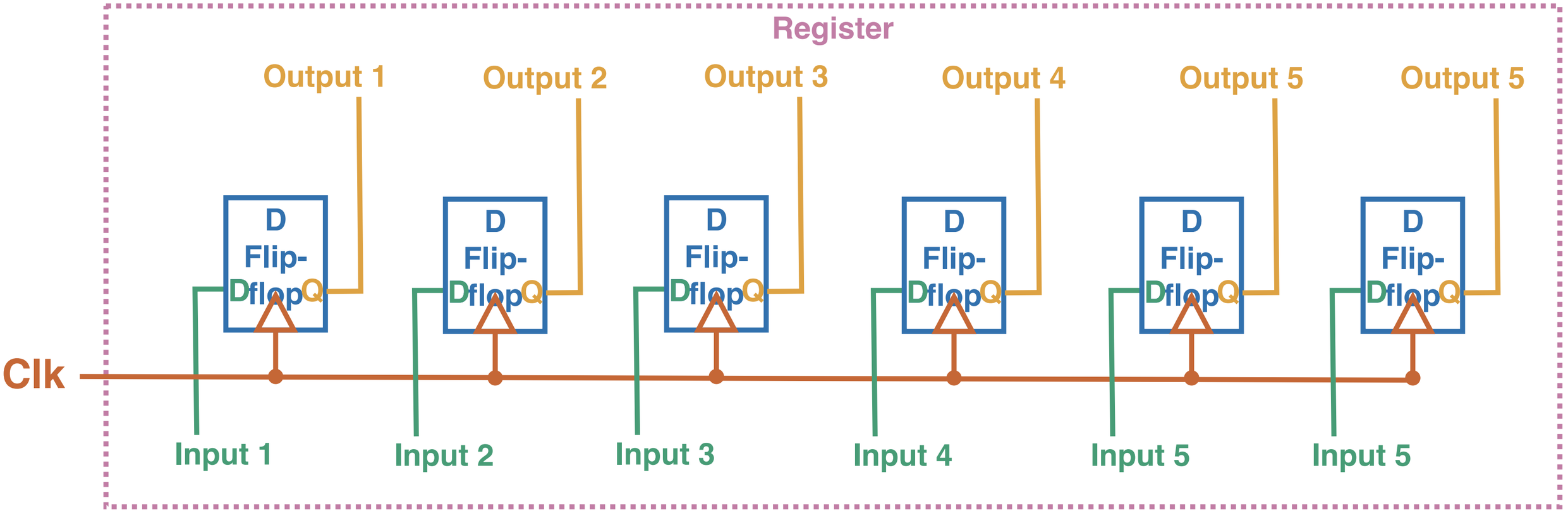
- **Register**: logic unit used to **store multiple bits** in a sequential logic circuit
  - A basic register can be built simply by using multiple **D-FF**s
  - Two common types of registers:

    **Parallel load register**

    **Shift register**

# Parallel load register

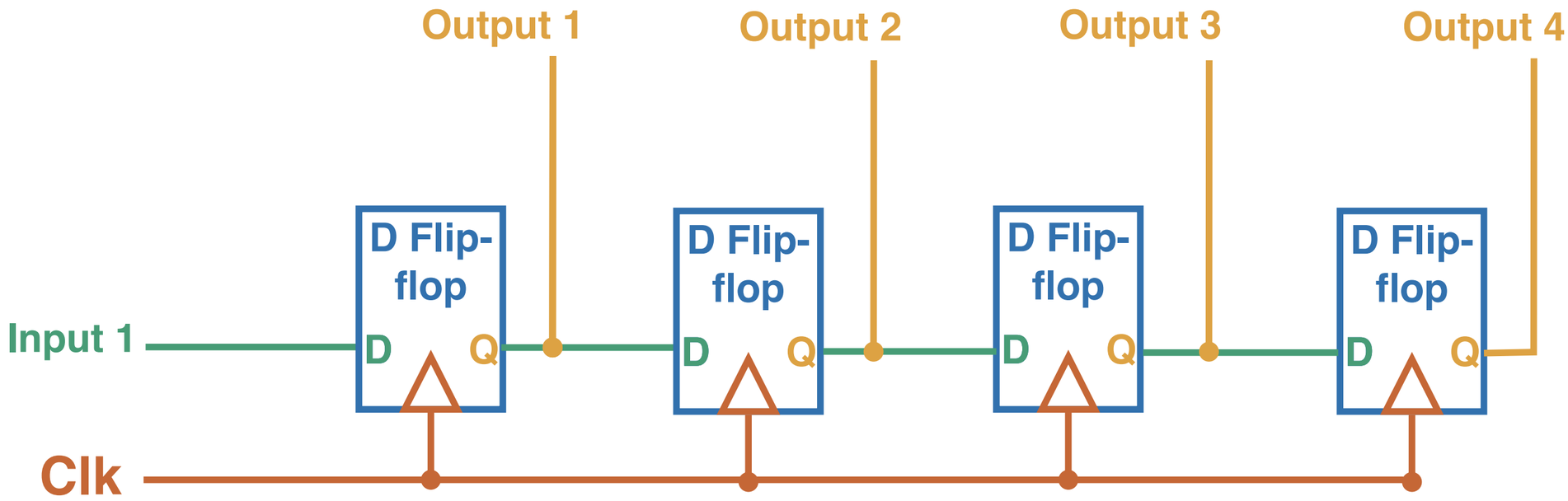
- **Parallel load register: individual bit values in the register are loaded simultaneously.**



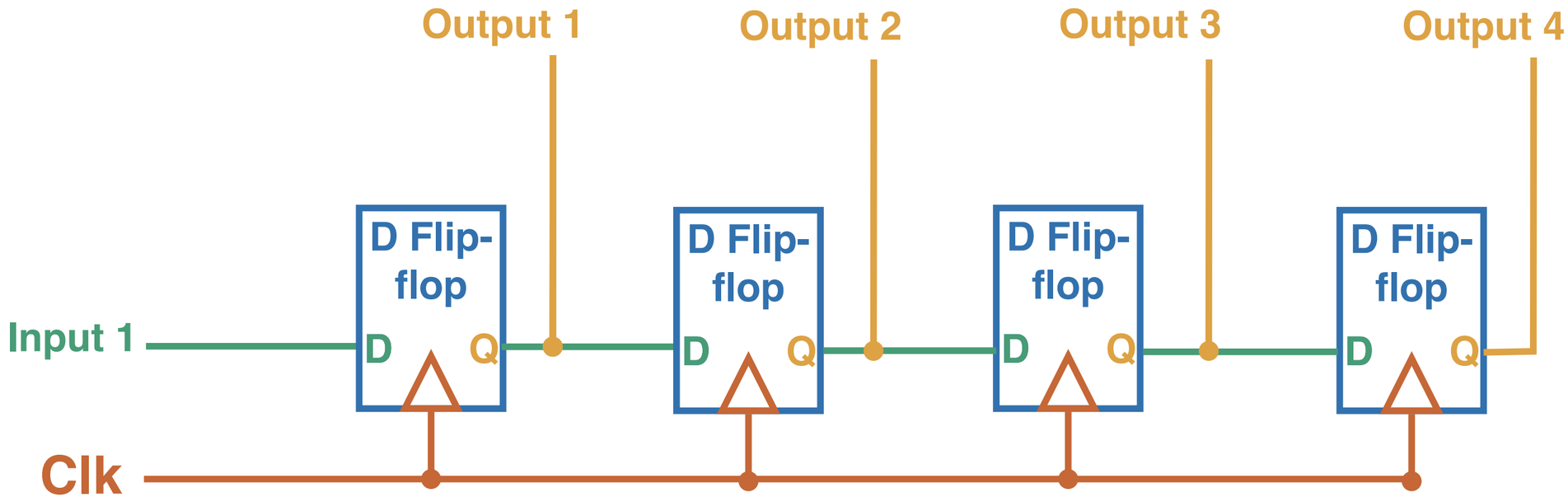
# Shift register

- **Shift register: a register that provides the ability to shift its contents**
  - The D-FFs share a single clock signal, which causes the data stored in the system to shift from one location to the next

For example,

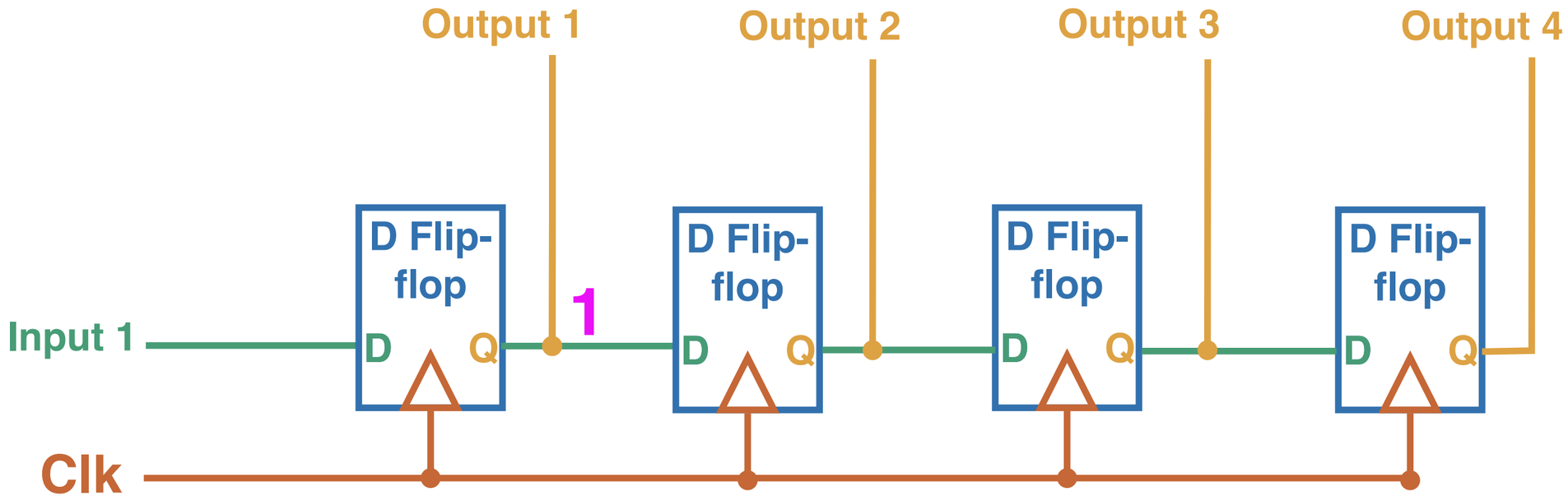


# What will we output 4 cycles later?



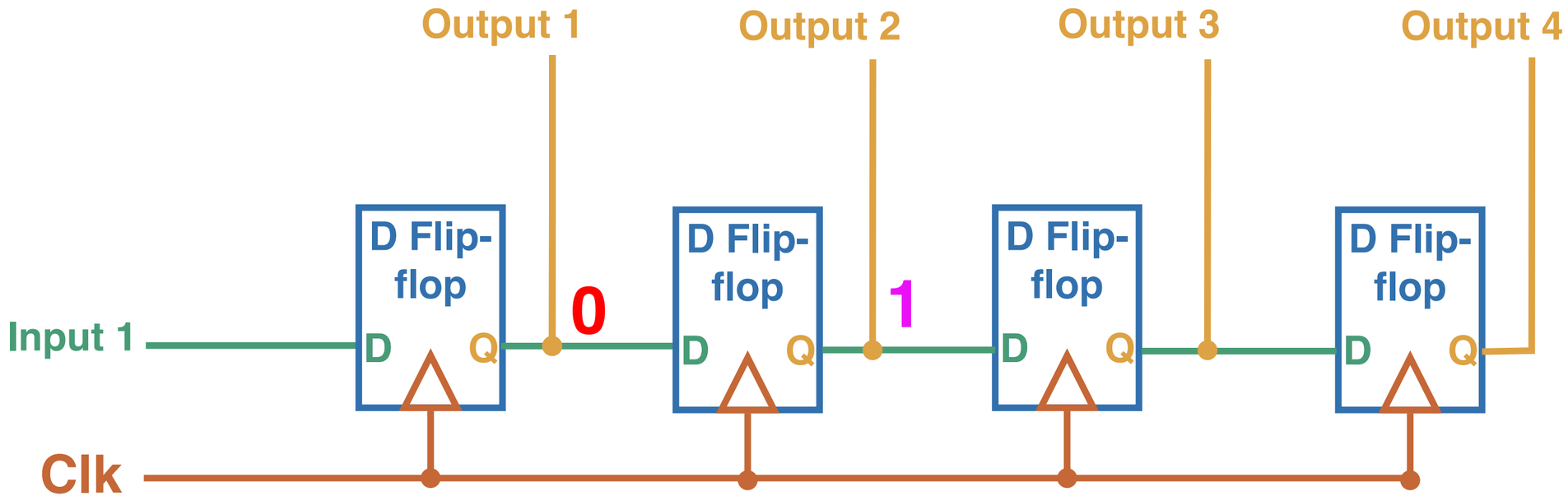

- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?

# 1 cycle later?



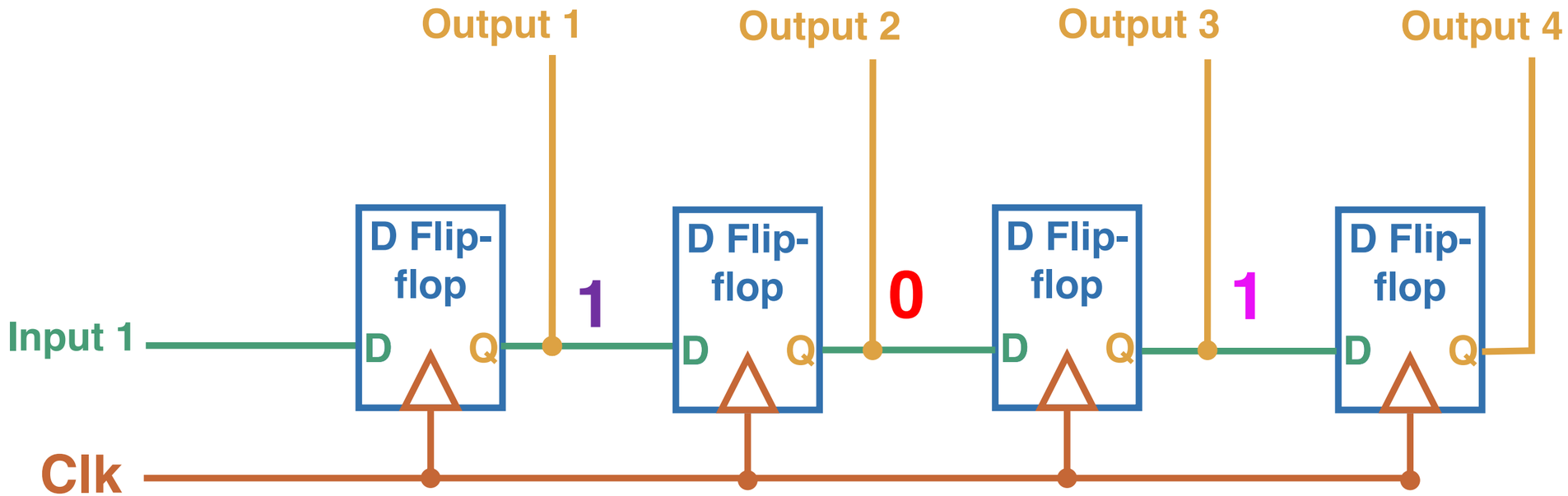

- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?

# 2 cycles later?



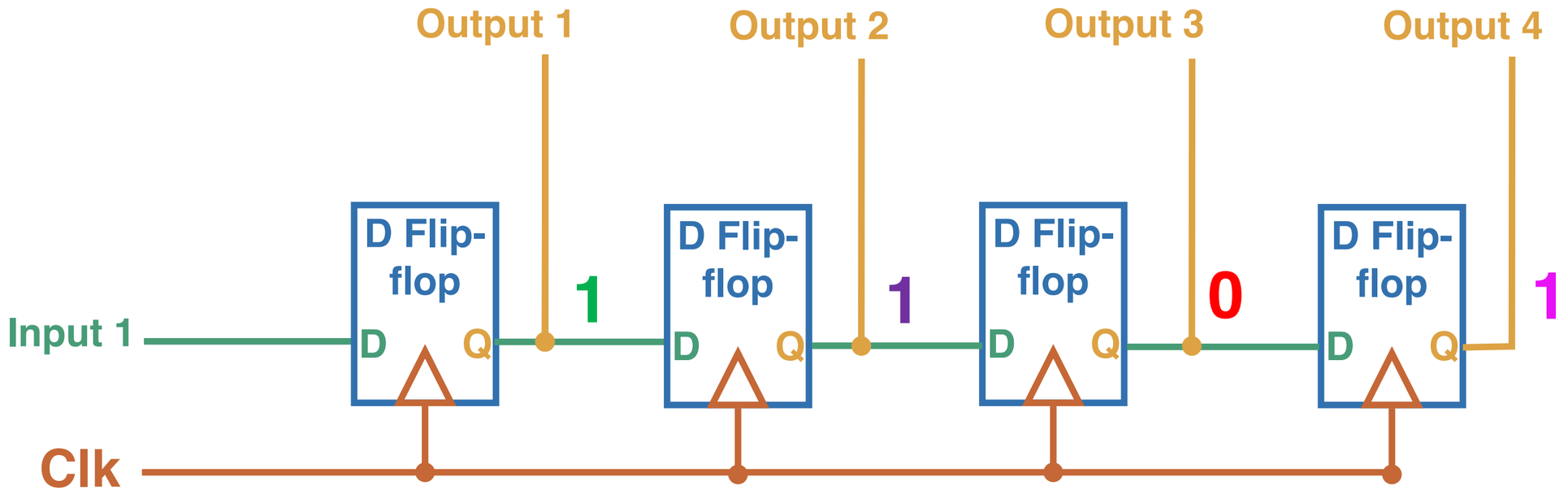

- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?

# 3 cycles later?



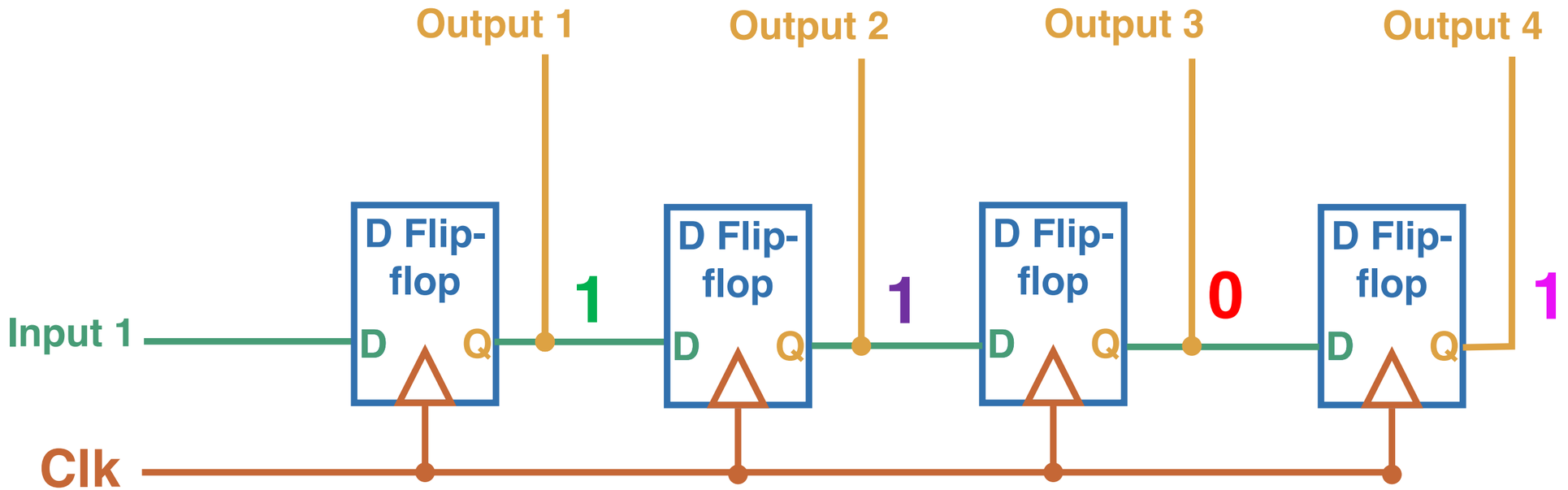

- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?

# 4 cycles later (aka the beginning of the 5th cycle)?




- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?

# What will we output 4 cycles later?

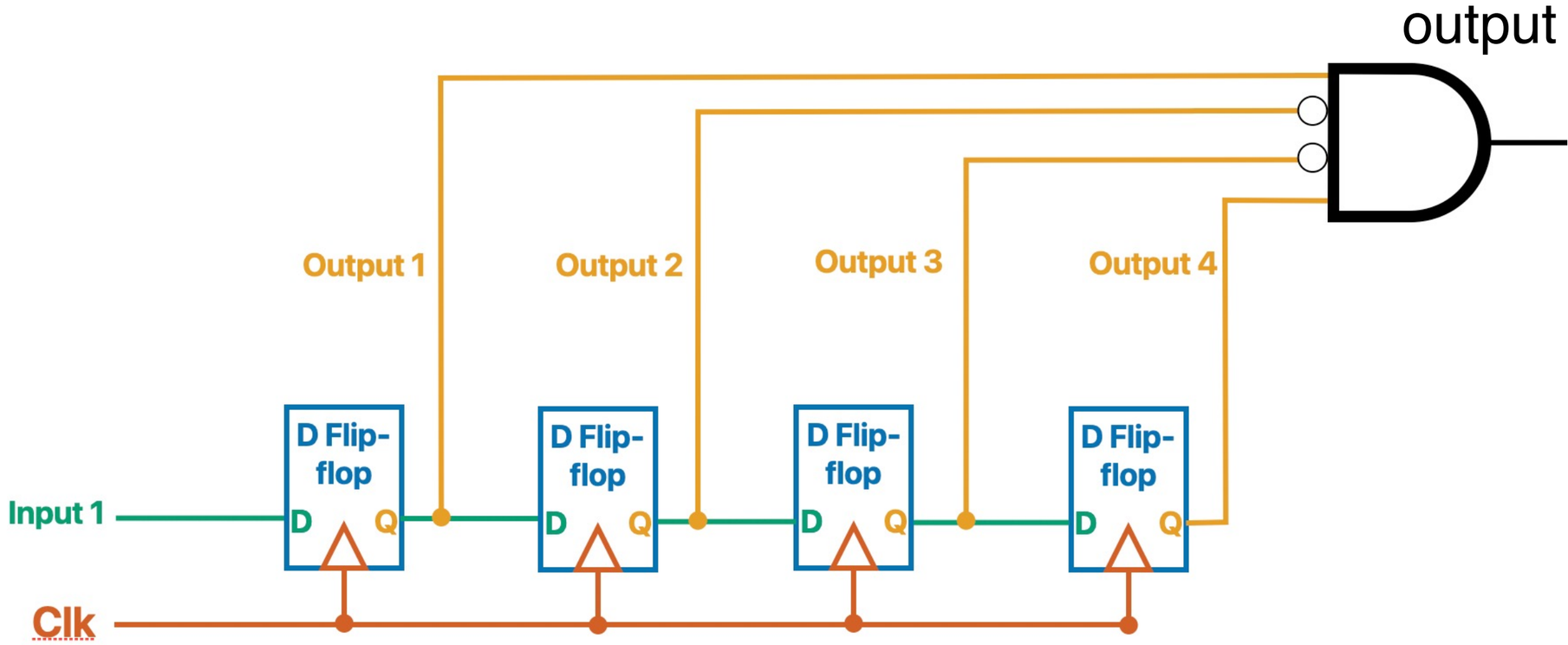



- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?

(1,1,0,1)

# Let's play with the shift register more…

- For the extended shift register, what sequence of input will circuit output "1" after 4 clock cycles?
  - A. (1, 1, 1, 1)
  - B. (0, 1, 0, 1)
  - C. (1, 0, 1, 0)
  - D. (0, 1, 1, 0)
  - E. (1, 0, 0, 1)



# Let's play with the shift register more…

- For the extended shift register, what sequence of input will the circuit output "1" after 4 clock cycles?

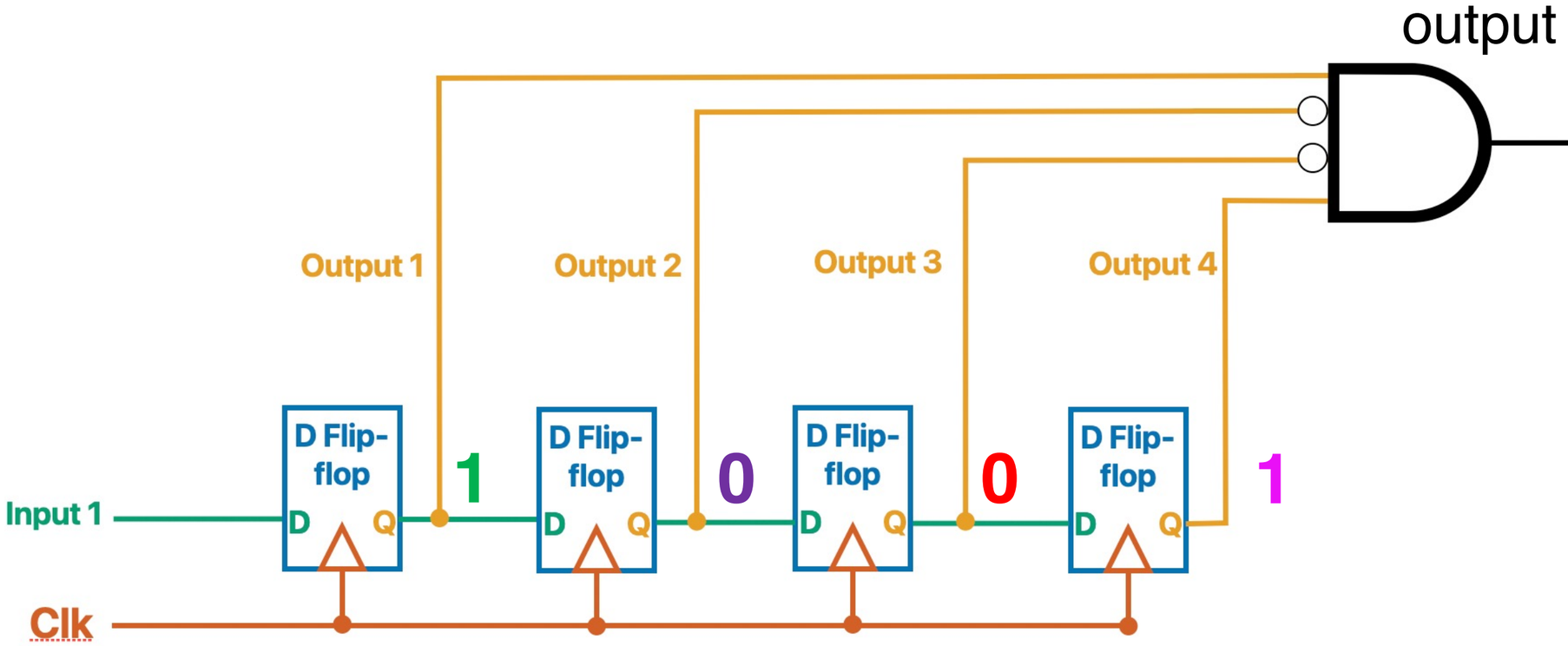
A. (1, 1, 1, 1)

B. (0, 1, 0, 1)

C. (1, 0, 1, 0)

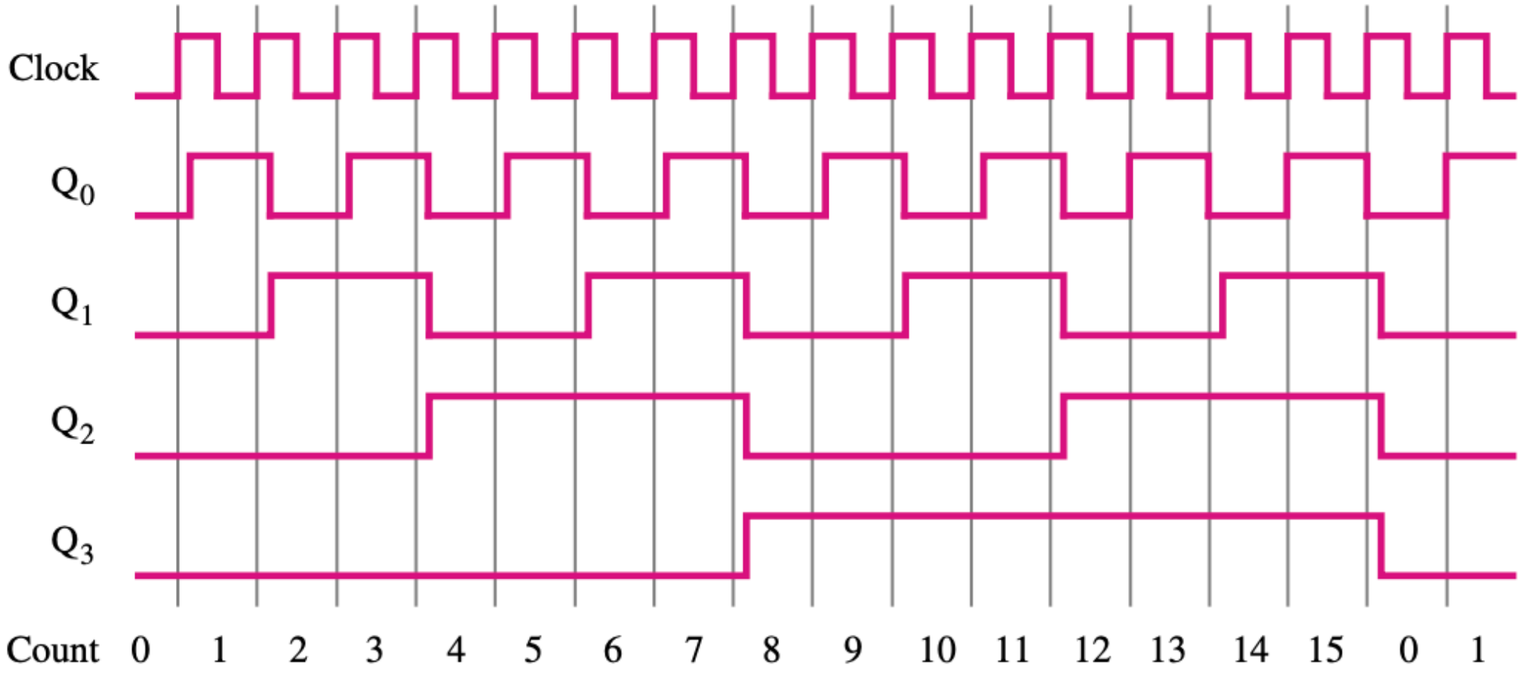
D. (0, 1, 1, 0)

E. (1, 0, 0, 1)



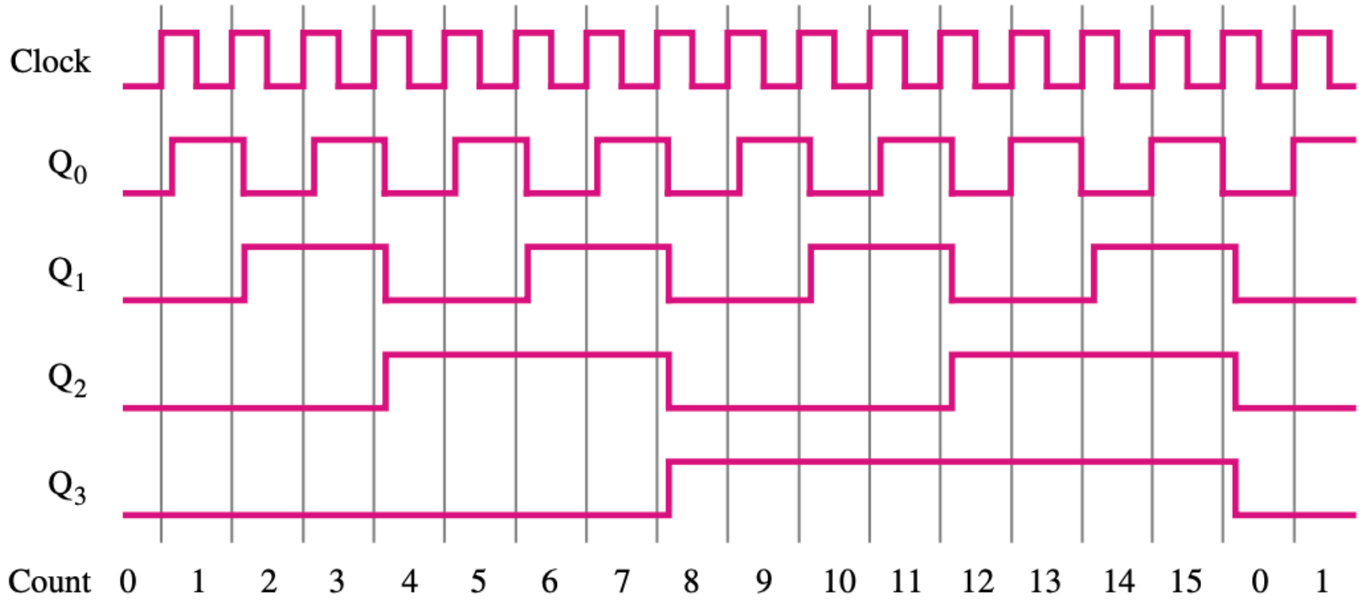
# 4-bit up-counter

Expectation:
Outputs $Q_3Q_2Q_1Q_0$ count in the sequence 0, 1, 2, . . . , 14, 15, 0, 1, and so on




Timing diagram

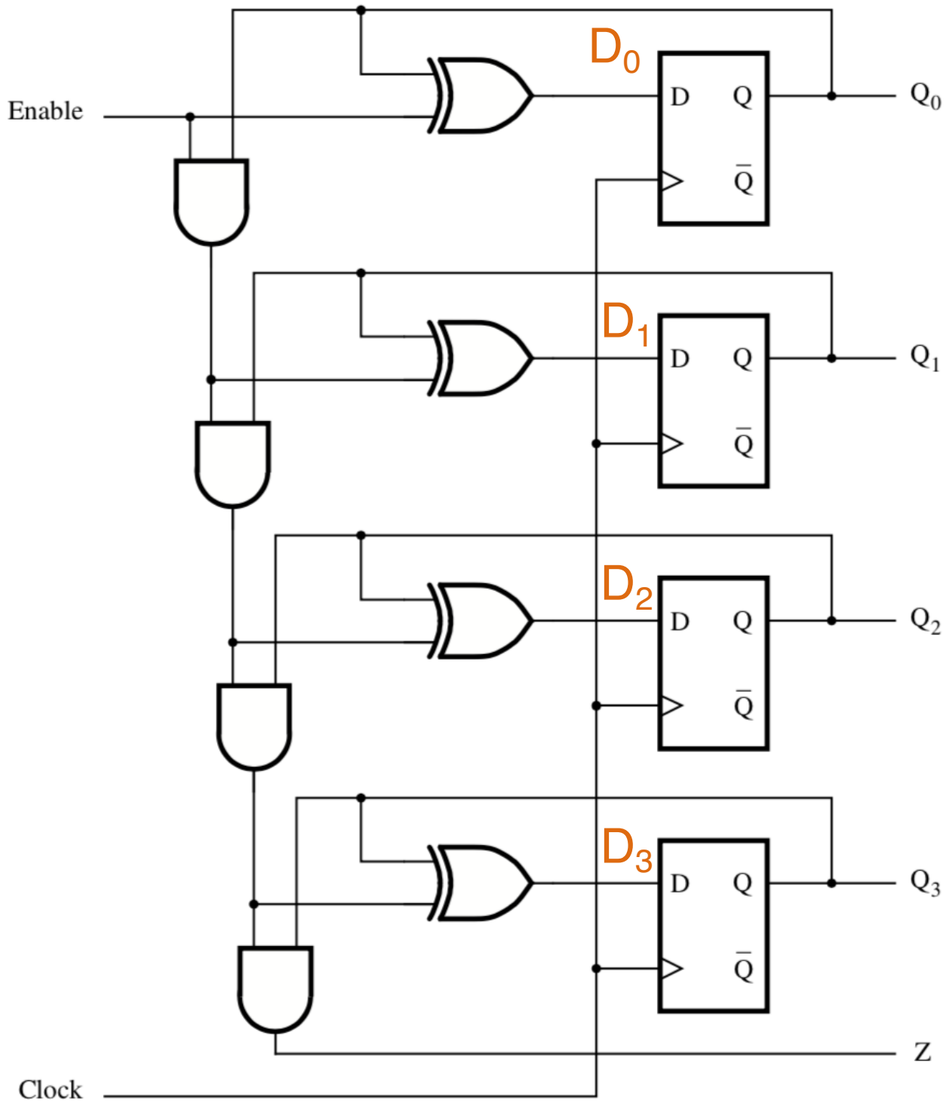
# 4-bit up-counter




| clock cycle | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 |

$Q_0$ always changes
$Q_1$ changes only when $Q_0 = 1$
$Q_2$ changes only when $Q_1 = Q_0 = 1$
$Q_3$ changes only when $Q_2 = Q_1 = Q_0 = 1$

# Up-counter




Counter counts the clock pulses only if *Enable* = 1.

$D_0 = Q_0 \oplus Enable$ ⟹ $Q_0(t+1) = \overline{Q_0}(t)$ if *Enable* = 1

$D_1 = Q_1 \oplus Q_0 \cdot Enable = Q_1 \oplus (Q_0 \cdot Enable)$

$D_2 = Q_2 \oplus Q_1 \cdot Q_0 \cdot Enable = Q_2 \oplus (Q_1 \cdot Q_0 \cdot Enable)$

$D_3 = Q_3 \oplus Q_2 \cdot Q_1 \cdot Q_0 \cdot Enable = Q_3 \oplus (Q_2 \cdot Q_1 \cdot Q_0 \cdot Enable)$

$Q_0$ always changes

$Q_1$ changes only when $Q_0 = 1$

$Q_2$ changes only when $Q_1 = Q_0 = 1$

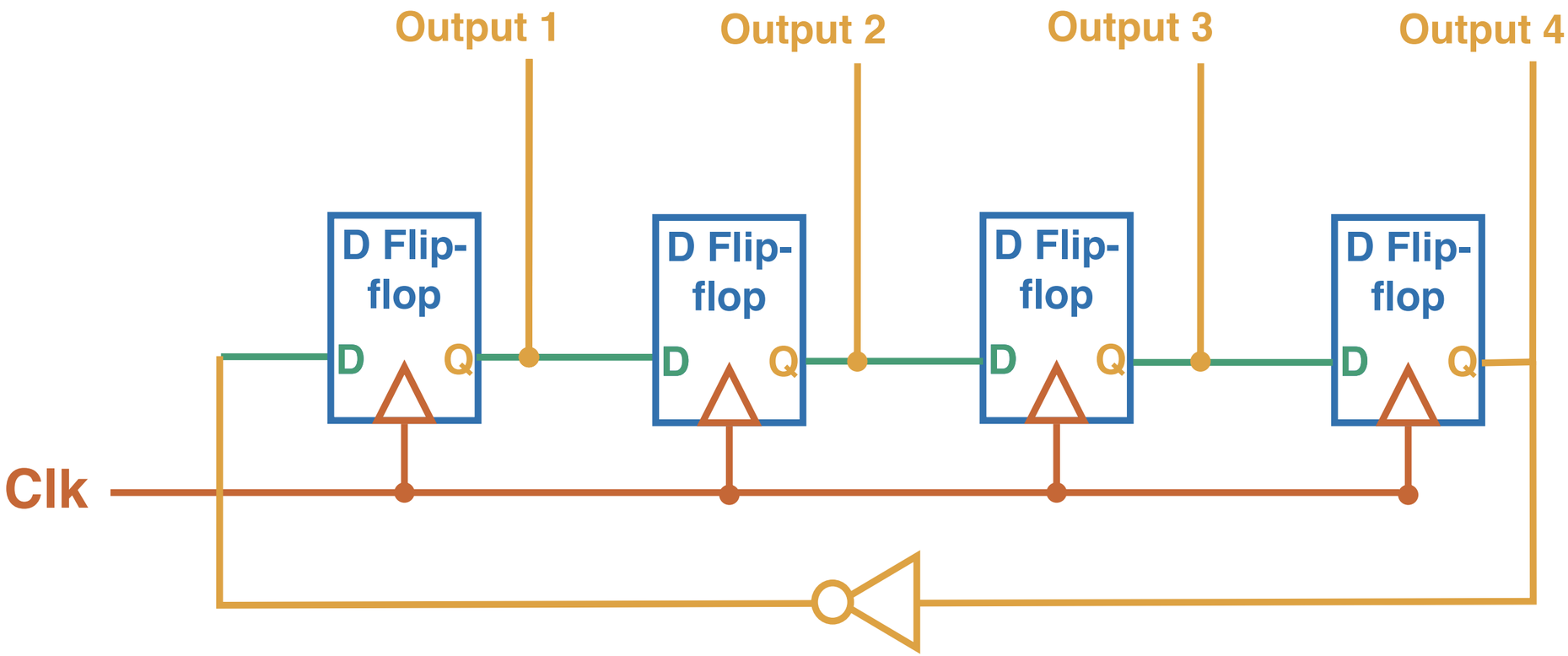
$Q_3$ changes only when $Q_2 = Q_1 = Q_0 = 1$

Z:

1) Status indicator, all 1s
2) Makes it easy for concatenating

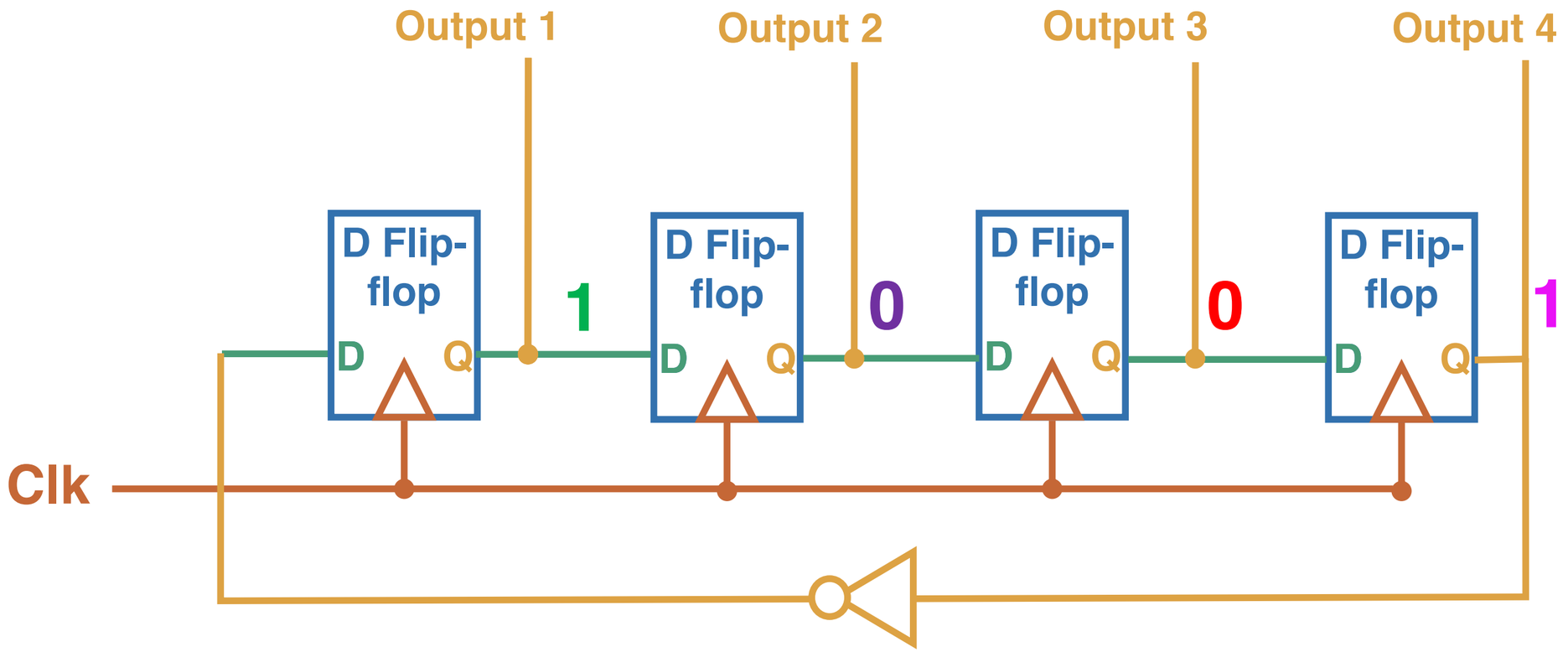
# Counters

- Sequences through a fixed set of patterns
- Note: definition is general
- For example, the one in the figure is a type of counter called Linear Feedback Shift Register (LFSR)



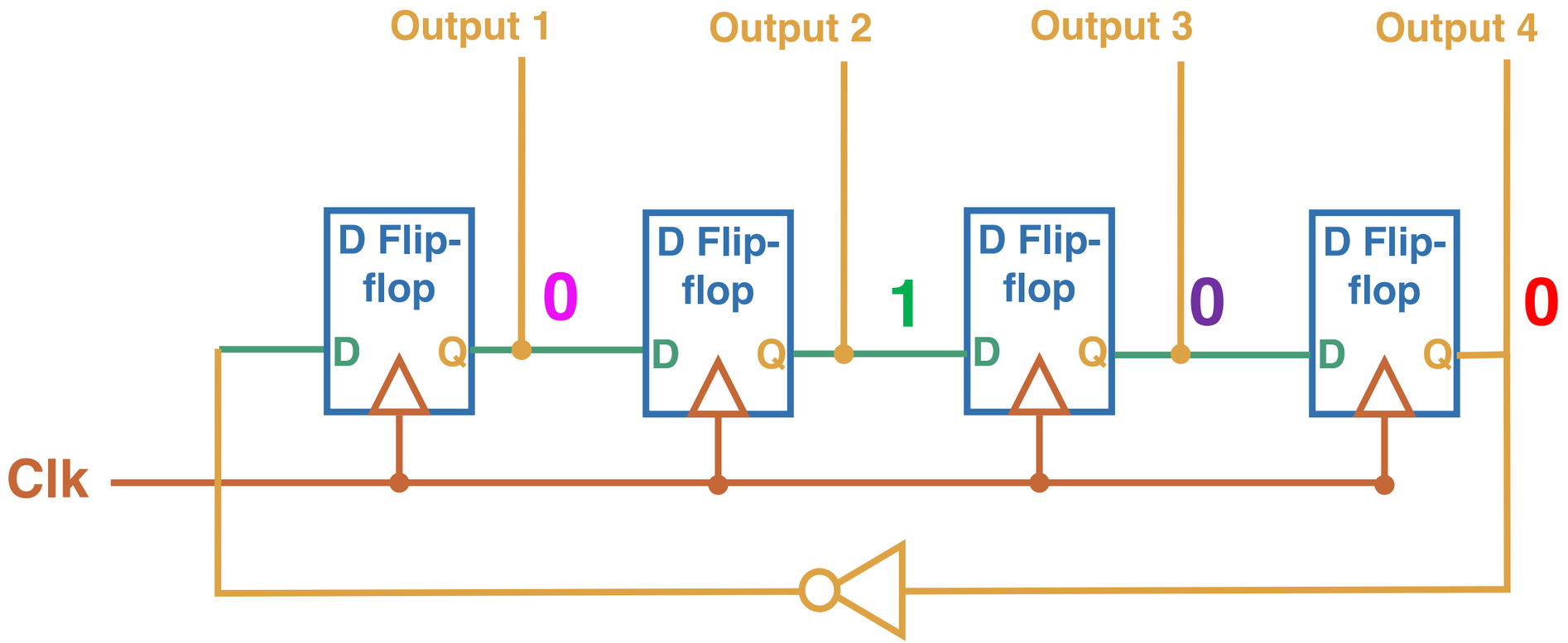
# Counters

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



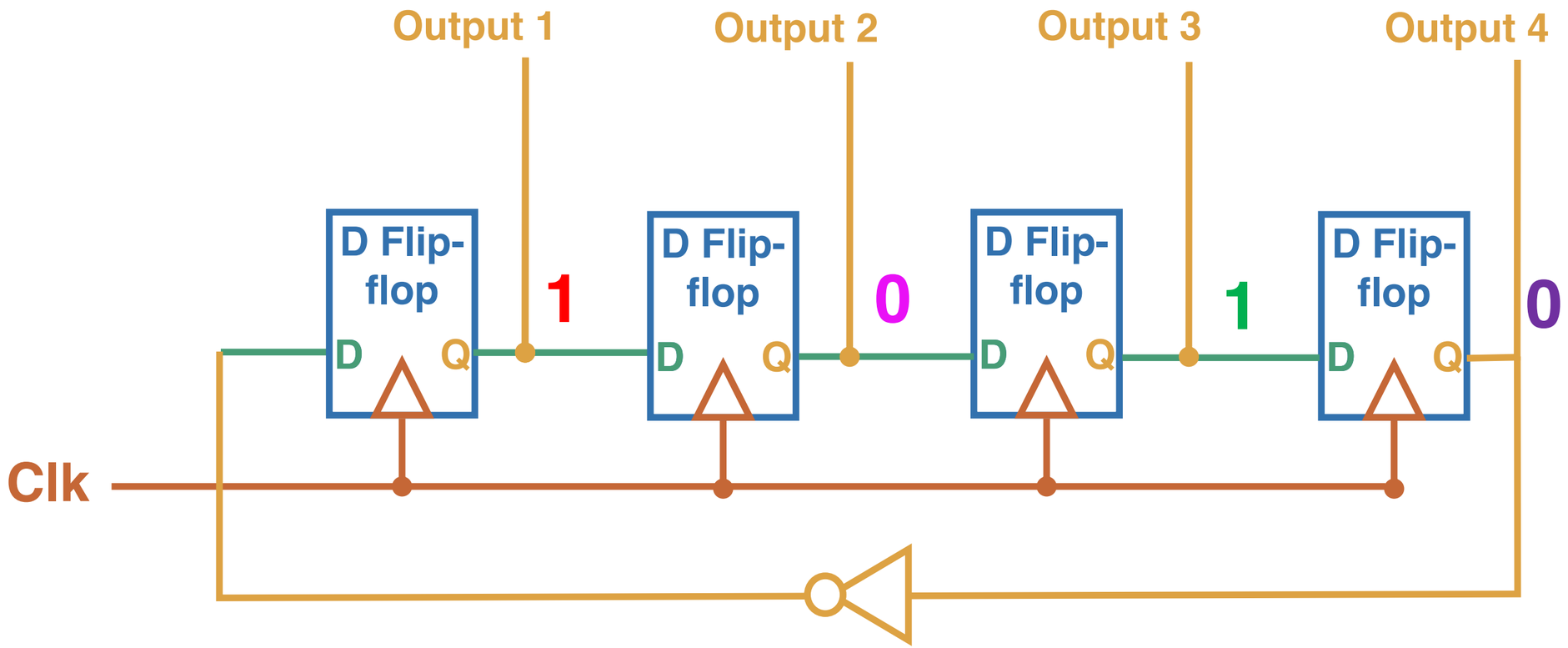
# After one clock cycle

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



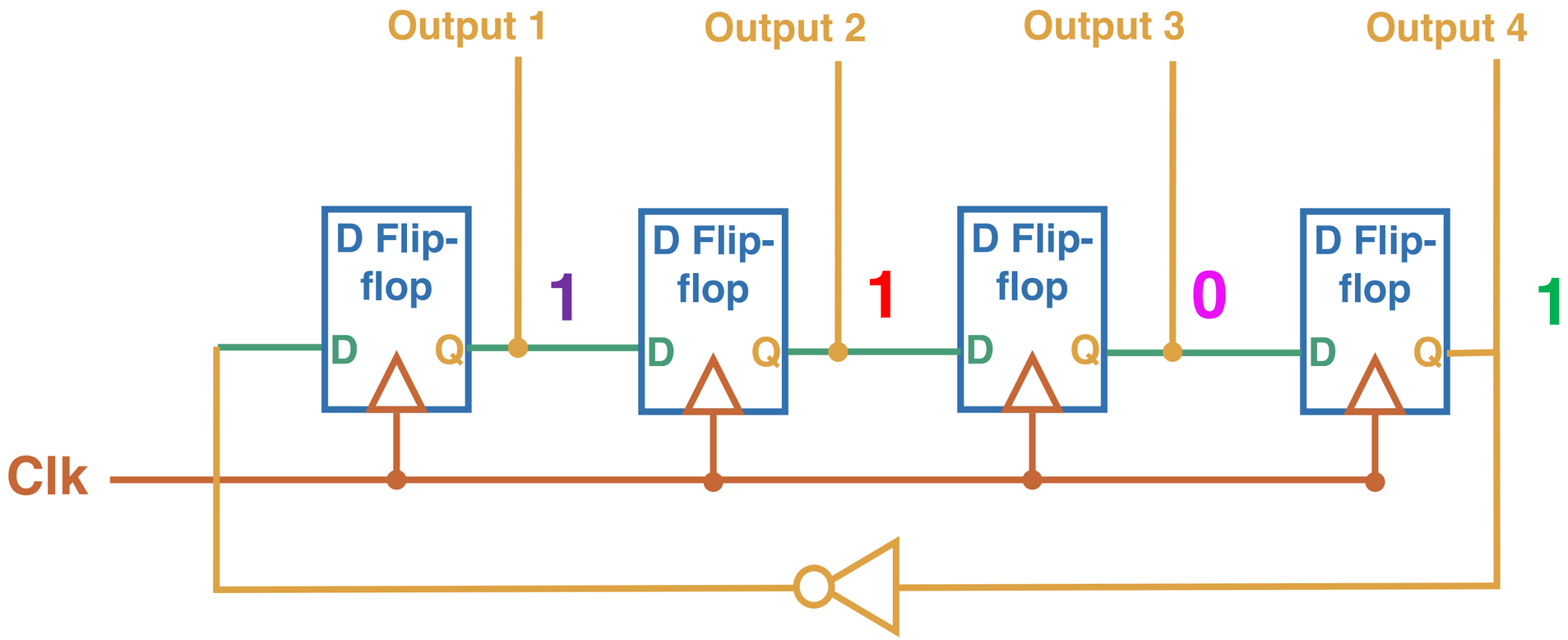
# After two clock cycles

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



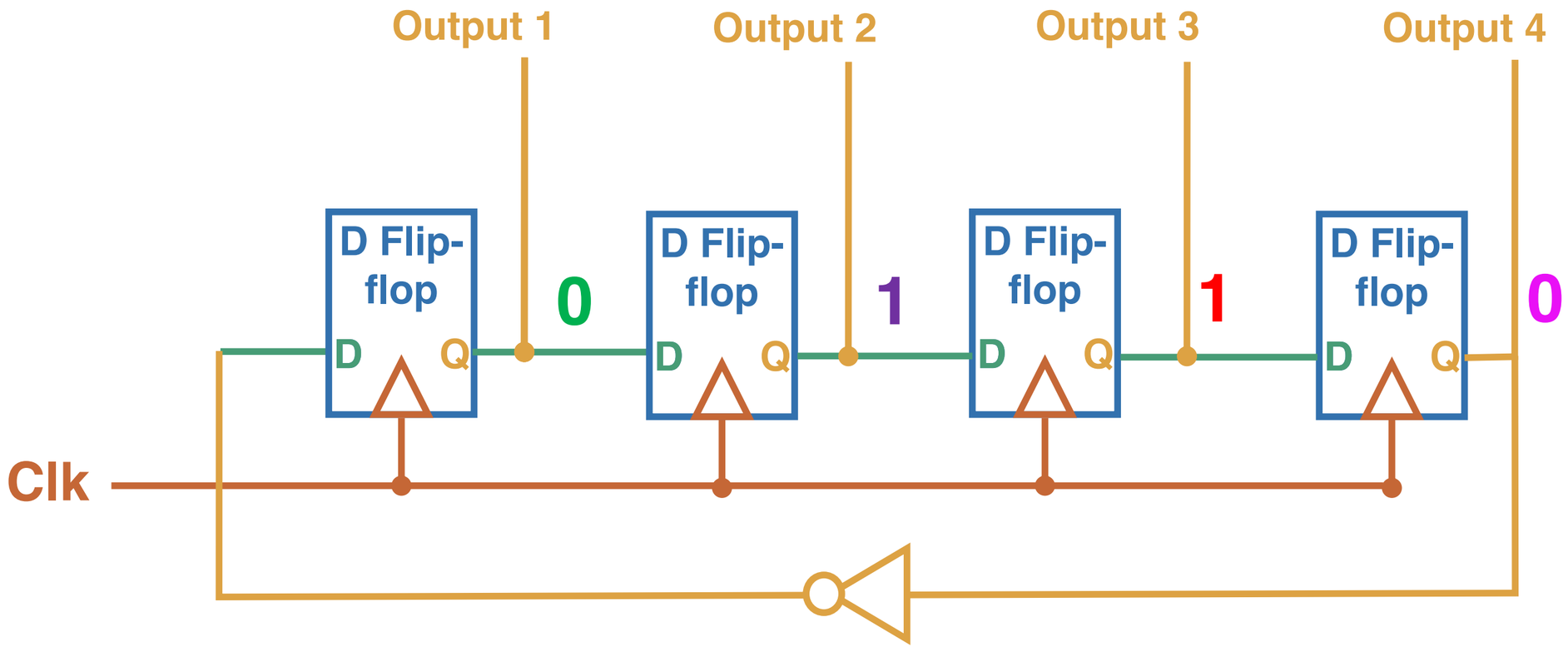
# After three clock cycles

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



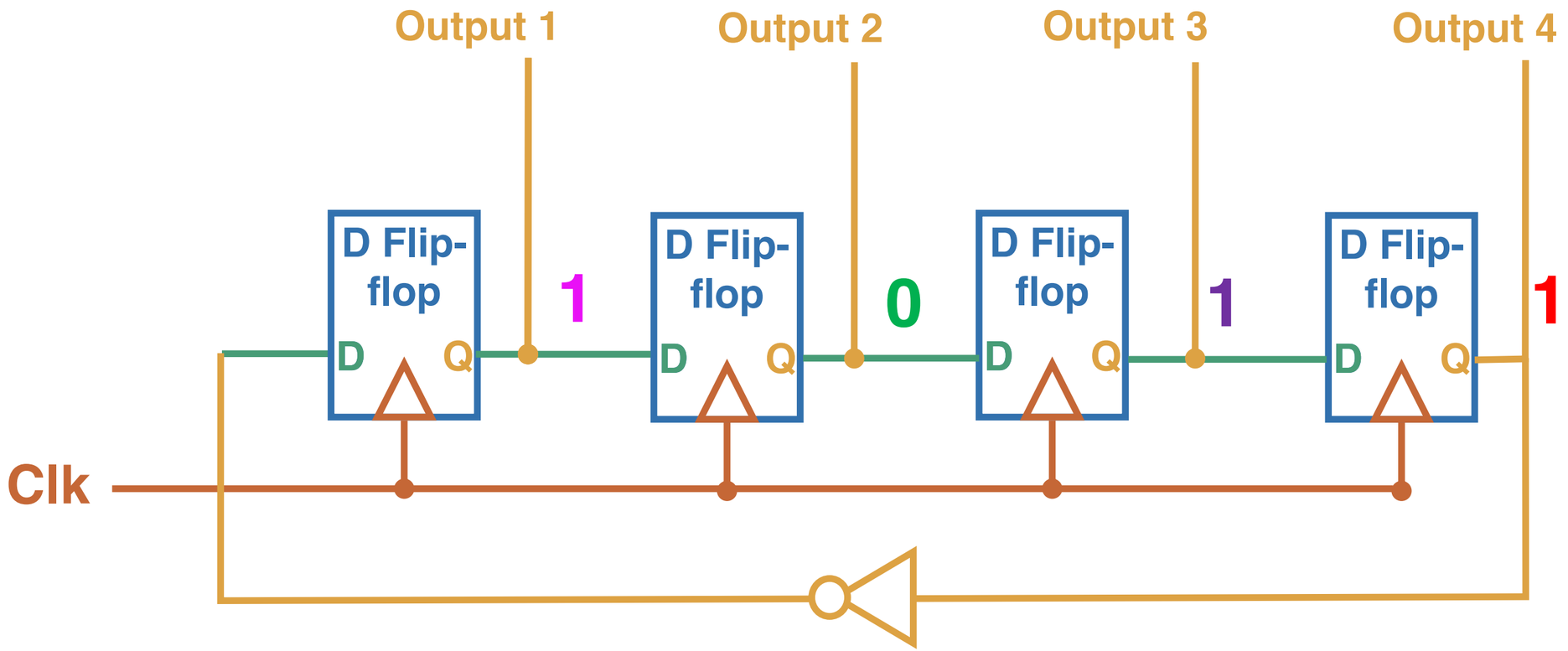
# After four clock cycles

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



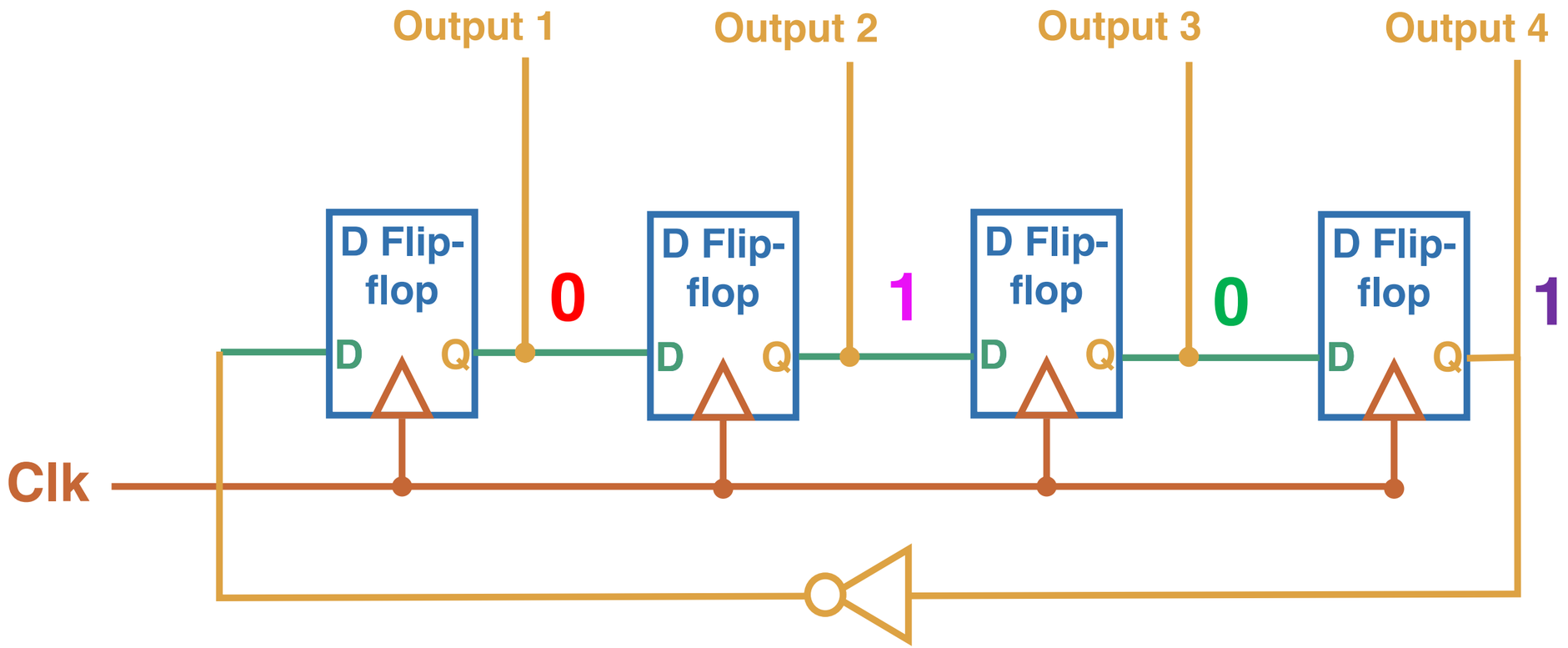
# After five clock cycles

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



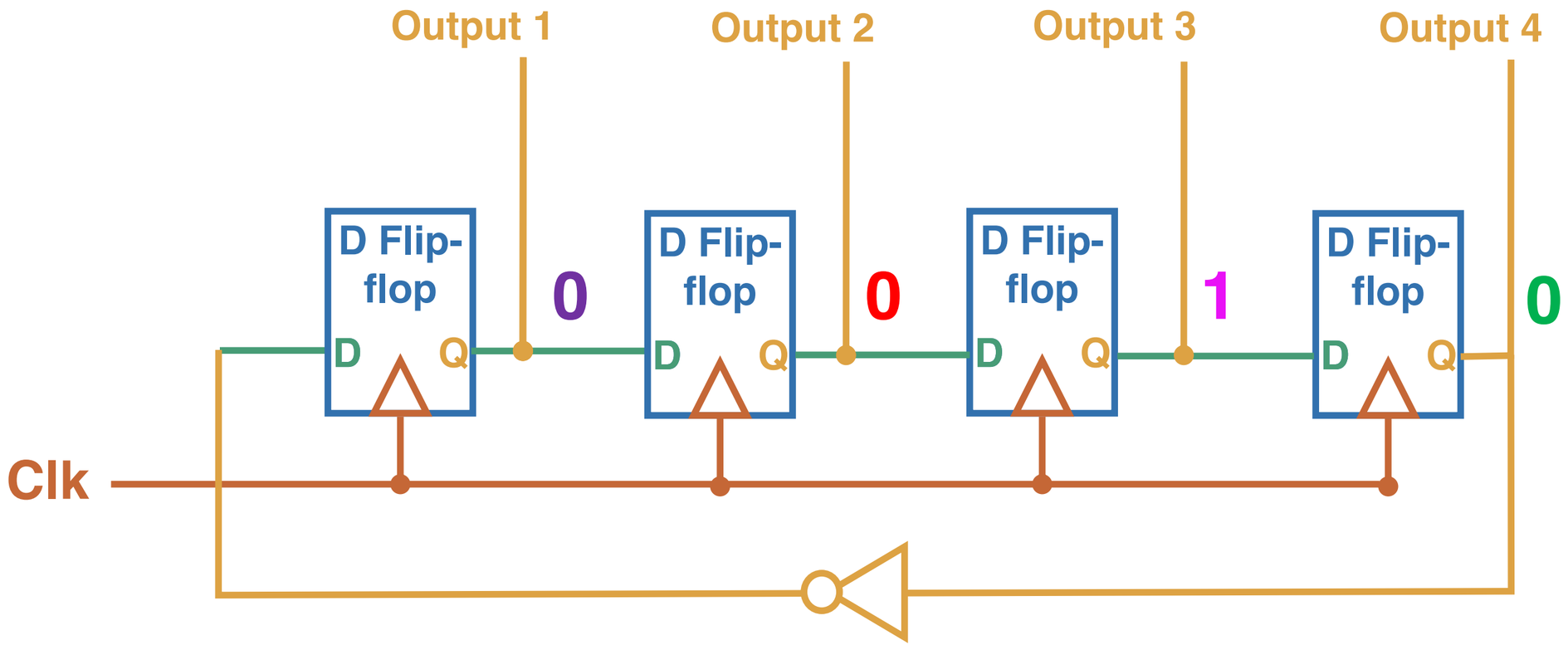
# After six clock cycles

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



# After seven clock cycles

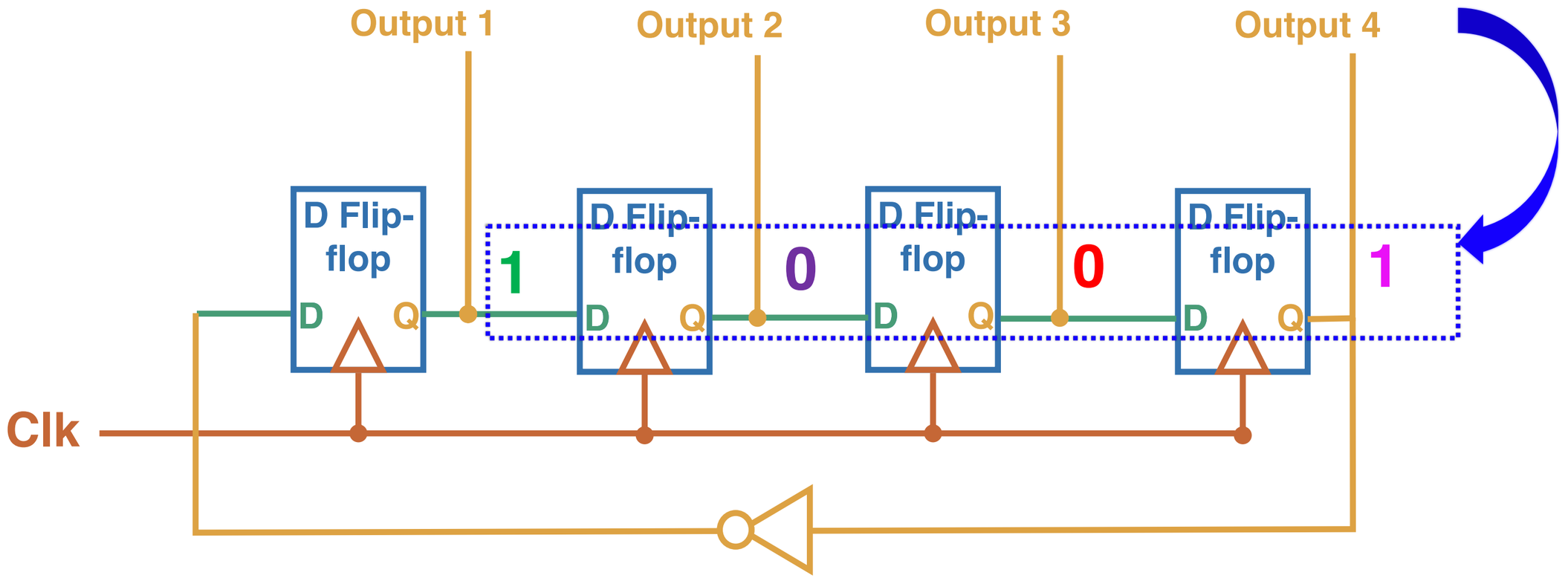
- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1



# After eight clock cycles

- Assume that initially, O1 = 1, O2 = 0, O3 = 0, O4 = 1

Back to the initial status!!



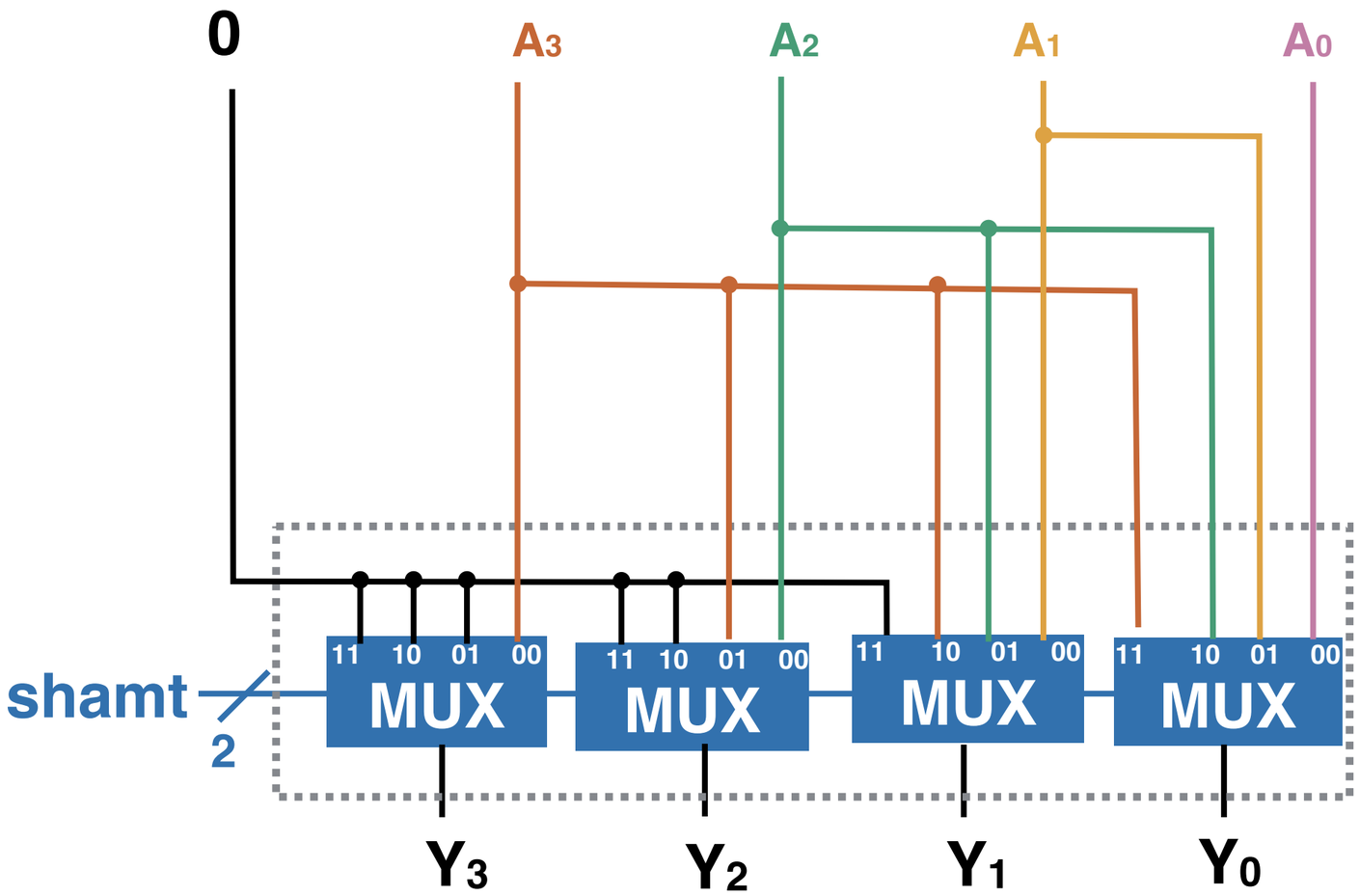
# Shifters

# Shifters

- **Logical shifter**: shifts value to left or right and fills empty spaces with 0's
  - 11001 >> 2 = 00110 (right)
  - 11001 << 2 = 00100 (left)
- **Arithmetic shifter**: same as logical shifter, but on right shift, fills empty spaces with the old most significant bit
  - Ex: 11001 >>> 2 = 11110 (right)
  - Ex: 11001 <<< 2 = 00100 (left)
- **Rotator**: rotates bits in a circle, such that bits shifted off one end are shifted into the other end
  - Ex: 11001 ROR 2 = 01110 (right)
  - Ex: 11001 ROL 2 = 00111 (left)

https://en.wikipedia.org/wiki/Circular_shift

# Shift “Right”



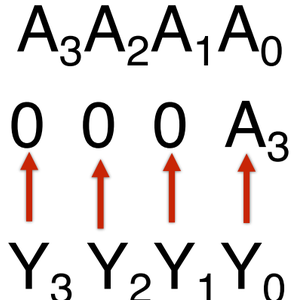

Example:
if S = 11
then
Y3 = 0
Y2 = 0
Y1 = 0
Y0 = A3

Example:
if S = 10
then
Y3 = 0
Y2 = 0
Y1 = A3
Y0 = A2

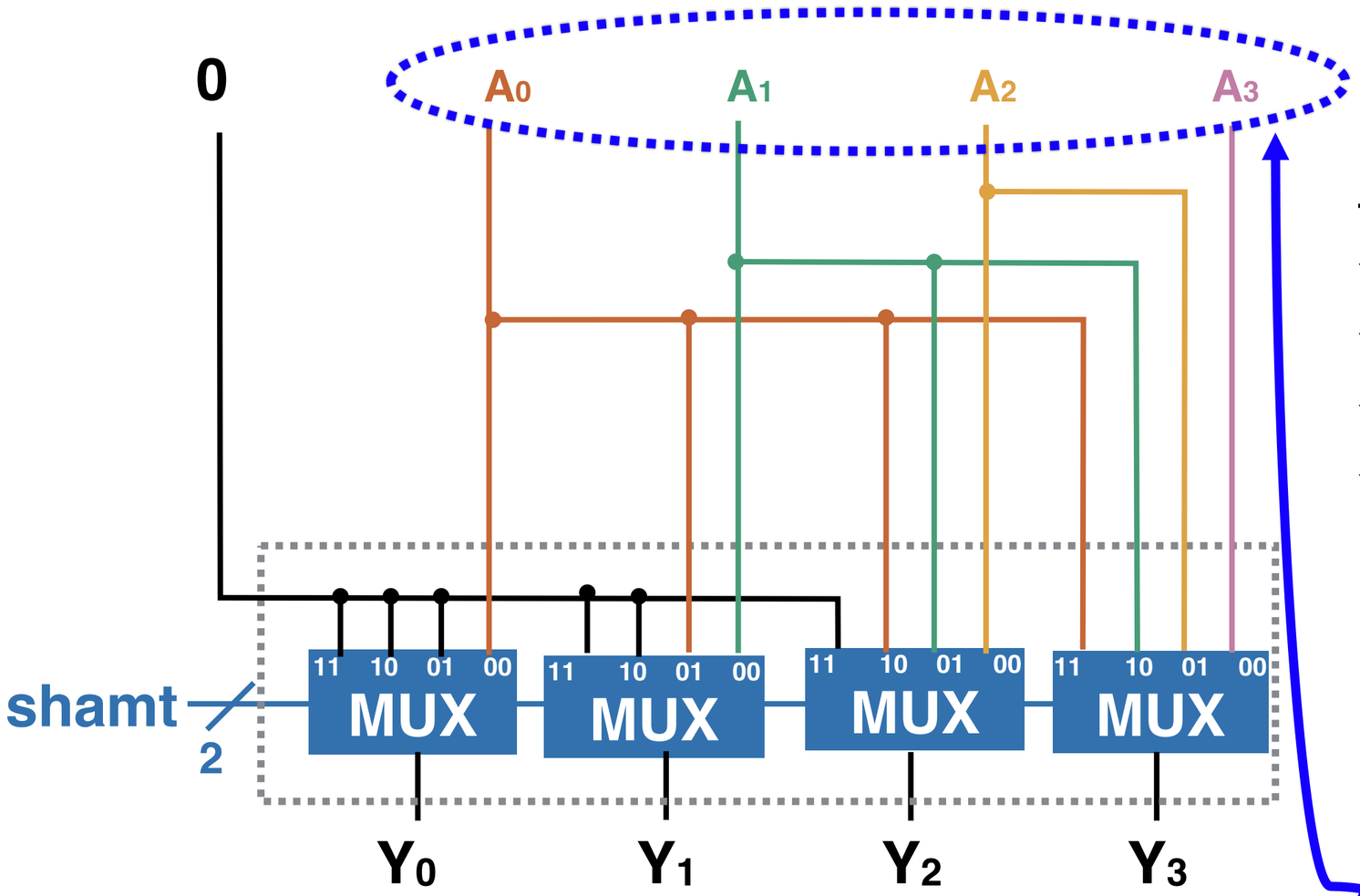
**The “chain” of multiplexers determines how many bits to shift**

**Based on the value of the selection input (shamt = shift amount)**

If S=11, shift right by 3 bits



# Shift “left"




Example:
if S = 01
then
Y3 = A2
Y2 = A1
Y1 = A0
Y0 = 0

Example:
if S = 10
then
Y3 = A1
Y2 = A0
Y1 = 0
Y0 = 0

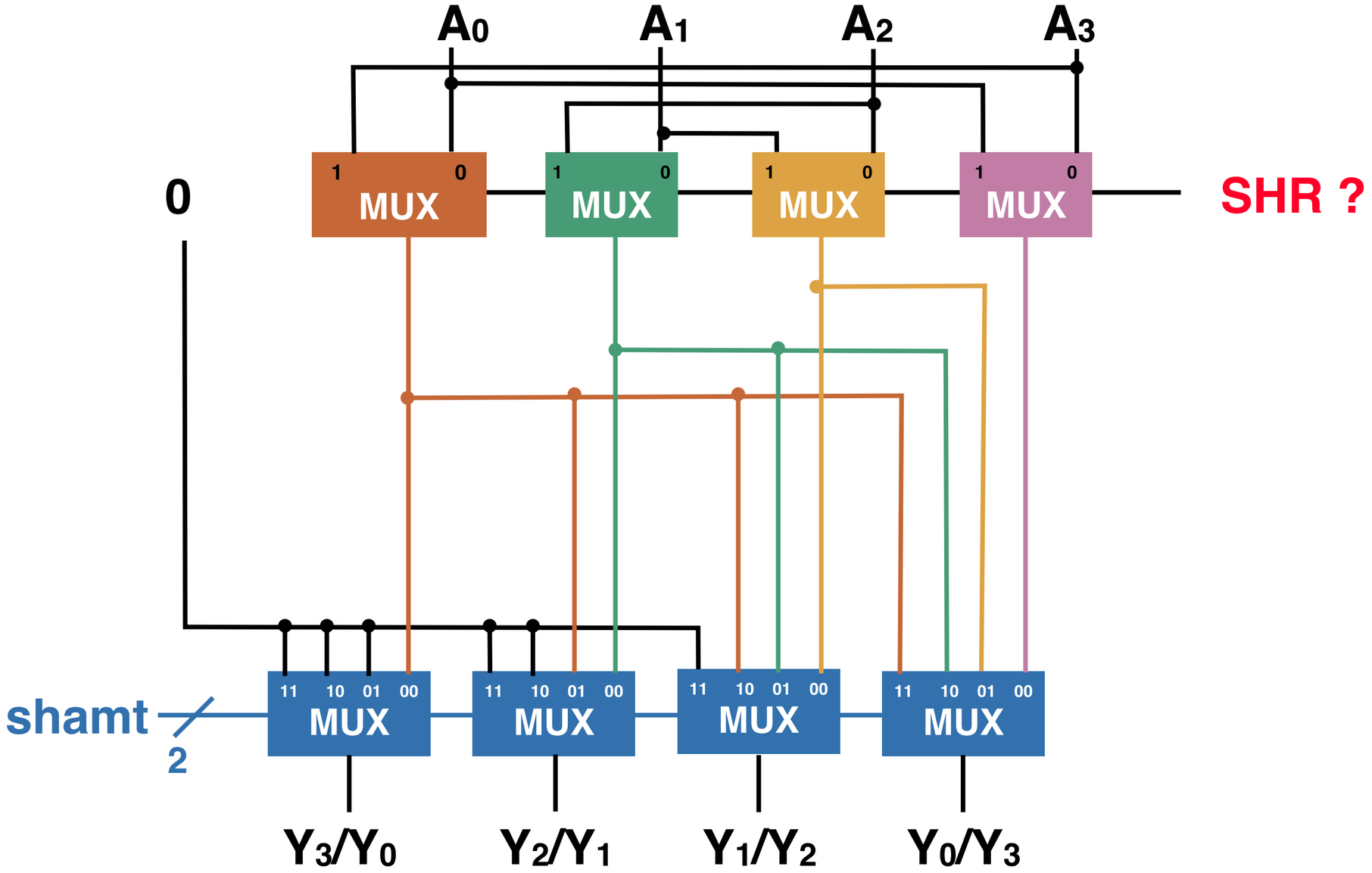
Example:
if S = 11
then
Y3 = A0
Y2 = 0
Y1 = 0
Y0 = 0

**The “chain” of multiplexers determines how many bits to shift**

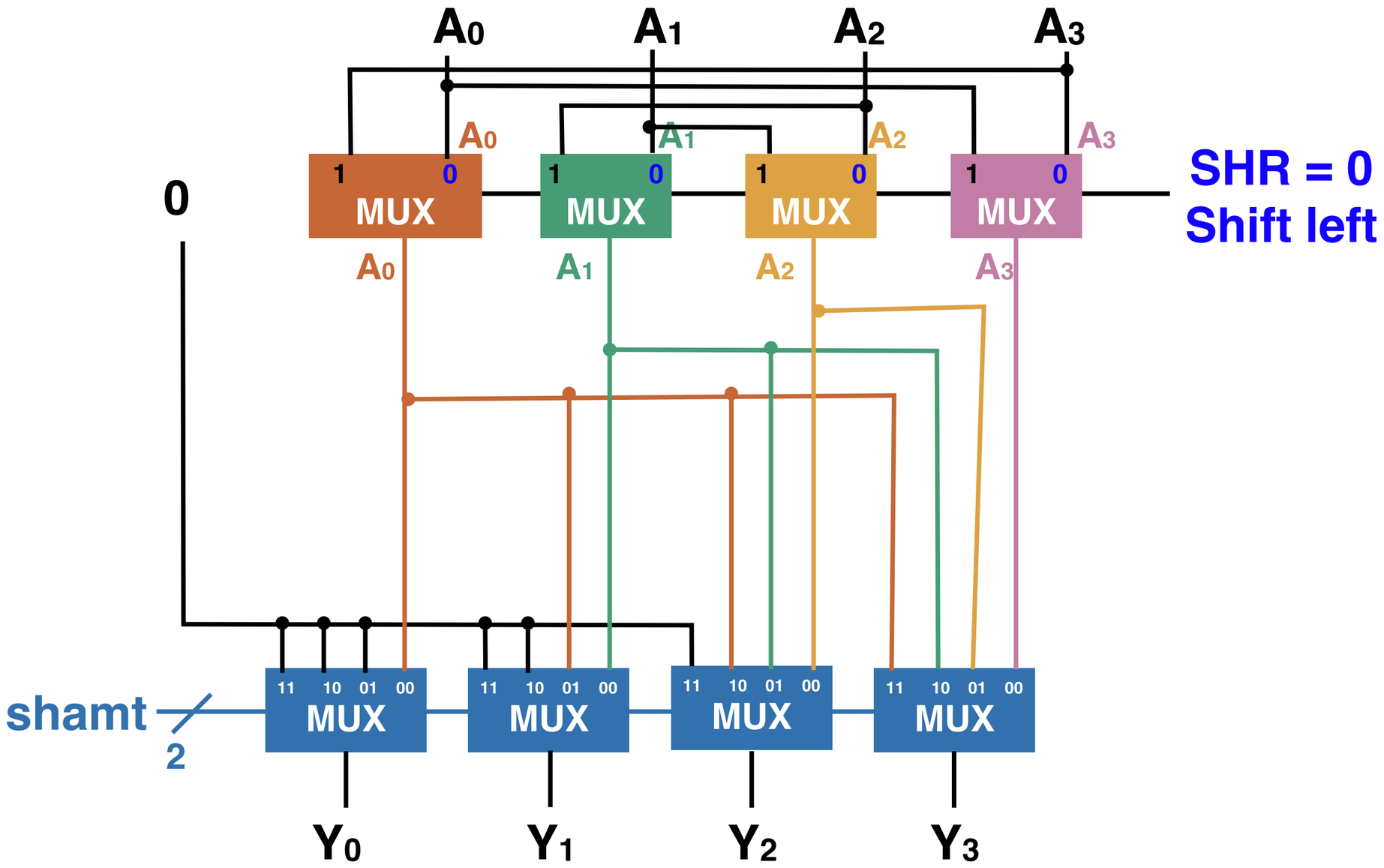
**Based on the value of the selection input (shamt = shift amount)**

We don’t need to modify the circuit, just change the order of inputs!

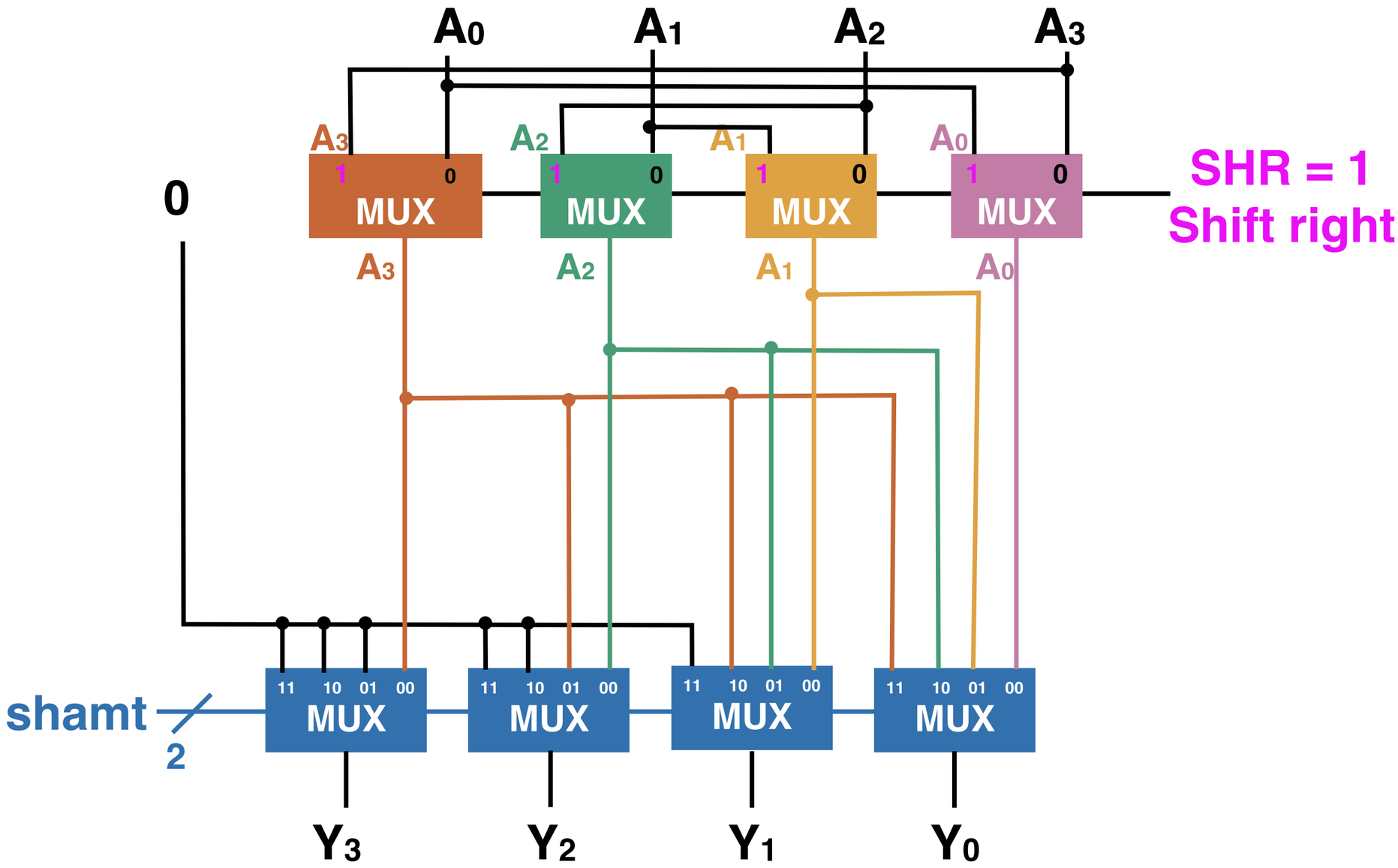
# Generic Shifter



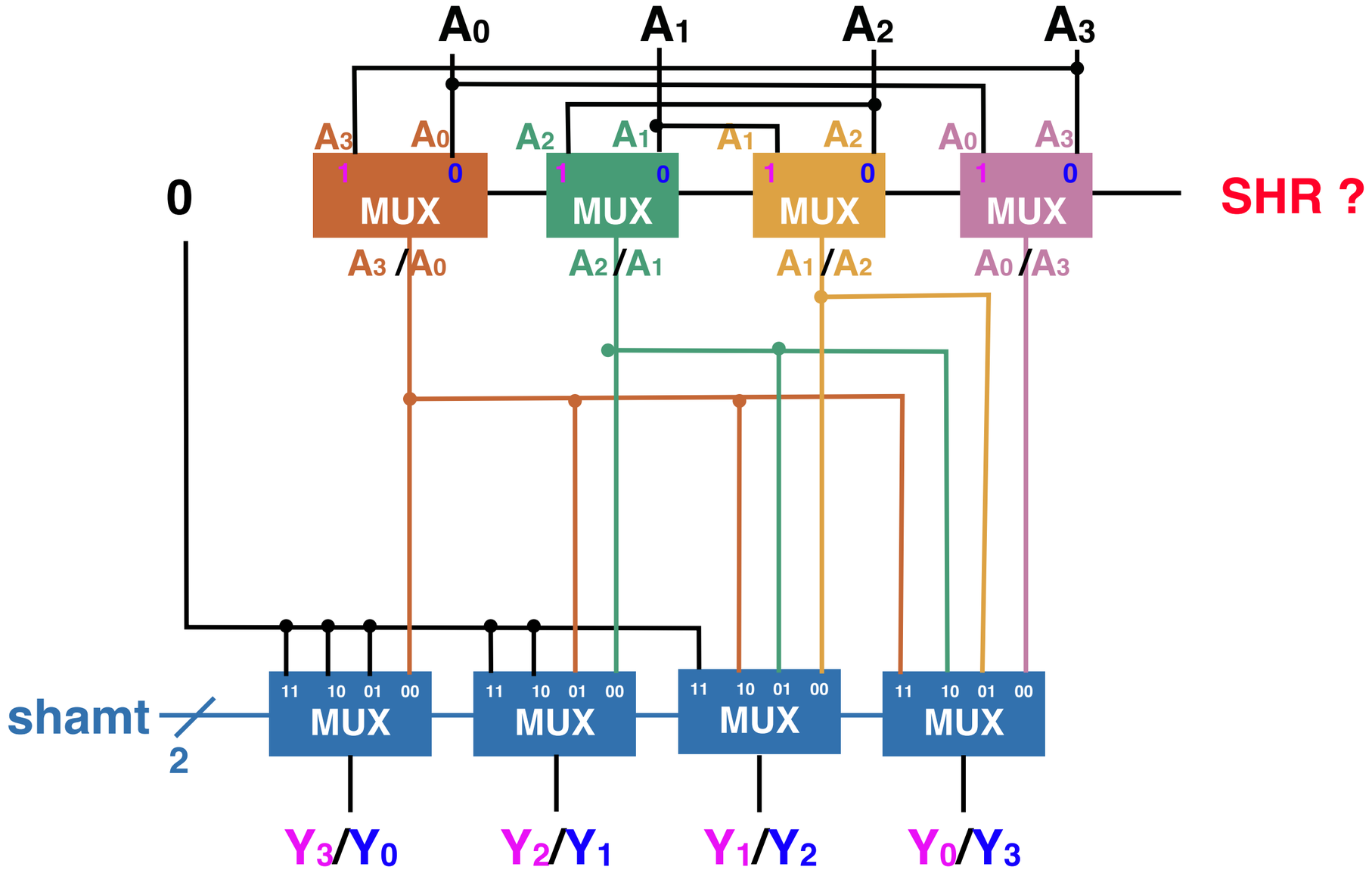
# Generic Shifter



# Generic Shifter

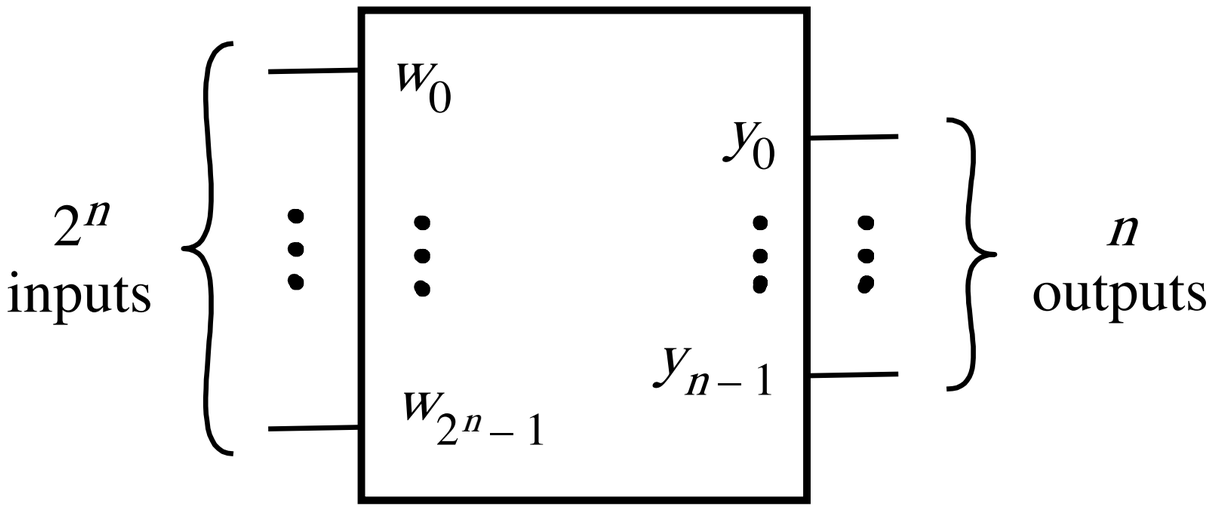


# Generic Shifter



# Encoder

# Binary Encoders




A $2^n$-to-$n$ binary encoder.

Exactly one of input signals should have a value of 1 ( one-hot coded), and the outputs present the binary number that identifies which input is equal to 1.

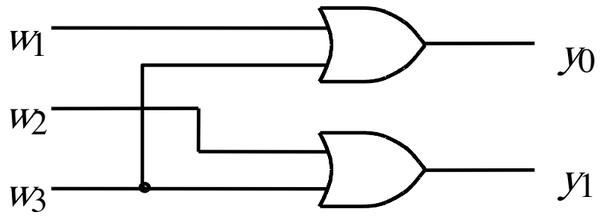
# Binary Encoders

| $w_3$ | $w_2$ | $w_1$ | $w_0$ | $y_1$ | $y_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

(a) Truth table (partial)

Incompletely specified function
(12 don't care conditions)

K-maps for $y_1$ and $y_0$




(b) Circuit

4-to-2 binary encoder

Encoders can be used for efficient data communication and storage, since $2^n$ bits are compressed to n bits in the process.

# Comparator

# Comparator

- A comparator compares the relative values of two binary numbers.
- If we design a comparator comparing two 4-bit unsigned numbers using a truth table, we need $2^8 = 256$ rows.
- We actually can compare the two numbers bit by bit.

$A = a_3a_2a_1a_0$ unsigned

$B = b_3b_2b_1b_0$ unsigned

Define $i_k=(b_k \oplus a_k)'$ ------ $i_k$ gives a 1 if $b_k = a_k$

Then

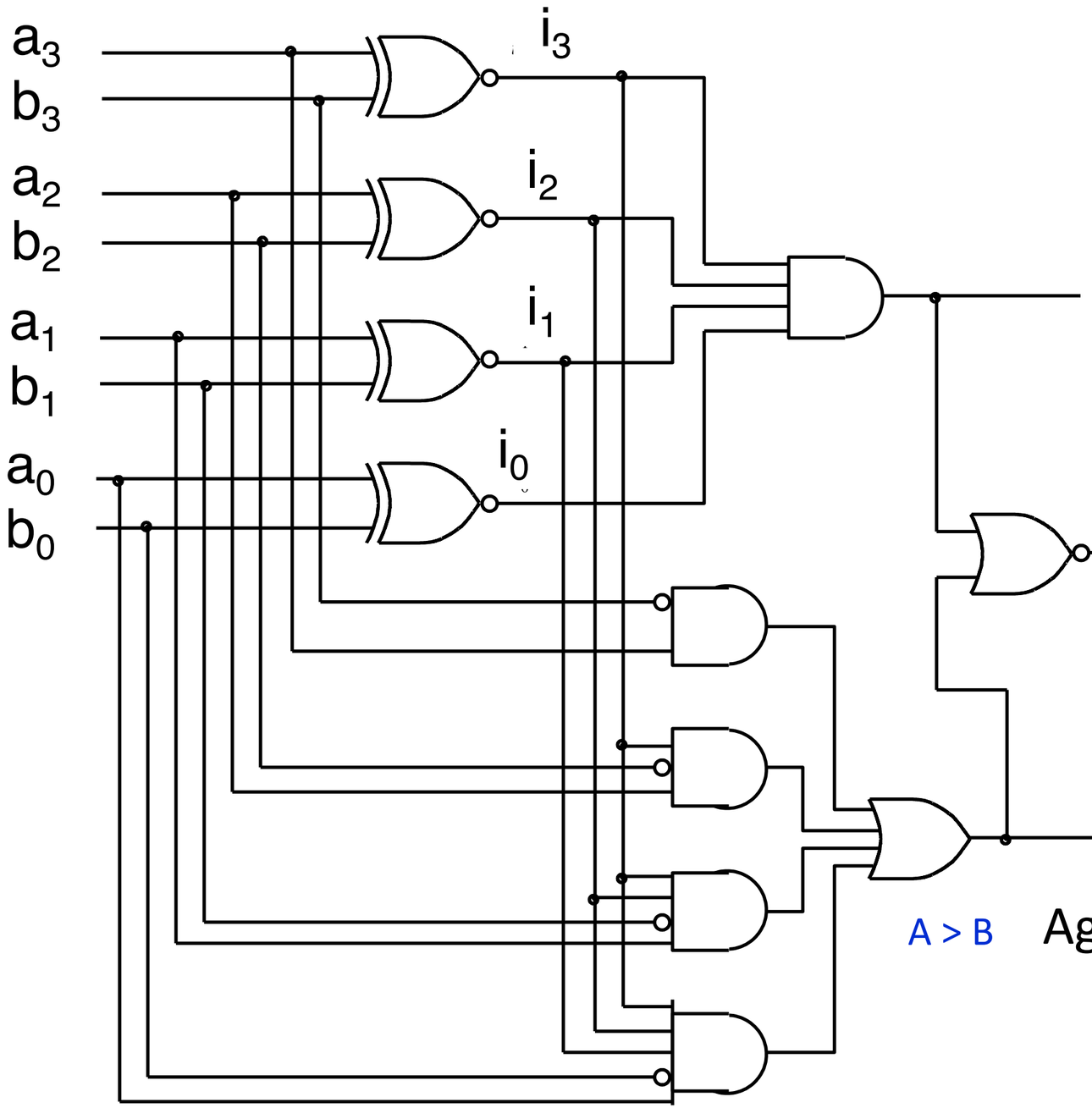
A = B $\quad AeqB = i_3i_2i_1i_0$

A > B $\quad AgtB = a_3b_3' + i_3a_2b_2' + i_3i_2a_1b_1' + i_3i_2i_1a_0b_0'$

A < B $\quad AltB = (AgtB + AeqB)'$

Note that we can compare the bits of A and B at the same position from left to right. When we found a difference, the comparison is done.

# Comparator

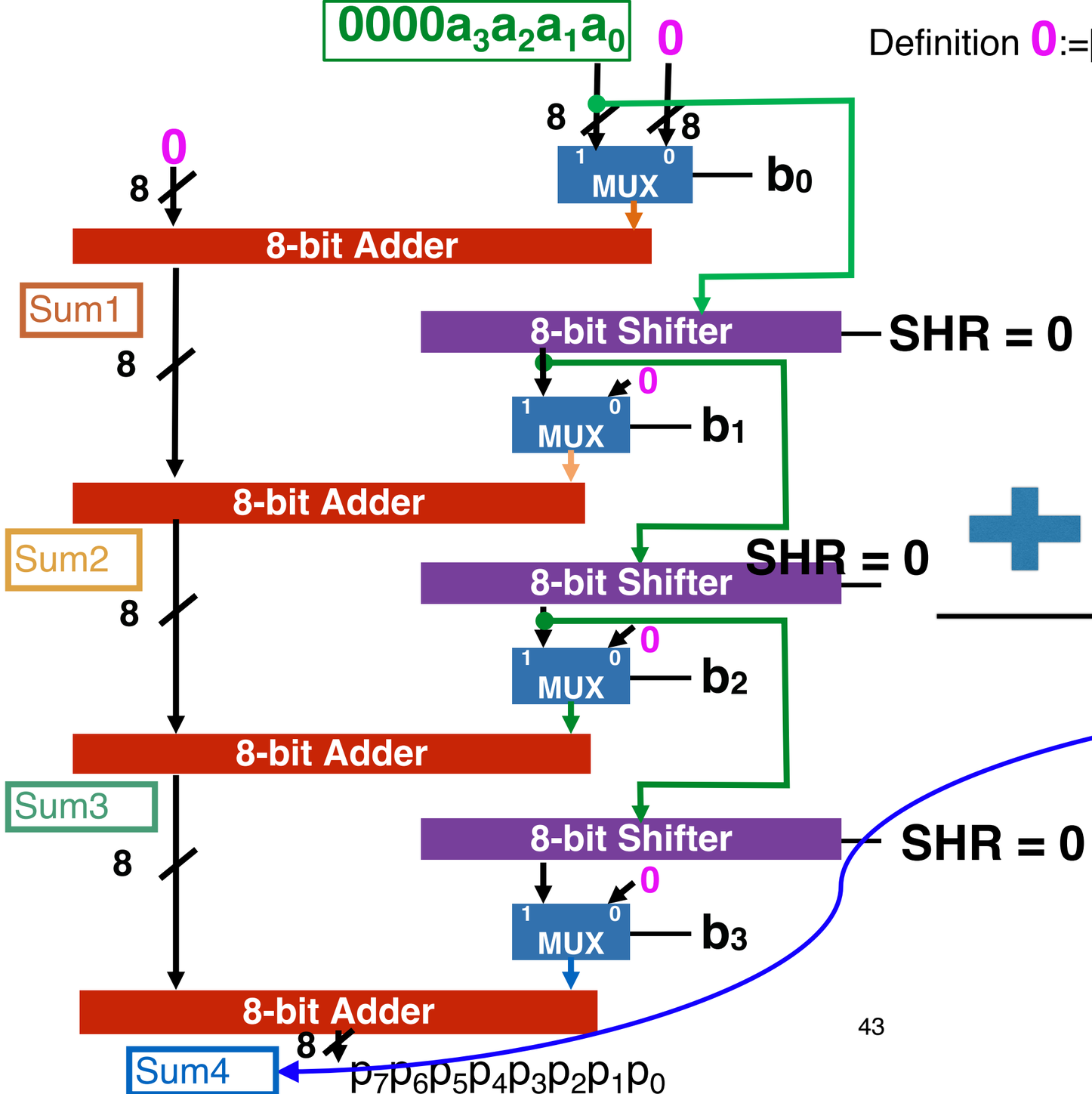



A = B $AeqB = i_3 i_2 i_1 i_0$

A < B $AltB = (AgtB + AeqB)'$

A > B $AgtB = a_3 b_3' + i_3 a_2 b_2' + i_3 i_2 a_1 b_1' + i_3 i_2 i_1 a_0 b_0'$

# Multiplier




# Shift and add

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
| | | | | × $b_3$ | $b_2$ | $b_1$ | $b_0$ |
| 0 | 0 | 0 | 0 | $a_3b_0$ | $a_2b_0$ | $a_1b_0$ | $a_0b_0$ |
| 0 | 0 | 0 | $a_3b_1$ | $a_2b_1$ | $a_1b_1$ | $a_0b_1$ | 0 |
| 0 | 0 | $a_3b_2$ | $a_2b_2$ | $a_1b_2$ | $a_0b_2$ | 0 | 0 |
| 0 | $a_3b_3$ | $a_2b_3$ | $a_1b_3$ | $a_0b_3$ | 0 | 0 | 0 |
| $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

Sum4

# Divider

# Division of positive binary numbers

- Repeated subtraction
  - Set quotient to 0
  - Repeat while (dividend >= divisor)
    - Subtract divisor from dividend
    - Add 1 to quotient
  - When dividend < divisor:
    - Reminder = dividend
    - Quotient is correct