

# EEE104 – Digital Electronics (I)

## Lecture 20

A Revision Class

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

# Binary Numbers

The weighting structure

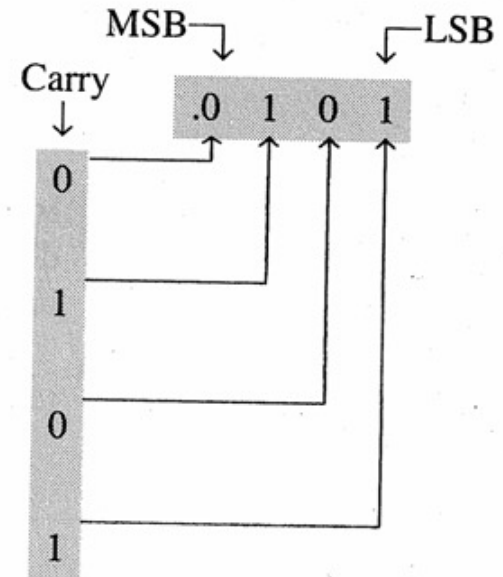
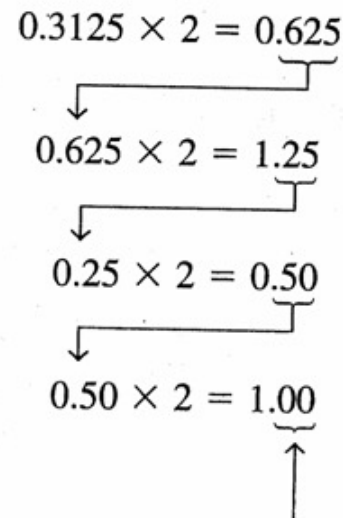
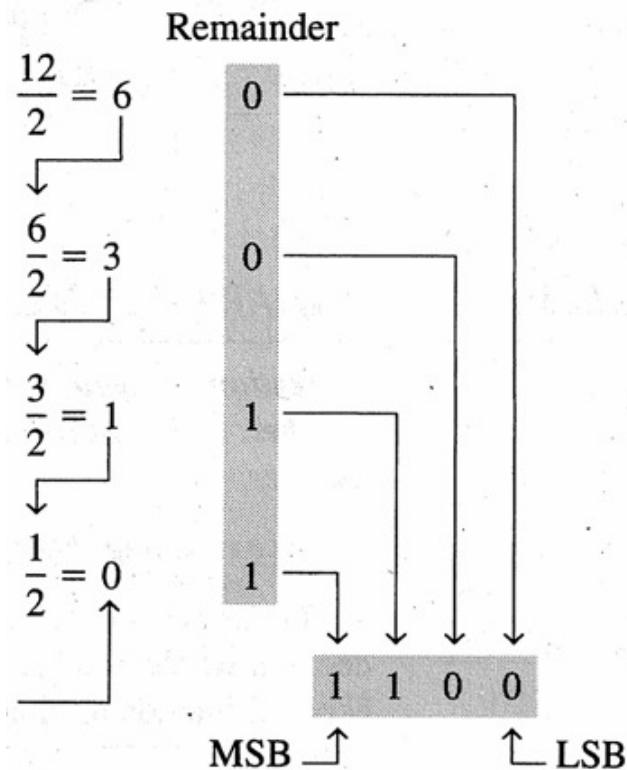
$$2^{n-1} \dots 2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2} \dots 2^{-n}$$

↑ Binary point

- To convert binary to decimal, sum the weights of all bits that are 1.

# Decimal-to-Binary Conversion

- The sum-of-weights method
- Repeated division-by-2 method for whole numbers
- Repeated multiplication by 2 for fractions



# Hexadecimal Numbers and BCD

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

- A BCD digit varies from 0000 to 1001.

- If a 4-bit sum is greater than 9 or generates a carry, add 6 (0110) to the sum.

9	1001	9	1001
+ 4	+ 0100	+ 9	+ 1001
13	1101	18	1 0010
	+ 0110		+ 0110
0001	0011	0001	1000
↓	↓	↓	↓
1	3	1	8

# Signed Numbers

- In the 2's complement system, a positive number is represented in the original form; a negative number is the 2's complements of the corresponding positive number.
- The weight of the sign bit is *negative*.
- In binary addition, **discard any final carry bit.**

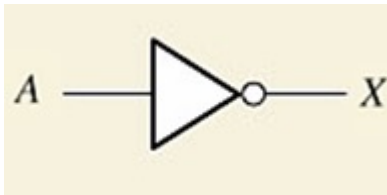
# Binary Numbers

## Pitfalls

- Forget to add 0110 in BCD addition.
- Add 0110 more than once in BCD addition.
- Forget to discard the sign bit in the result of signed number addition.
- Only the final result is given. The process leading to the solution is missing.
- For binary addition, carry out the decimal addition and convert the result back to binary.
- Forget to add one in the calculation of 2's complements.

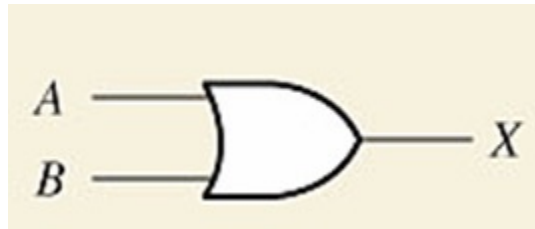
# Logic Gates

## 1. Inverter



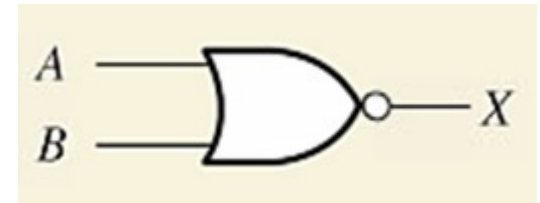
$$X = \bar{A}$$

## 3. OR Gate



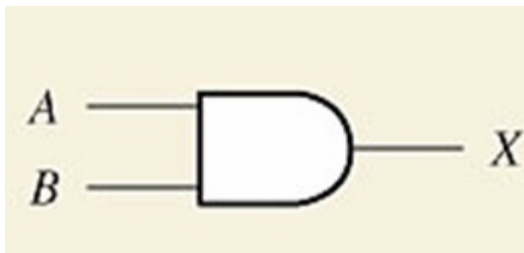
$$X = A + B$$

## 5. NOR Gate



$$X = \overline{A + B}$$

## 2. AND Gate



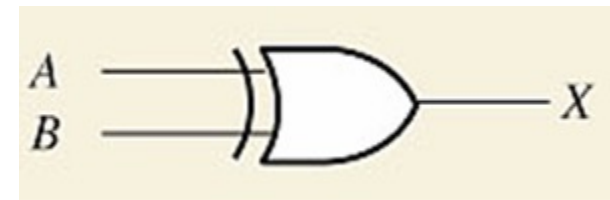
$$X = AB \quad X = A \cdot B$$

## 4. NAND Gate



$$X = \overline{AB}$$

## 6. XOR Gate

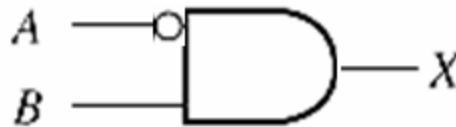


$$X = A \oplus B$$

# Logic Gates

## Pitfalls

- Mix the logic symbols for AND gates and OR gates.
- There is no such logic gate.



- In drawing the waveforms of a circuit, forget to copy the input waveforms, and/or forget to use dashed lines to align inputs and outputs.



# Laws and Rules of Boolean Algebra

L1  $A + B = B + A$      $AB = BA$

L2  $A + (B + C) = (A + B) + C$      $A(BC) = (AB)C$

L3  $A(B + C) = AB + AC$

1.  $A + 0 = A$

2.  $A + 1 = 1$

3.  $A \cdot 0 = 0$

4.  $A \cdot 1 = A$

5.  $A + A = A$

6.  $A + \bar{A} = 1$

7.  $A \cdot A = A$

8.  $A \cdot \bar{A} = 0$

9.  $\bar{\bar{A}} = A$

10.  $A + AB = A$

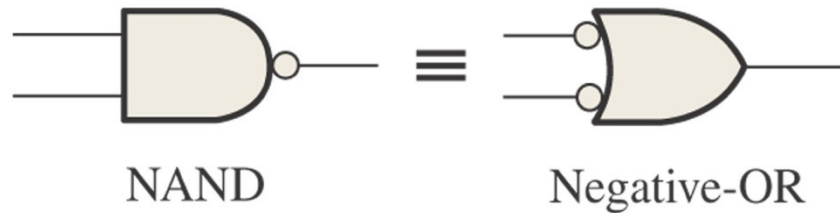
11.  $A + \bar{A}B = A + B$

12.  $(A + B)(A + C) = A + BC$

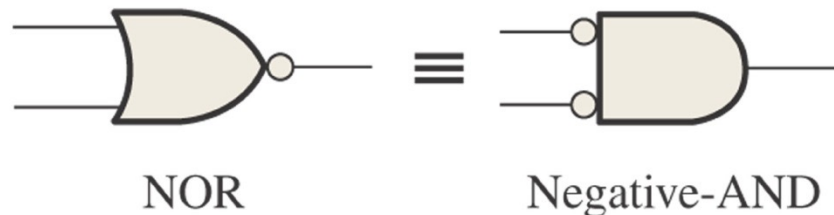
# DeMorgan's Theorems

**Break the bar, change the sign.**

$$\overline{XY} = \bar{X} + \bar{Y}$$



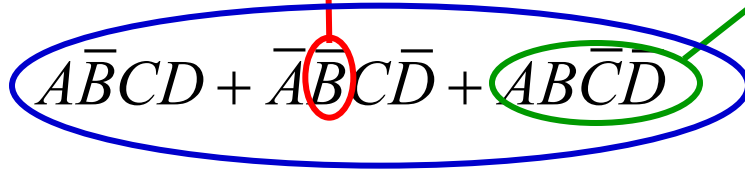
$$\overline{X + Y} = \bar{X} \bar{Y}$$



# Boolean Expressions

- A **literal** is a variable or its complement.

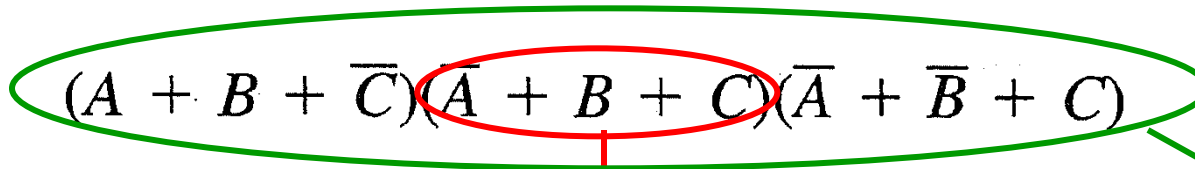
- A **product term** is the product of literals.



The diagram shows the expression  $A\bar{B}CD + \bar{A}\bar{B}C\bar{D} + A\bar{B}\bar{C}D$  enclosed in a blue oval. A red circle highlights the literal  $\bar{B}$  in the second term, with a red arrow pointing to the definition of a literal. A green circle highlights the entire third term  $A\bar{B}\bar{C}D$ , with a green arrow pointing to the definition of a product term. A blue arrow points from the blue oval to the definition of a sum-of-products.

$$A\bar{B}CD + \bar{A}\bar{B}C\bar{D} + A\bar{B}\bar{C}D$$

- A **sum-of-products (SOP)** is the sum of product terms.



The diagram shows the expression  $(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + C)$  enclosed in a green oval. A red circle highlights the sum term  $(\bar{A} + B + C)$ , with a red arrow pointing to the definition of a sum term. A green arrow points from the green oval to the definition of a product-of-sums.

$$(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + C)$$

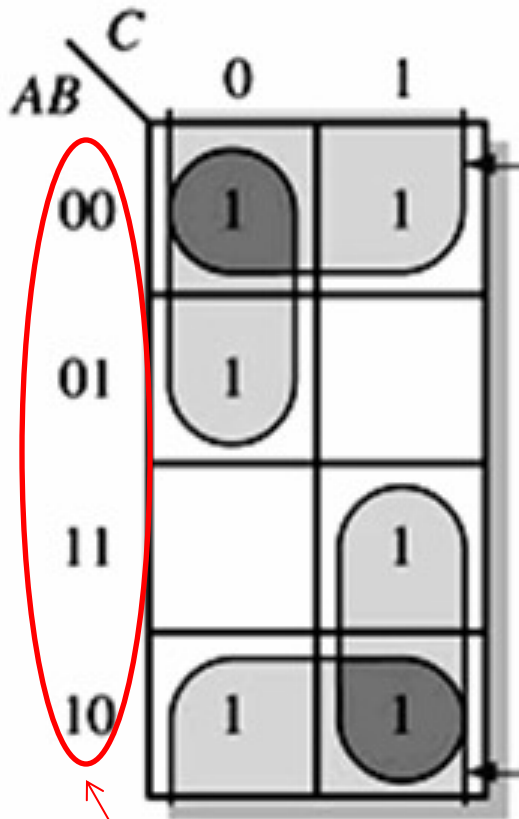
- A **sum term** is the sum of literals.

- A **product-of-sums (POS)** is the product of sum terms.

# Karnaugh Map SOP Minimization

## Grouping the 1s

- The goal is to **maximize the size** of the groups and to **minimize the number** of groups.
- A group may contain 1, 2, 4, 8, or 16 adjacent cells.
- Each 1 must be included in one or **more** groups.

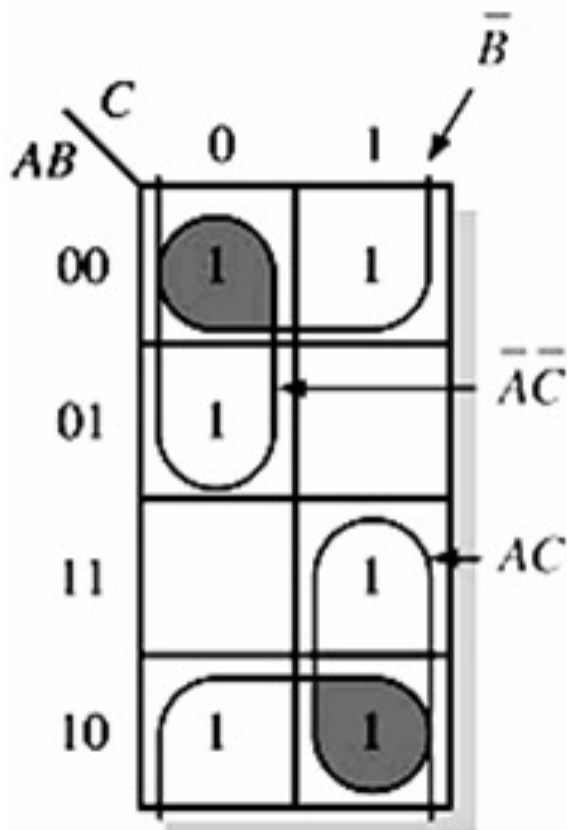


**In adjacent order**

# Karnaugh Map SOP Minimization

## Determine the Minimum SOP

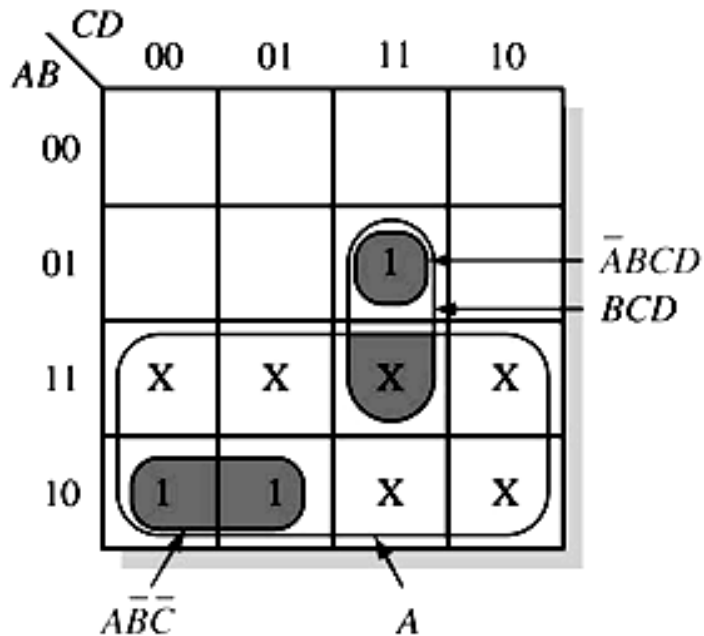
1. When a variable appears in both complemented and uncomplemented form in a group, that variable is eliminated.
2. Variables that are the same for all cells of the group must appear — 1 for uncomplemented form and 0 for complemented form.



# Karnaugh Map SOP Minimization

## Don't cares

1. If a don't care X can contribute to the extension of a 1's group, then it is thought as 1.
2. If it is not helpful to the extension of a 1's group, then it is thought as 0.



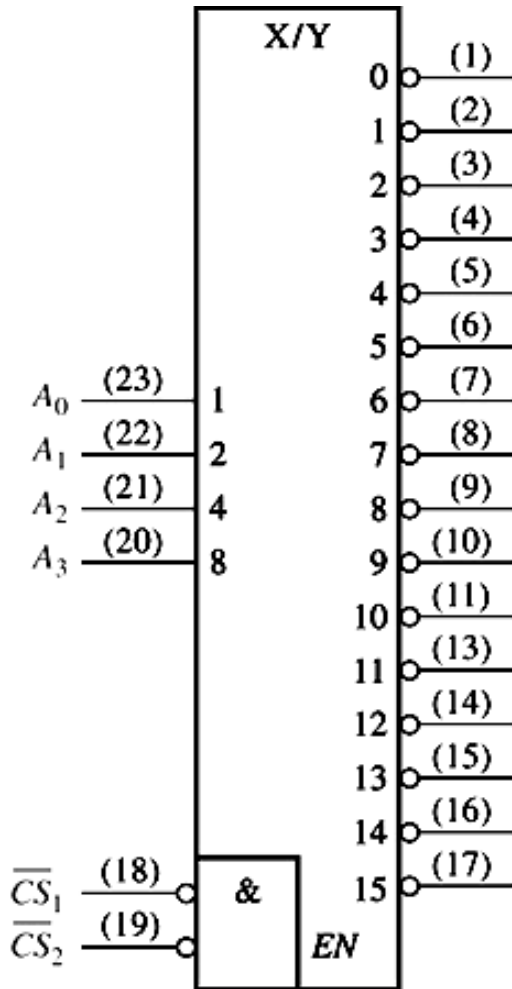
(b) Without "don't cares"  $Y = A\bar{B}\bar{C} + \bar{A}BCD$   
 With "don't cares"  $Y = A + BCD$

# Karnaugh Maps

## Pitfalls

- Variable values should be sorted in 00, 01, **11**, 10.
- Variable identifiers are sorted according to alphabetical order or from MSB to LSB, e.g. AB/C, WX/YZ,  $Q_2Q_1/Q_0$
- 1's groups are not fully maximized - forget wrap-around adjacency.
- Don't know how to use don't cares. X's are ignored in the process of grouping 1's.
- Don't know much about K-maps for POS.

# Decoders

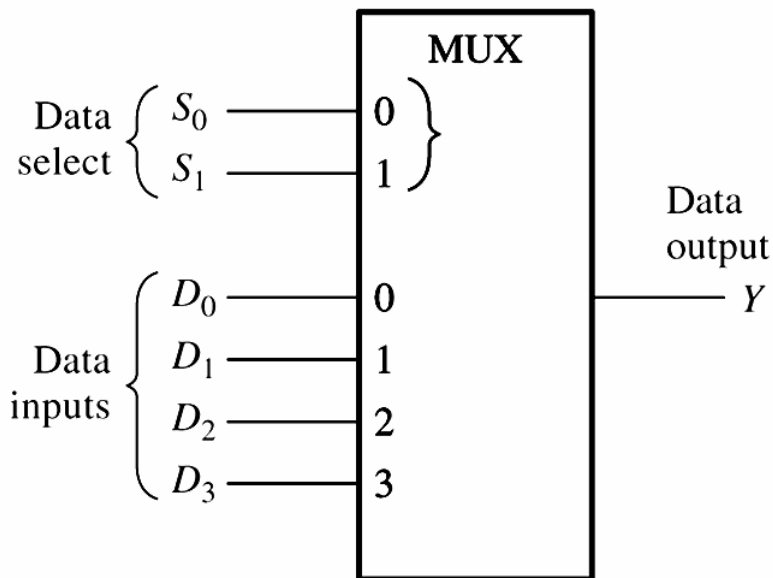


- A decoder is used to detect a specified combination of input bits (code), e.g. 7 is active-LOW for inputs 0111.
- When enabled, only one output is active for 4-line-to-16-line decoder; one or more output will be active for a BCD-to-7-segment decoder.
- When disabled, no output will be active.



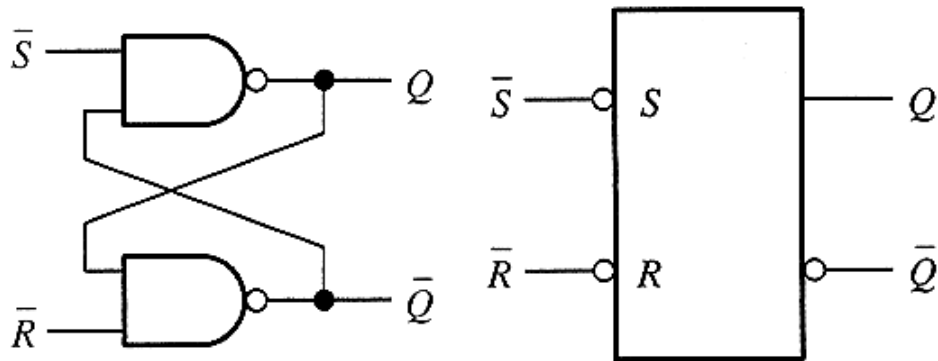
# Multiplexers

- A *multiplexer (MUX)*, also known as a *data selector*, outputs one of its multiple data inputs.
- The *data select* inputs will decide which data input is to be switched to the output line.

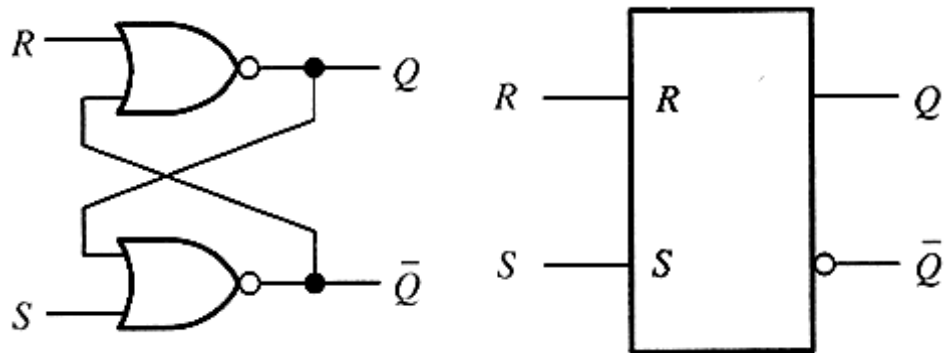


DATA-SELECT INPUTS		INPUT SELECTED
$S_1$	$S_0$	
0	0	$D_0$
0	1	$D_1$
1	0	$D_2$
1	1	$D_3$

# Latches – The S-R Latch

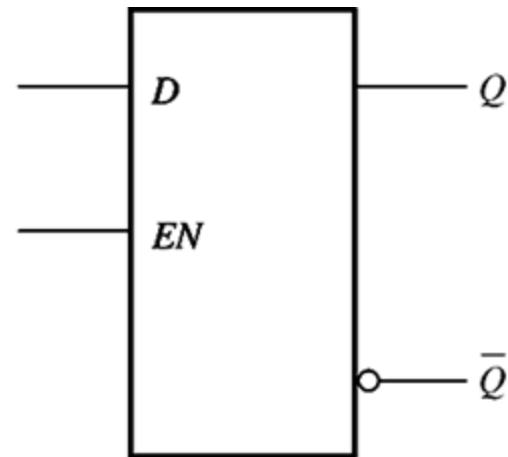
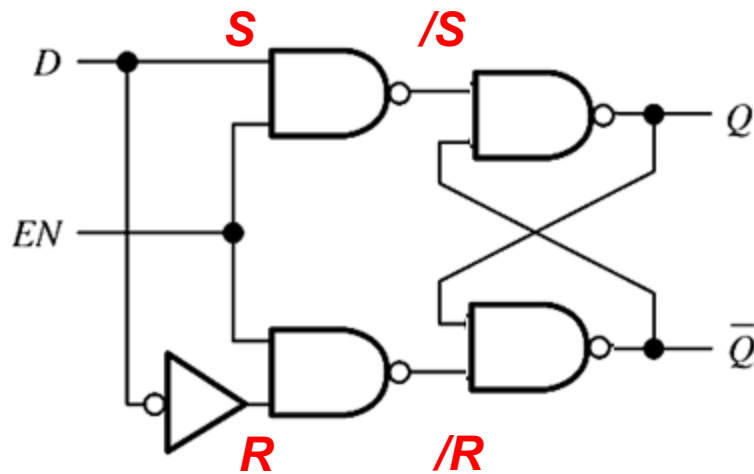
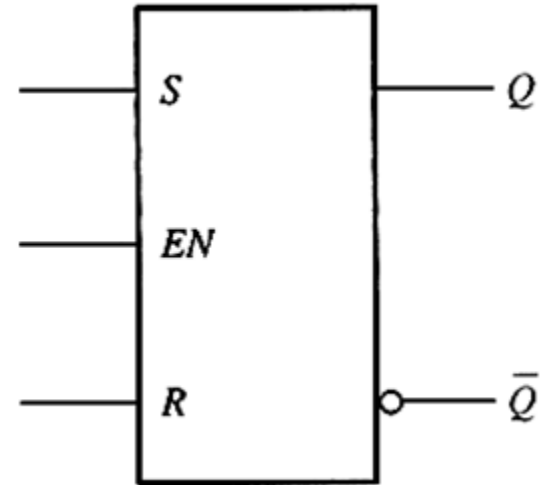
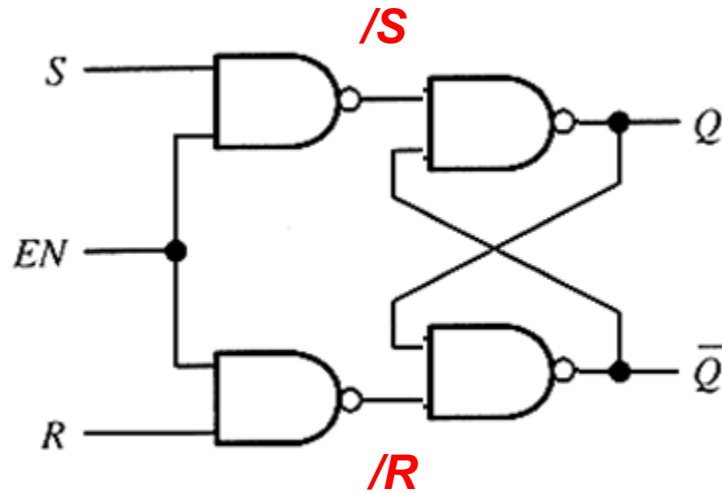


INPUTS		OUTPUTS	
$\bar{S}$	$\bar{R}$	$Q$	$\bar{Q}$
1	1	$Q_0$	$\bar{Q}_0$
0	1	1	0
1	0	0	1
0	0	?	?

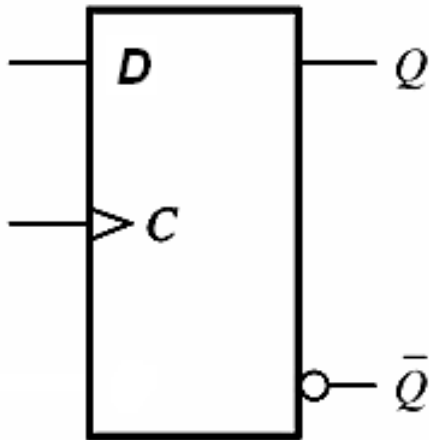


INPUTS		OUTPUTS	
$S$	$R$	$Q$	$\bar{Q}$
0	0	$Q_0$	$\bar{Q}_0$
0	1	0	1
1	0	1	0
1	1	?	?

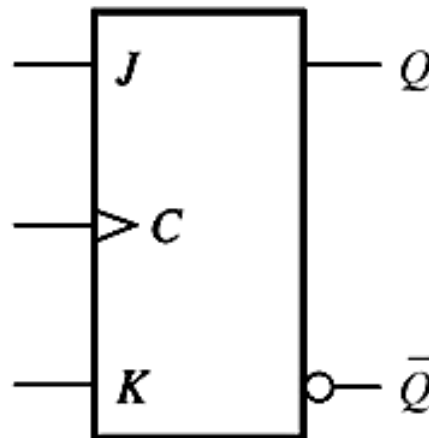
# Latches – The S-R Latch



# Edge-Triggered Flip-Flops



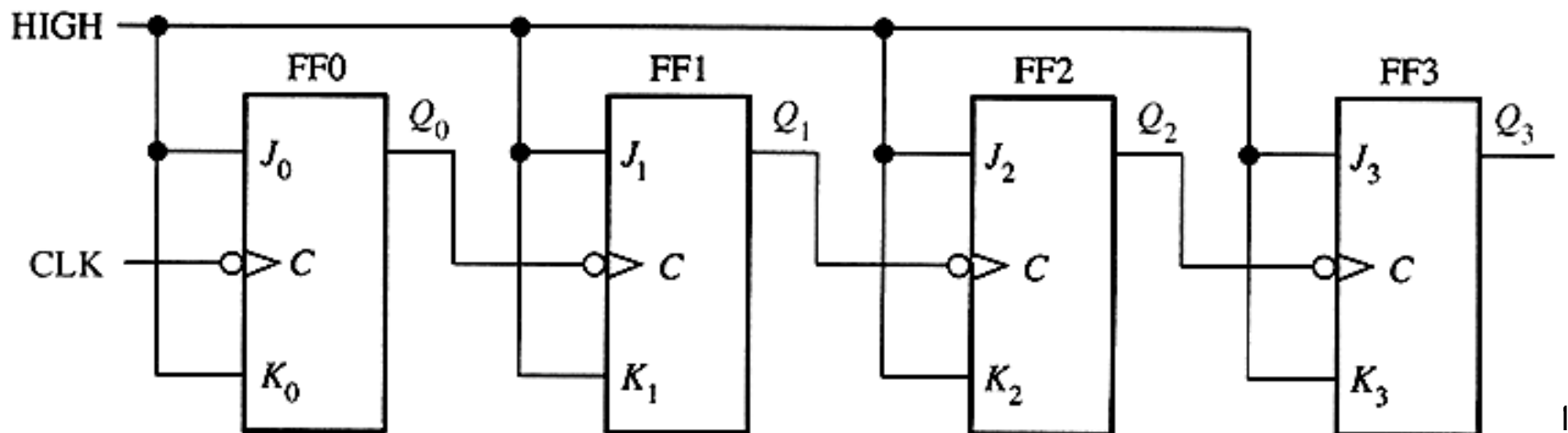
INPUTS		OUTPUTS		COMMENTS
$D$	CLK	$Q$	$\bar{Q}$	
1	$\uparrow$	1	0	SET (stores a 1)
0	$\uparrow$	0	1	RESET (stores a 0)



INPUTS			OUTPUTS		COMMENTS
$J$	$K$	CLK	$Q$	$\bar{Q}$	
0	0	$\uparrow$	$Q_0$	$\bar{Q}_0$	No change
0	1	$\uparrow$	0	1	RESET
1	0	$\uparrow$	1	0	SET
1	1	$\uparrow$	$\bar{Q}_0$	$Q_0$	Toggle

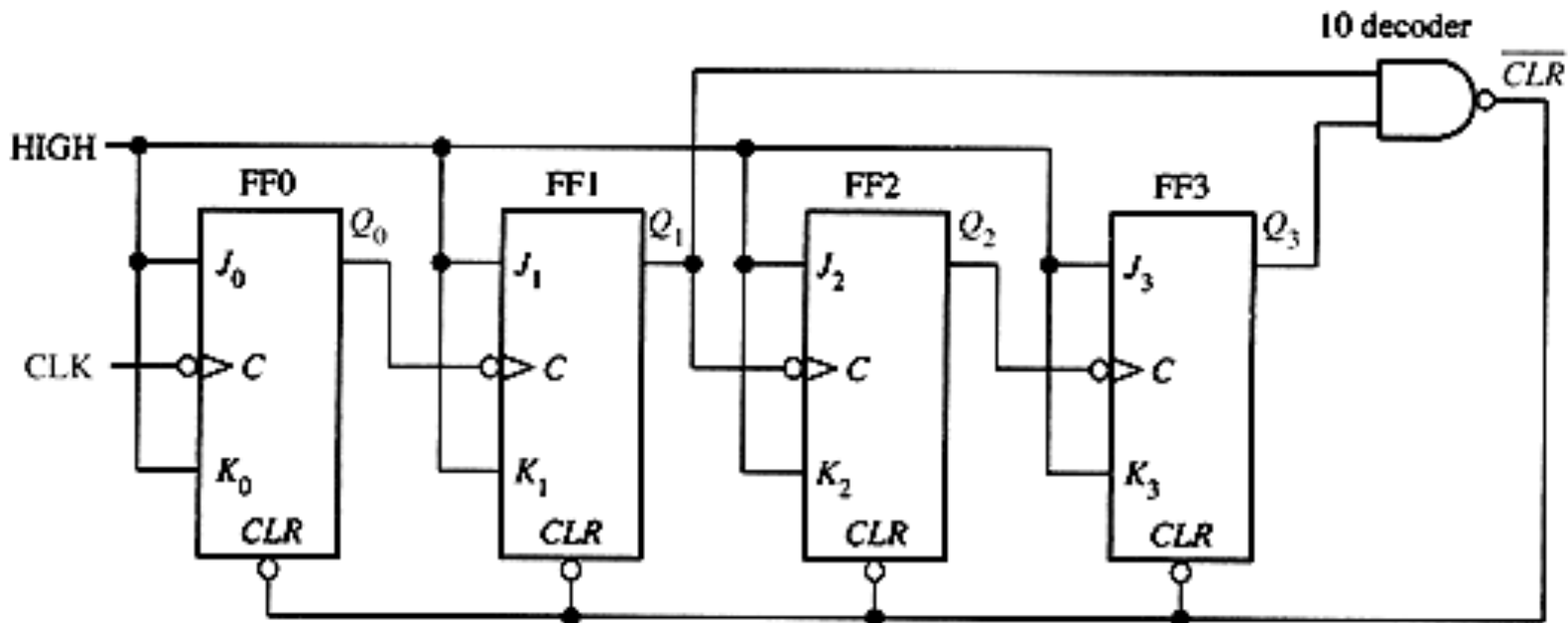
# Asynchronous Binary Counters

- J and K are HIGH for all flip-flops.
- Each flip-flop, except the first, is clocked by the **output** of the preceding one if it is **negative edge-triggered**.
- If it is **positive-edge triggered**, it is clocked by **complemented output** of the preceding one.



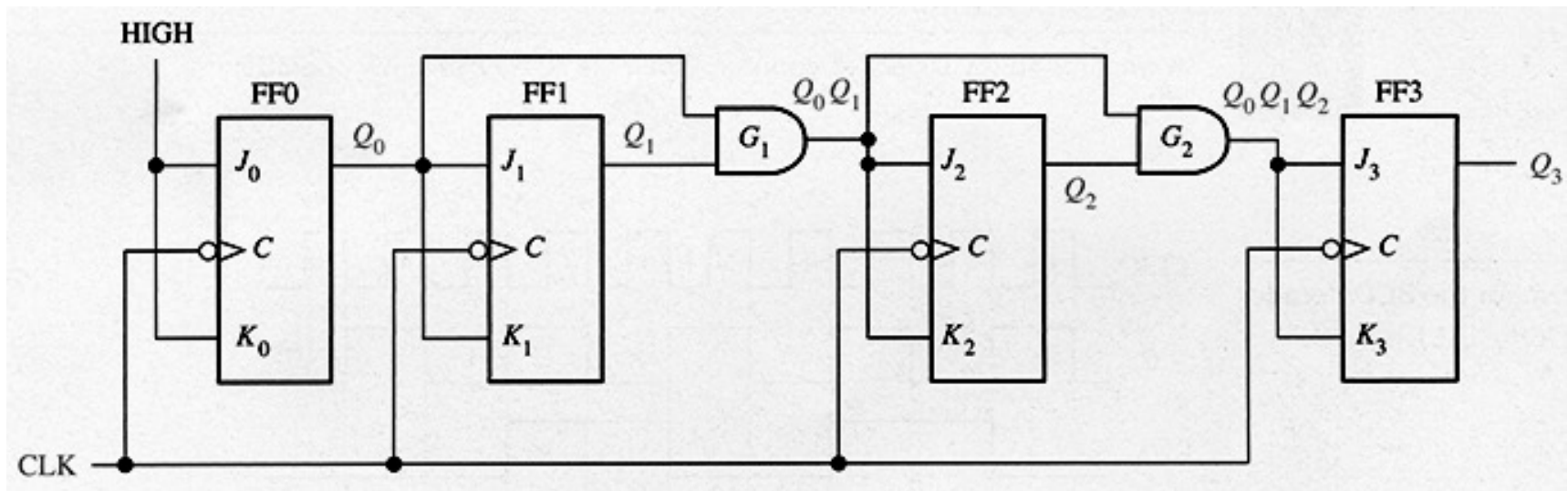
# Asynchronous Binary Counters

- A truncated sequence can be produced by clearing and/or presetting the flip-flops at proper states.
- For a decade counter, once state 1010 (**this is 10, not 9**) appears, the decoder will reset the counter.



# Synchronous Binary Counters

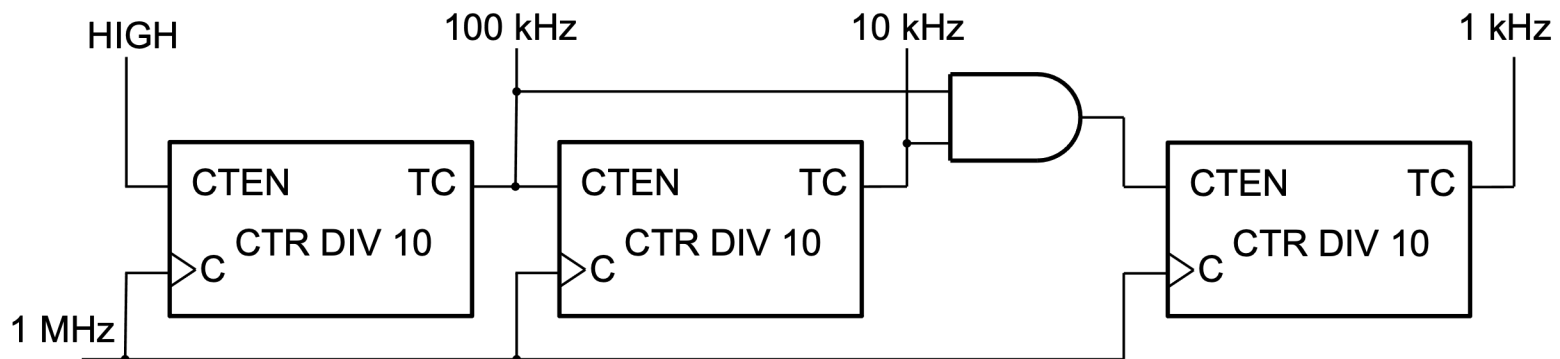
- All the flip-flops are clocked by the same CLK.
- J and K of the flip-flops other than the first one are the decoding result of the outputs.



# Synchronous Binary Counters

For IC synchronous counters

- Count enable (CTEN) of the 1<sup>st</sup> counter is HIGH.
- CTEN of each of other counters is connected to Terminal count (TC) output of the preceding counter.

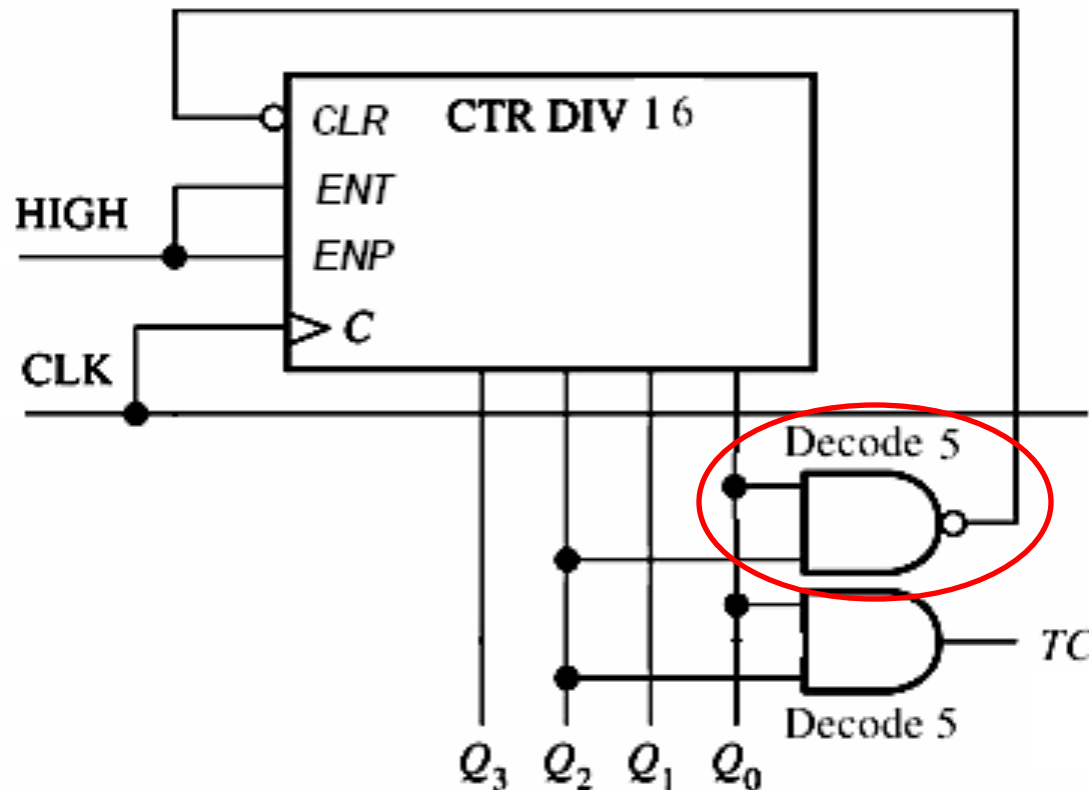




# Synchronous Binary Counters

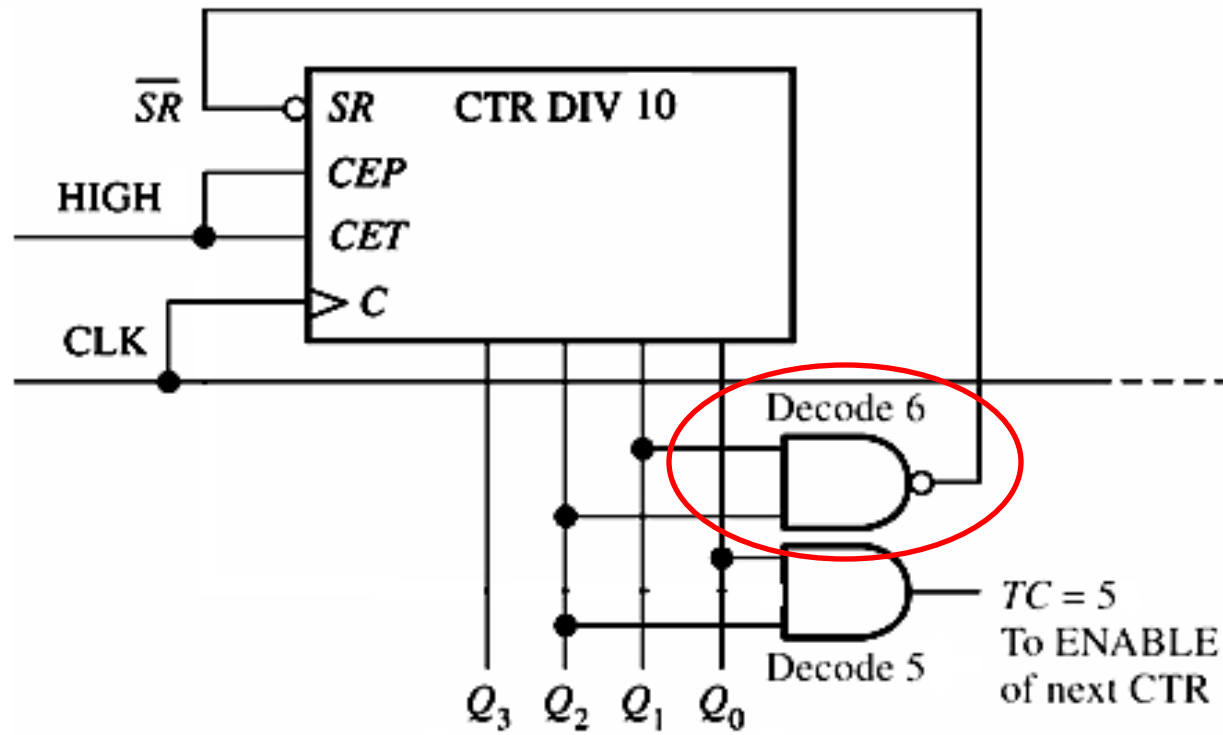
Truncated sequences can be realized by *decoding the terminal count and then clearing the counters*, e.g. a divide-by-6 counter.

If CLR is **synchronous**, decode **terminal count**.



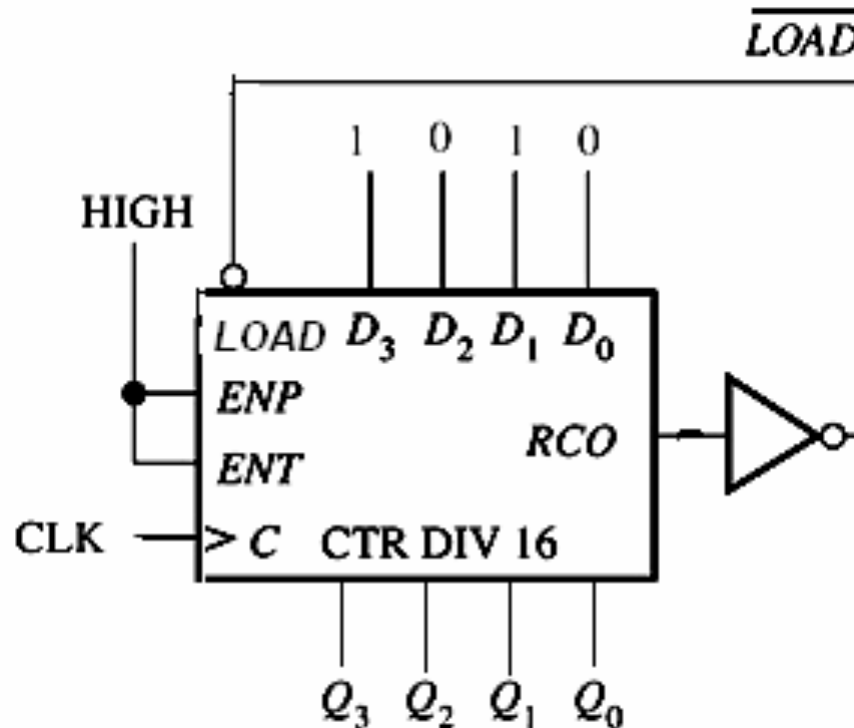
# Synchronous Binary Counters

If CLR is **asynchronous**, then decode **terminal count + 1**.



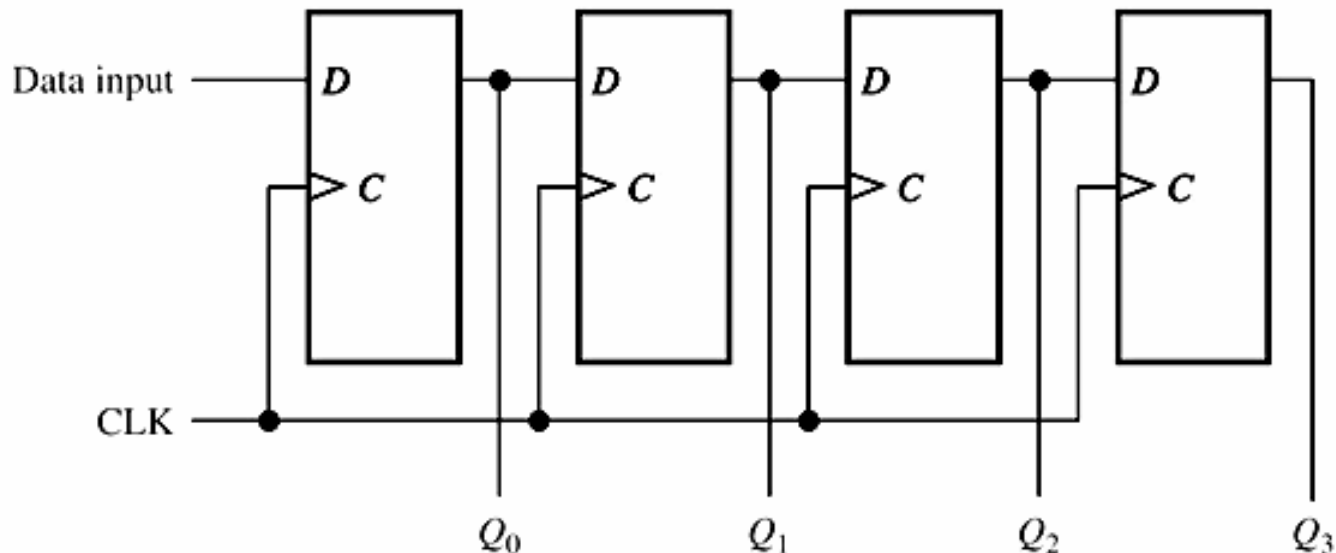
# Synchronous Binary Counters

Truncated sequences can also be realized by *loading an initial count at the terminal count*. e.g. a divide-by-6 counter.



# Shift Registers

- A shift register is made up of a set of cascaded flip-flops which store and move data.
- Serial In/Serial Out and Serial In/Parallel Out



# Shift Registers

- Parallel In/Serial Out
- A Parallel In/Parallel Out shift register is a set of separate flip-flops sharing the same clock.

