Schedule…

Date	Day	Class No.	Title	Chapters	HW Due date	Lab Due date	Exam
1 Dec	Mon	25	Final Review			LAB 8	
2 Dec	Tue						
3 Dec	Wed	26	Final Review				
4 Dec	Thu						
5 Dec	Fri		Recitation		HW 11		
6 Dec	Sat						
7 Dec	Sun						
8 Dec	Mon	27	Final Review			LAB 9	
9 Dec	Tue						

Summary

Jacob 6:12

12 O be wise; what can I say more?

Lecture 25 – Final Review

Final Exam

- 17 – 21 December
- Chapters 2 – 6, 8, 13 – 15
- 30 questions
  - 30 multiple choice (answer on bubble sheet!)
    - 1 point each
  - 0 long answer (show your work!)
    - 4 or 5 points each
- Closed book!
  - Three 3x5 cards allowed
- Calculators allowed
- No time limit
- Study lecture slides and homework

Final Exam Review…Overview

1. Exam 1 Review
2. Exam 2 Review
3. Binary Numbers
   - Signed & Unsigned
   - Conversions
4. Logic Functions
   - Conversions among 3 representations
5. Boolean Algebra
6. Combinational Logic
7. Sequential Logic
8. Digital to Analog Converters (DACs)

Binary Numbers – Unsigned

Binary word (B): a sequence of n **1**s and **0**s

$$B = b_{n-1}b_{n-2}\ldots b_2 b_1 b_0 . b_{-1}b_{-2}\ldots b_{-(m-1)}b_{-m}$$

Binary point

- EX:
  - B = 10100101.1001 (n = 8, m=4)

Converting from binary (B) to decimal (D)

$$B = b_{n-1}b_{n-2}\ldots b_2 b_1 b_0 . b_{-1}b_{-2}\ldots b_{-(m-1)}b_{-m}$$

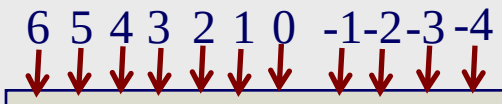
$$D = b_{n-1}\cdot 2^{n-1} + b_{n-2}\cdot 2^{n-2} + \ldots + b_1\cdot 2^1 + b_0\cdot 2^0 + b_{-1}\cdot 2^{-1} + b_{-2}\cdot 2^{-2} + \ldots + b_{-(m-1)}\cdot 2^{-(m-1)} + b_{-m}\cdot 2^{-m}$$

Binary Numbers – Unsigned

Example1: What is $\mathbf{0110101.0101}_2$ in decimal?

Binary Numbers – Unsigned

Example1: What is **0110101.0101_2** in decimal?



$$0110101.0101_2 = 0\cdot 2^6 + 1\cdot 2^5 + 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4}$$

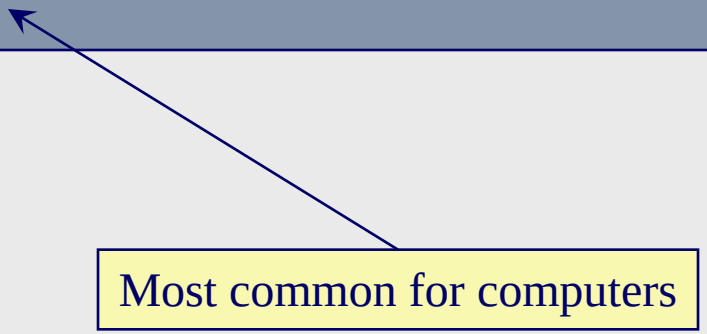
$$= 32 + 16 + 4 + 1 + 0.25 + 0.0625$$

$$= 53.3125_{10}$$

Binary Numbers – Signed

3 common representations for **signed** integers:

1. Sign magnitude
2. 1's compliment
3. 2's compliment



Binary Numbers – Sign-Magnitude

Range: $-(2^{n-1}-1) \longrightarrow (2^{n-1}-1)$

Representations

- 01111_{binary} => $15_{decimal}$
- 11111 => -15
- 00000 => 0
- 10000 => -0

The **MSB** encodes the sign:

0 = +

1 = –

Problem

- Difficult addition/subtraction
  - check signs
  - convert to positive
  - use adder or subtractor as required
- How to add two sign-magnitude numbers?
  - Ex: 1 + (-4)

Binary Numbers – 1's Complement

Range: $-\left(2^{n-1}-1\right) \longrightarrow \left(2^{n-1}-1\right)$

Representations

- 00110_{binary} => 6_{decimal}
- 11001 => -6
- 00000 => 0
- 11111 => -0

To negate a number, Invert it, bit-by-bit.

MSB still encodes the sign:

0 = +

1 = –

Problem

- Difficult addition/subtraction
  - no need to check signs as before
  - cumbersome logic circuits
    - *end-around-carry*
- How to add to one's complement numbers?
  - Ex: 4 + (-3)

Binary Numbers – Two's Complement

Range: $-(2^{n-1}) \longrightarrow (2^{n-1}-1)$

Representation:

- If number is **positive** or **zero**,
  - normal binary representation, zeroes in upper bit(s)
- If number is **negative**,
  - start with positive number
  - flip every bit (i.e., take the one's complement)
  - then add one

MSB still encodes the sign:

`0` = +

`1` = –

```
 00101  (5)          01001  (9)
 11010  (1's comp)   10110  (1's comp)
+    1               +    1
-----               -----
 11011  (-5)         10111  (-9)
```

Binary Numbers – Signed

Example2: What is the 2's complement of $\mathbf{0110101}_2$?

Binary Numbers – Signed

Example2: What is the 2's complement of $\mathbf{0110101}_2$?

```
  0110101     (53)

  1001010     (1's comp)
+       1     (add 1)
-------
  1001011     (-53)
```

Binary Numbers – Decimal to Binary

Positive numbers

1. start with empty result
2. if decimal number is odd, prepend '`1`' to result
   else prepend '`0`'
3. divide number by 2, throw away fractional part (INTEGER divide)
4. if number is non-zero, go back to 2 else you are done

Negative numbers

- do above for **positive version** of number and **negate result.**

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

	Is it odd?
195	1

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

	Is it odd?
195	1
97	1

Integer division by 2:
195/2 = 97.5 → 97

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

Integer division by 2:
97/2 = 48.5 → 48

	Is it odd?
195	1
97	1
48	0

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

Integer division by 2:
48/2 = 24

	Is it odd?
195	1
97	1
48	0
24	0

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

Integer division by 2:
24/2 = 12

	Is it odd?
195	1
97	1
48	0
24	0
12	0

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

Integer division by 2:
12/2 = 6

	Is it odd?
195	1
97	1
48	0
24	0
12	0
6	0

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

Integer division by 2:
6/2 = 3

	Is it odd?
195	1
97	1
48	0
24	0
12	0
6	0
3	1

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

Integer division by 2:
3/2 = 1.5 → 1

	Is it odd?
195	1
97	1
48	0
24	0
12	0
6	0
3	1
1	1

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

Integer division by 2:
1/2 = 0.5 → 0

	Is it odd?
195	1
97	1
48	0
24	0
12	0
6	0
3	1
1	1
0	0

Binary Numbers – Decimal to Binary

Example3: convert -195 to 2's complement binary notation

1. First find positive version (195) in binary – then take 2's complement

	Is it odd?
195	1
97	1
48	0
24	0
12	0
6	0
3	1
1	1
0	0

011000011_2

2's complement

100111101_2

Hexadecimal Notation

- Binary is hard to read and write by hand
- Hexadecimal is a common alternative
  - 16 digits are **0123456789ABCDEF**

```
0100 0111 1000 1111 = 0x478F
1101 1110 1010 1101 = 0xDEAD
1011 1110 1110 1111 = 0xBEEF
1010 0101 1010 0101 = 0xA5A5
```

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

1. Separate binary code into groups of 4 bits (starting from the right)
2. Translate each group into a single hex digit

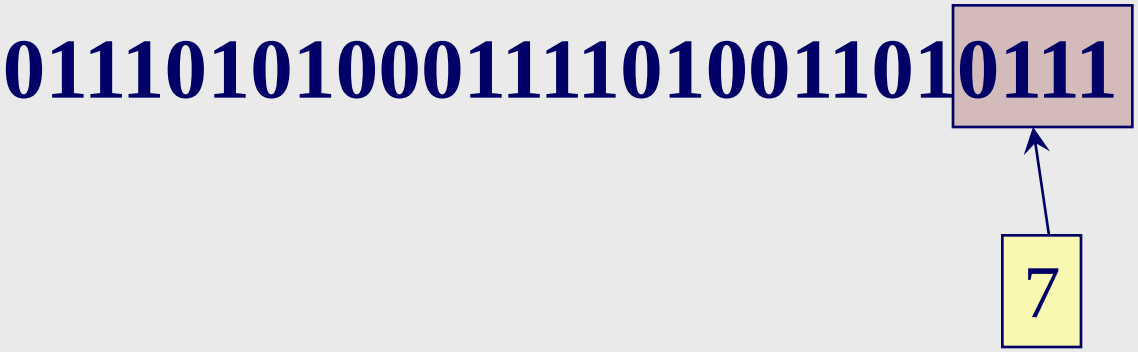
0x is a common prefix for writing numbers which means *hexadecimal*

Binary Numbers – Hexadecimal

Example4: convert $\mathbf{01110101000111101001101011 1_2}$ to hexadecimal notation

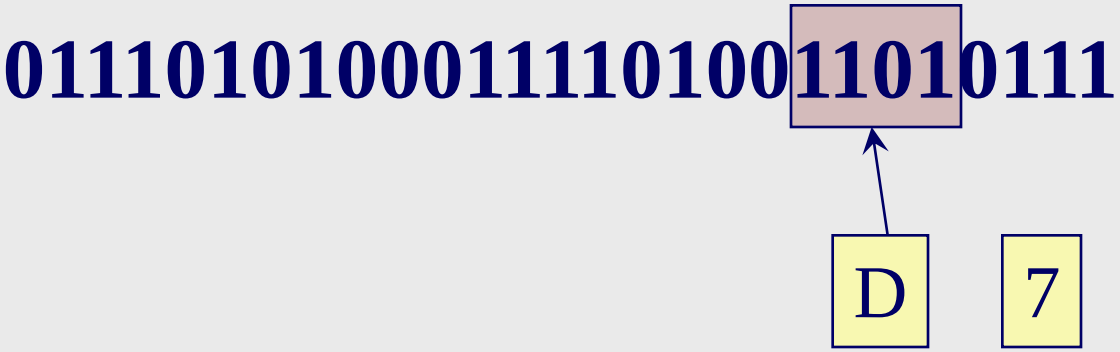
Binary Numbers – Hexadecimal

Example4: convert $\mathbf{01110101000111101001101011 1_2}$ to hexadecimal notation



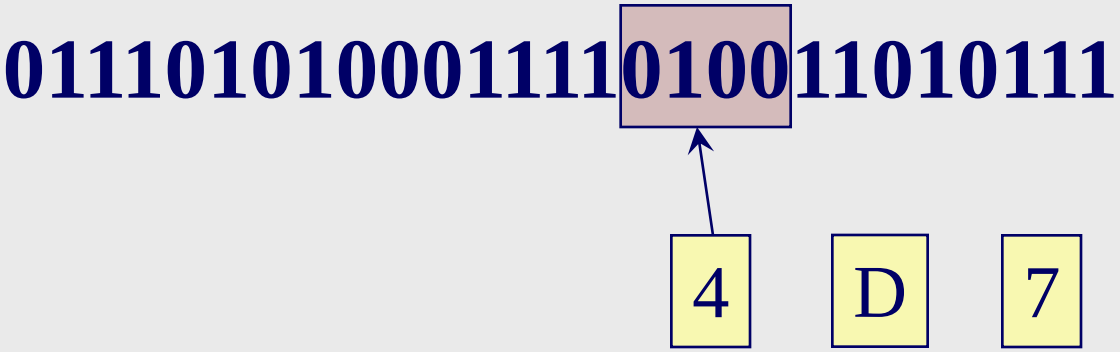
Binary Numbers – Hexadecimal

Example4: convert $\mathbf{01110101000111101001101011 1_2}$ to hexadecimal notation



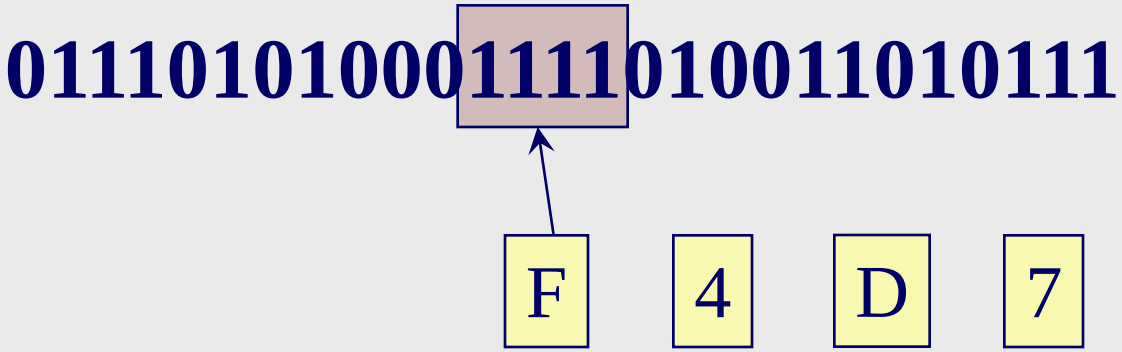
Binary Numbers – Hexadecimal

Example4: convert **$01110101000111101001101011 1_2$** to hexadecimal notation



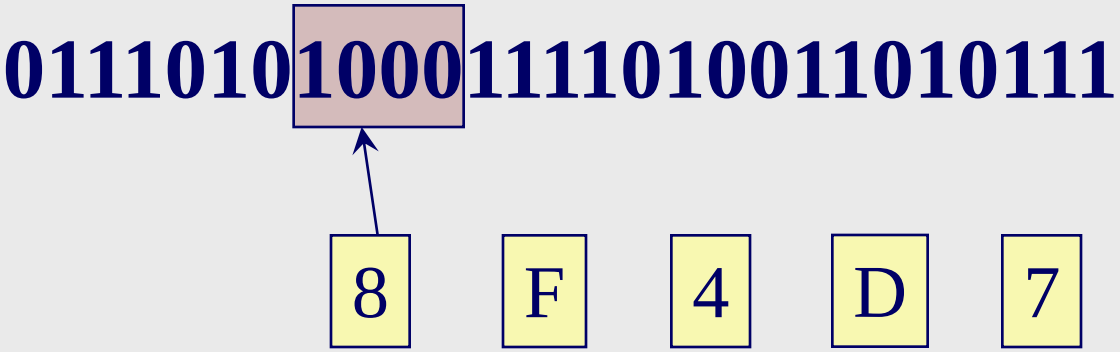
Binary Numbers – Hexadecimal

Example4: convert $\mathbf{01110101000111101001101011 1_2}$ to hexadecimal notation



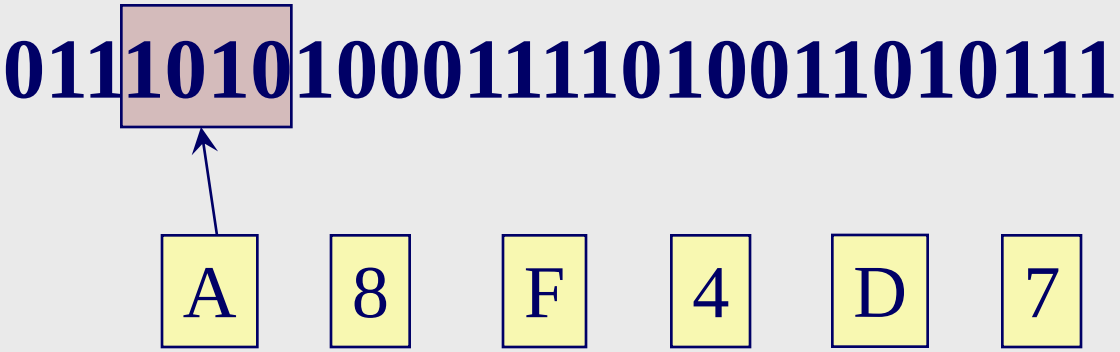
Binary Numbers – Hexadecimal

Example4: convert **$01110101000111101001101011 1_2$** to hexadecimal notation



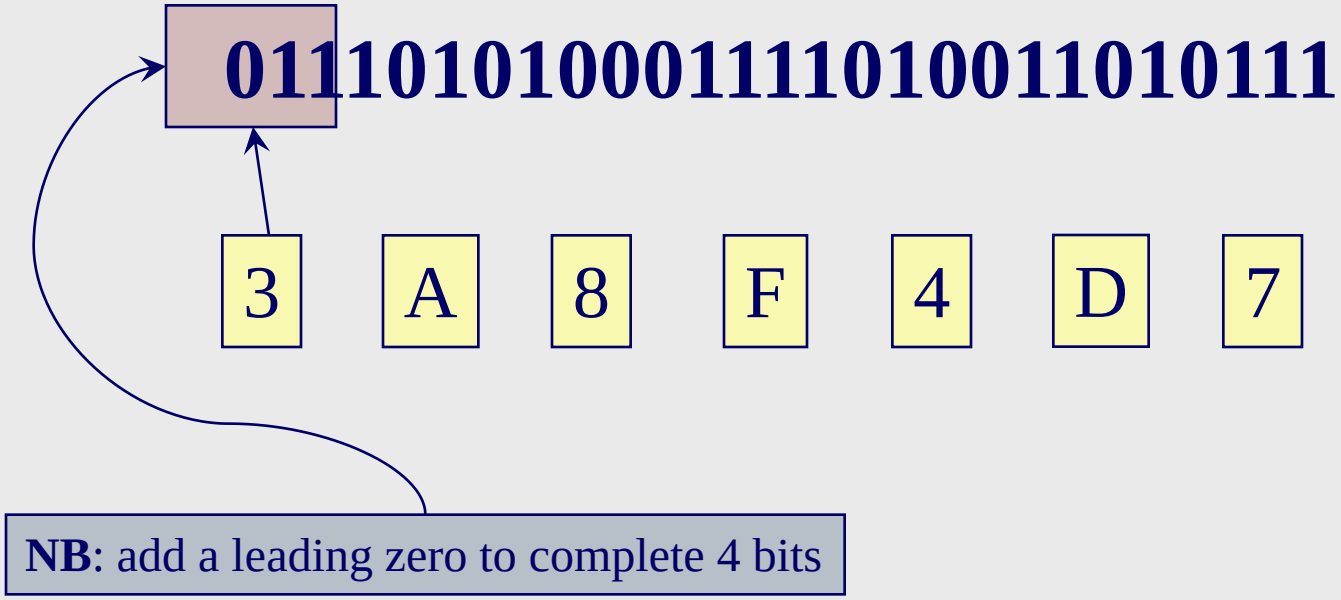
Binary Numbers – Hexadecimal

Example4: convert $\mathbf{01110101000111101001101011 1_2}$ to hexadecimal notation



Binary Numbers – Hexadecimal

Example4: convert **$01110101000111101001101011 1_2$** to hexadecimal notation



Binary Numbers – Hexadecimal

Example4: convert $\mathbf{01110101000111101001101011 1_2}$ to hexadecimal notation

01110101000111101001101011 1

0x3A8F4D7

Binary Numbers

Example5: Complete the following table

	Unsigned	Sign-magitude	1's complement	2's complement	HEX
01100110					
11101011					
93					
-93					

Binary Numbers

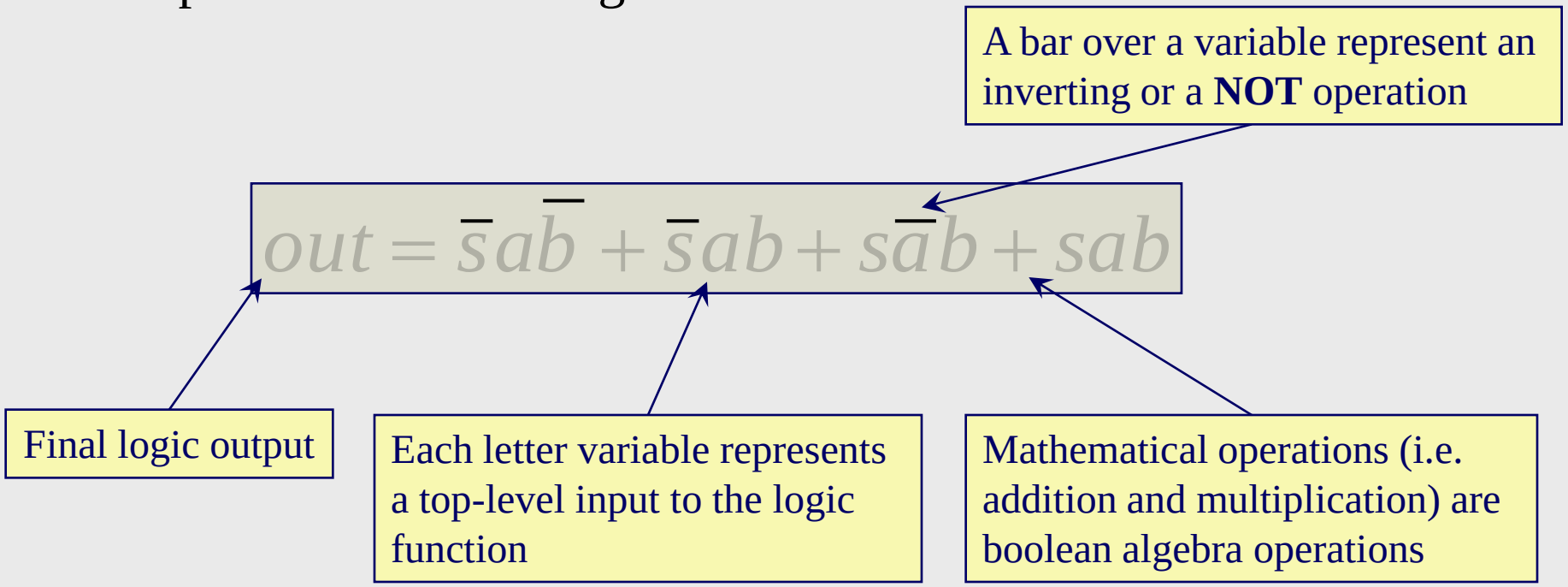
Example5: Complete the following table

	Unsigned	Sign-magitude	1's complement	2's complement	HEX
01100110	102	102	102	102	0x66
11101011	235	-107	-20	-21	0xEB
93	01011101	01011101	01011101	01011101	0x5D
-93	NA	11011101	10100010	10100011	0xA3

Logic Functions

3 different ways to represent logic functions:

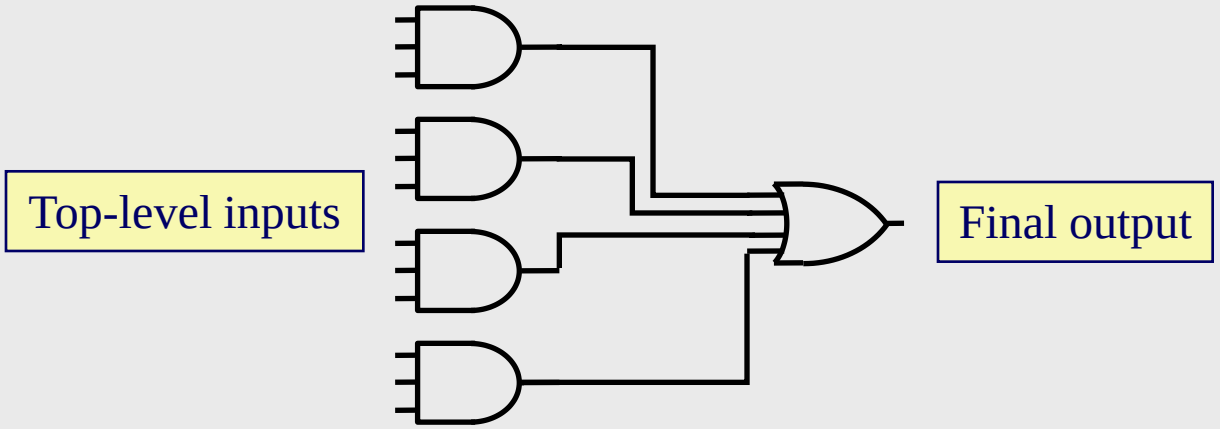
1. **Equation**: a mathematical representation of a logic function



Logic Functions

3 different ways to represent logic functions:

2. **Gates**: a visual block representation of the function



Four 3-input **AND** gates feeding into one 4-input **OR** gate

Logic Functions

3 different ways to represent logic functions:

3. **Truth Table**: indicates what the output will be for every possible input combination

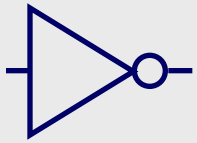
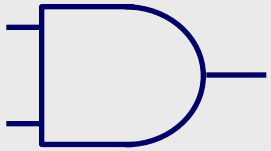
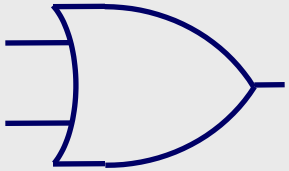
A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

If there are n inputs (left-hand columns) there will be 2^n entries (rows) in the table
EX: 3 inputs require $2^3 = 8$ rows

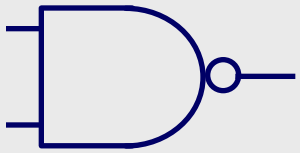
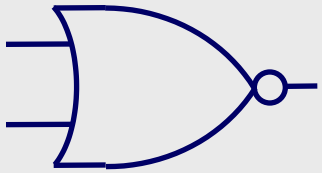
There will always be at least one output (right-hand columns)

For each input combination (row) output(s) will be either **0** or **1**

Logic Functions – Gates

Type	Symbol	Equation	Truth Table
NOT		$OUT = \overline{IN}$	IN | OUT 0 | 1 1 | 0
AND		$OUT = A \cdot B$	A B | OUT 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$OUT = A + B$	A B | OUT 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1

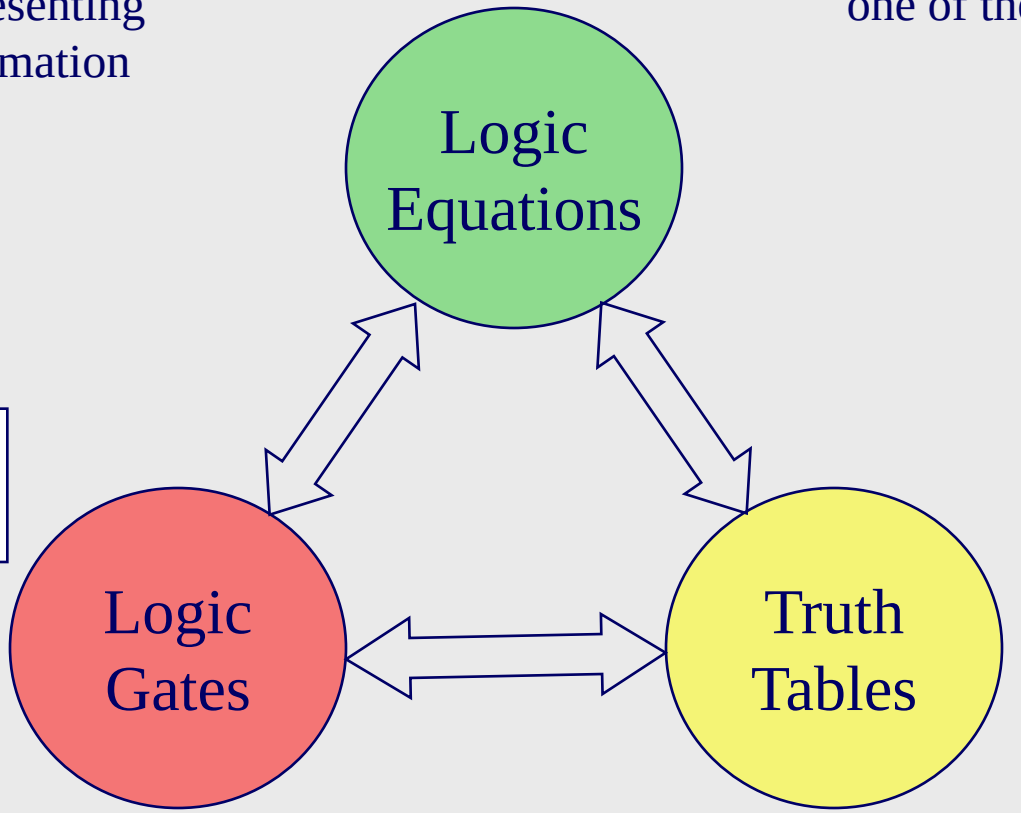
Logic Functions – Gates

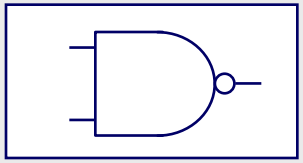
Type	Symbol	Equation	Truth Table
NAND		$OUT = \overline{A \cdot B}$	A B OUT 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$OUT = \overline{A + B}$	A B OUT 0 0 1 0 1 0 1 0 0 1 1 0

Logic Functions – Translation

These are three different ways of representing logical information

$\overline{A \cdot B}$





You can convert any one of them to any other

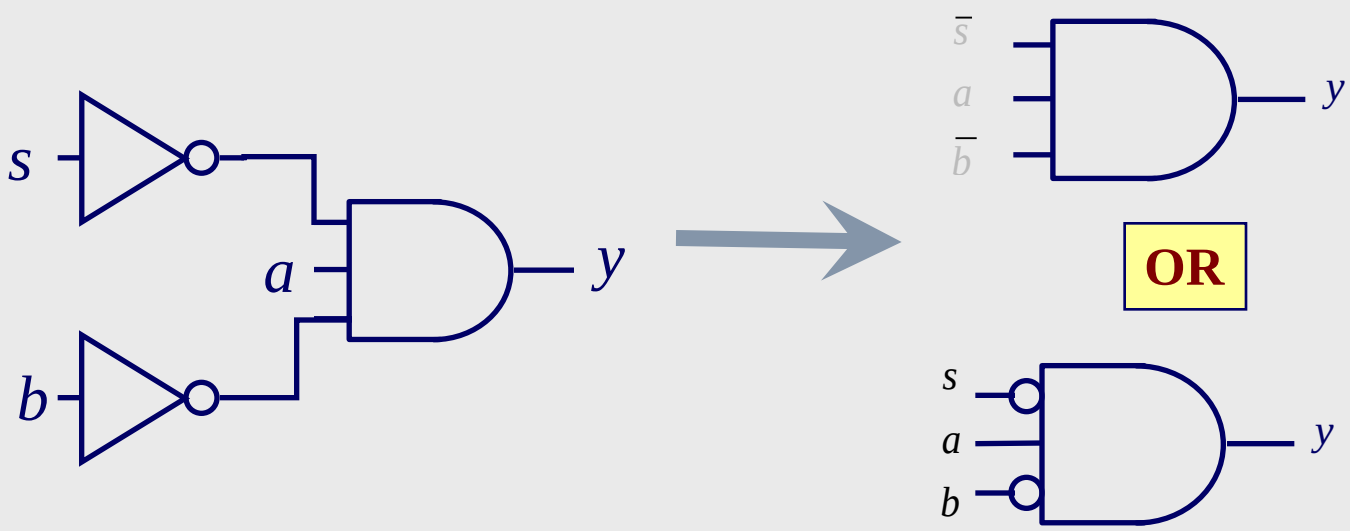
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

Equations to Gates

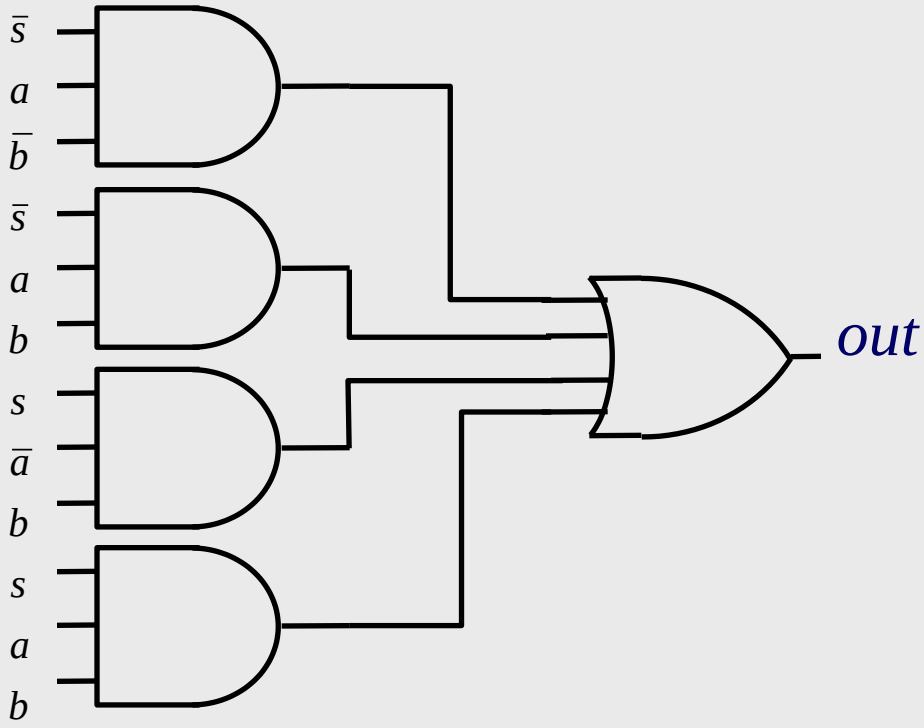
$$y = \text{NOT}(s) \ \text{AND} \ a \ \text{AND} \ \text{NOT}(b)$$

OR

$$y = \bar{s} \cdot a \cdot \bar{b}$$



Logic Functions – Gates to Equations



$$out = (\bar{s} \cdot a \cdot \bar{b}) + (\bar{s} \cdot a \cdot b) + (s \cdot \bar{a} \cdot b) + (s \cdot a \cdot b)$$

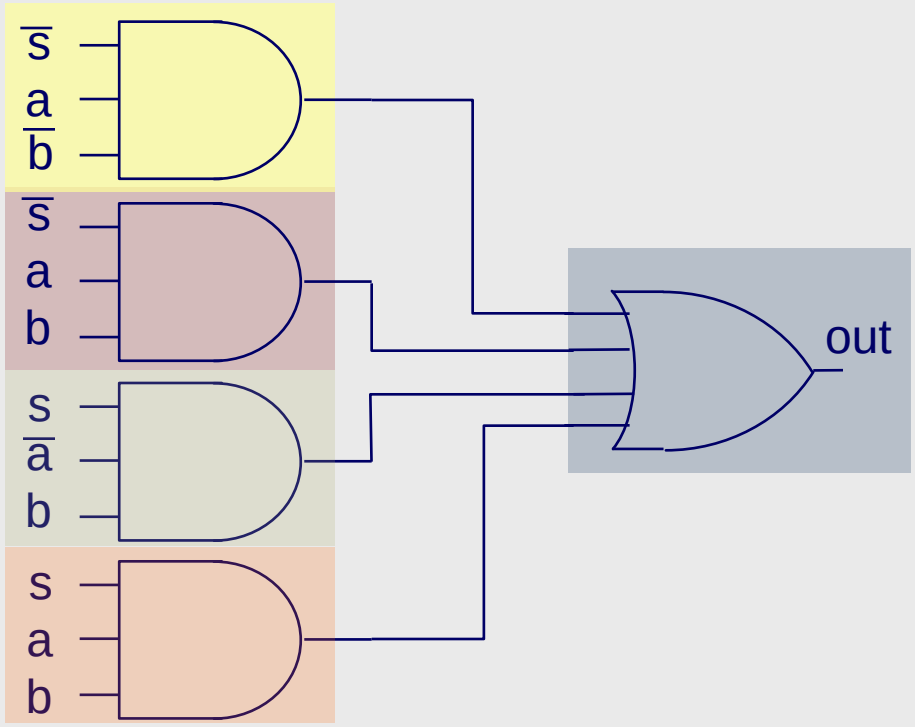
OR

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

Logic Functions – Truth Tables to Gates

- Each row of truth table is an **AND** gate
- Each output column is an **OR** gate

S	A	B	OUT
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



Logic Functions – Truth Table to Equations

- Write out truth table a combination of **AND**'s and **OR**'s
  - equivalent to gates
  - easily converted to gates

S	A	B	OUT
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

Logic Functions – Equations to Truth Tables

- For each **AND** term
  - fill in the proper row on the truth table

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

S	A	B	OUT
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Boolean Algebra – Rules

1. $0 + X = X$
2. $1 + X = 1$
3. $X + X = X$
4. $X + \overline{X} = 1$
5. $0 \cdot X = 0$
6. $1 \cdot X = X$
7. $X \cdot X = X$
8. $X \cdot \overline{X} = 0$
9. $\overline{\overline{X}} = X$

10. $X + Y = Y + X$
11. $X \cdot Y = Y \cdot X$
12. $X + (Y + Z) = (X + Y) + Z$
13. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
14. $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$

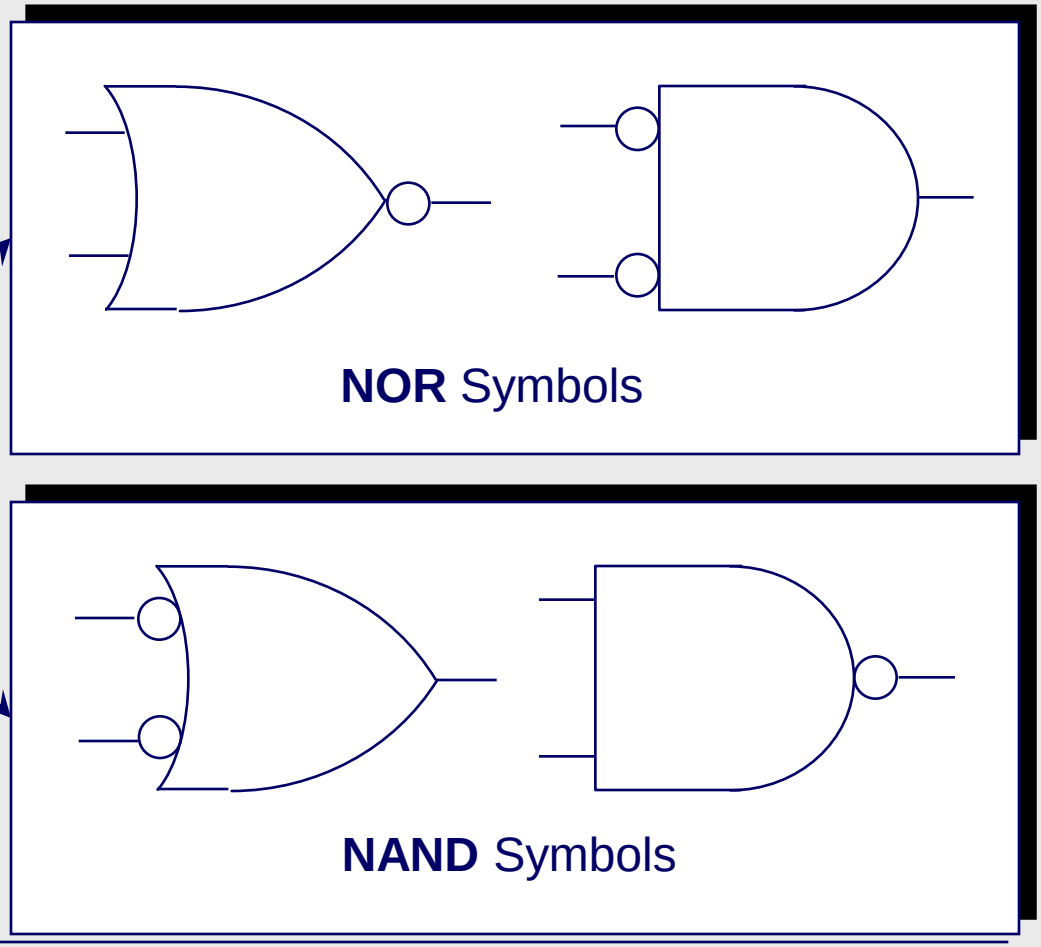
15. $X + X \cdot Z = X$
16. $X \cdot (X + Y) = X$
17. $(X + Y) \cdot (X + Z) = X + Y \cdot Z$
18. $X + \overline{X} \cdot Y = X + Y$
19. $X \cdot Y + Y \cdot Z + \overline{X} \cdot Z = X \cdot Y + \overline{X} \cdot Z$

Boolean Algebra – DeMorgan's Law

To distribute the bar, change the operation.

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

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



Boolean Algebra

Example6: simplify the following function

$$OUT = \overline{A}\,\overline{B}D + \overline{A}BD + BCD + ACD$$

Boolean Algebra

Example6: simplify the following function

$$OUT = \overline{A}\,\overline{B}D + \overline{A}BD + BCD + ACD \quad \text{Rule 14}$$

$$OUT = \overline{A}D\left(\overline{B} + B\right) + BCD + ACD \quad \text{Rule 4}$$

$$OUT = \overline{A}D + BCD + ACD \quad \text{Rule 14}$$

$$OUT = D\left(\overline{A} + AC\right) + BCD \quad \text{Rule 18}$$

$$OUT = D\left(\overline{A} + C\right) + BCD \quad \text{Rule 14}$$

$$OUT = \overline{A}D + CD + BCD \quad \text{Rule 14}$$

$$OUT = CD(1 + B) + \overline{A}D \quad \text{Rule 2}$$

$$OUT = CD + \overline{A}D \quad \text{Rule 14}$$

$$OUT = D\left(\overline{A} + C\right)$$

Boolean Algebra

Example7: Simplify the equation created by the following truth table

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Algebra

Example7: Simplify the equation created by the following truth table

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$$Z = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$

Boolean Algebra

Example7: Simplify the equation created by the following truth table

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$$Z = \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$

$$Z = \overline{A}C\left(\overline{B} + B\right) + A\overline{B}\left(\overline{C} + C\right) + AB\left(\overline{C} + C\right)$$

$$Z = \overline{A}C + A\overline{B} + AB$$

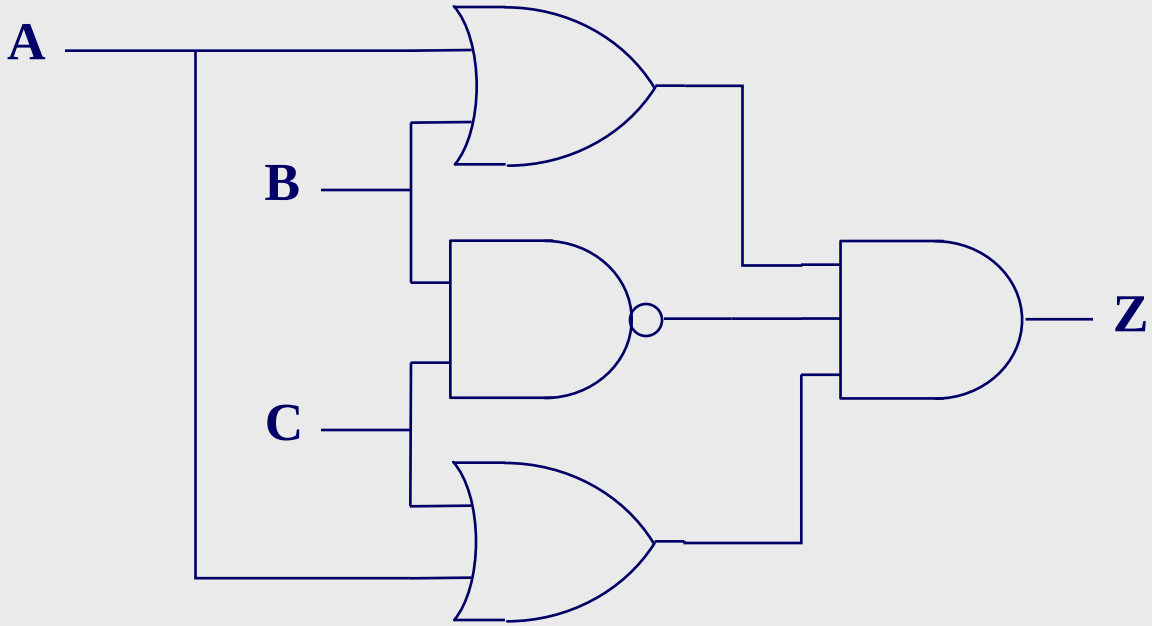
$$Z = \overline{A}C + A\left(\overline{B} + B\right)$$

$$Z = \overline{A}C + A$$

$$Z = A + C$$

Boolean Algebra

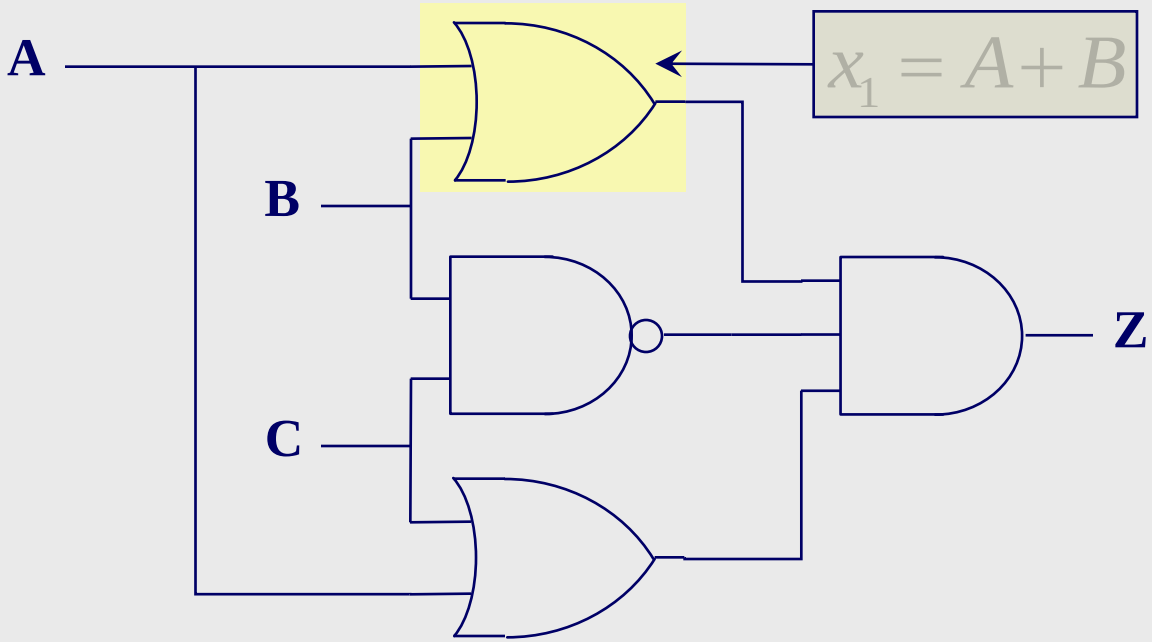
Example8: Determine the truth table



A	B	C	Z
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

Boolean Algebra

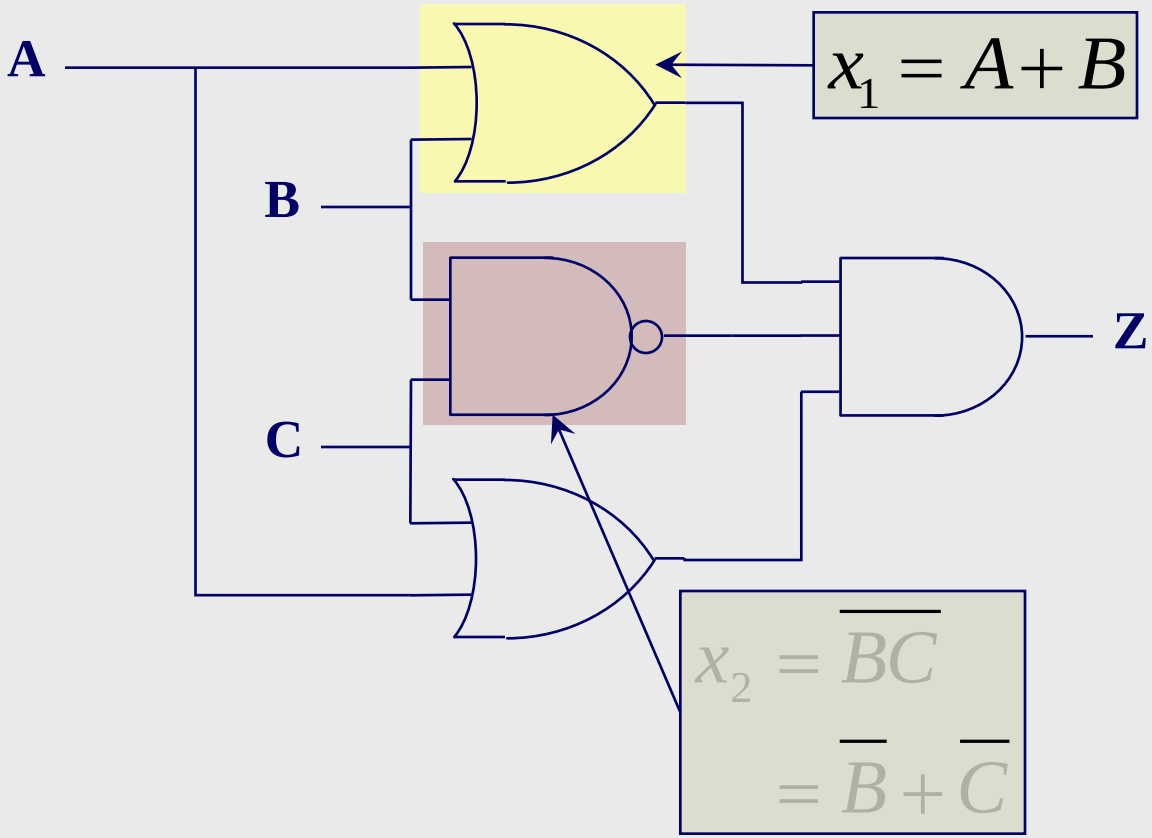
Example8: Determine the truth table



A	B	C	x_1	Z
0	0	0	0	?
0	0	1	0	?
0	1	0	1	?
0	1	1	1	?
1	0	0	1	?
1	0	1	1	?
1	1	0	1	?
1	1	1	1	?

Boolean Algebra

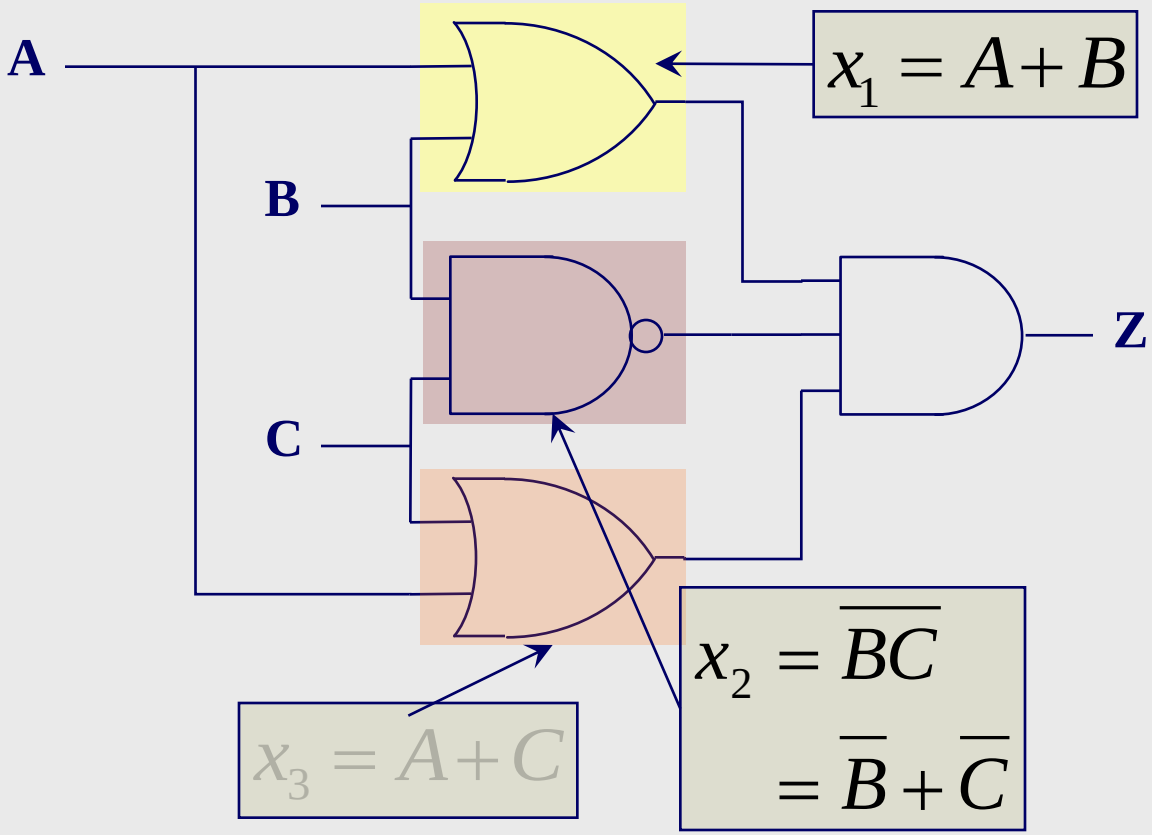
Example8: Determine the truth table



A	B	C	x_1	x_2	Z
0	0	0	0	1	?
0	0	1	0	1	?
0	1	0	1	1	?
0	1	1	1	0	?
1	0	0	1	1	?
1	0	1	1	1	?
1	1	0	1	1	?
1	1	1	1	0	?

Boolean Algebra

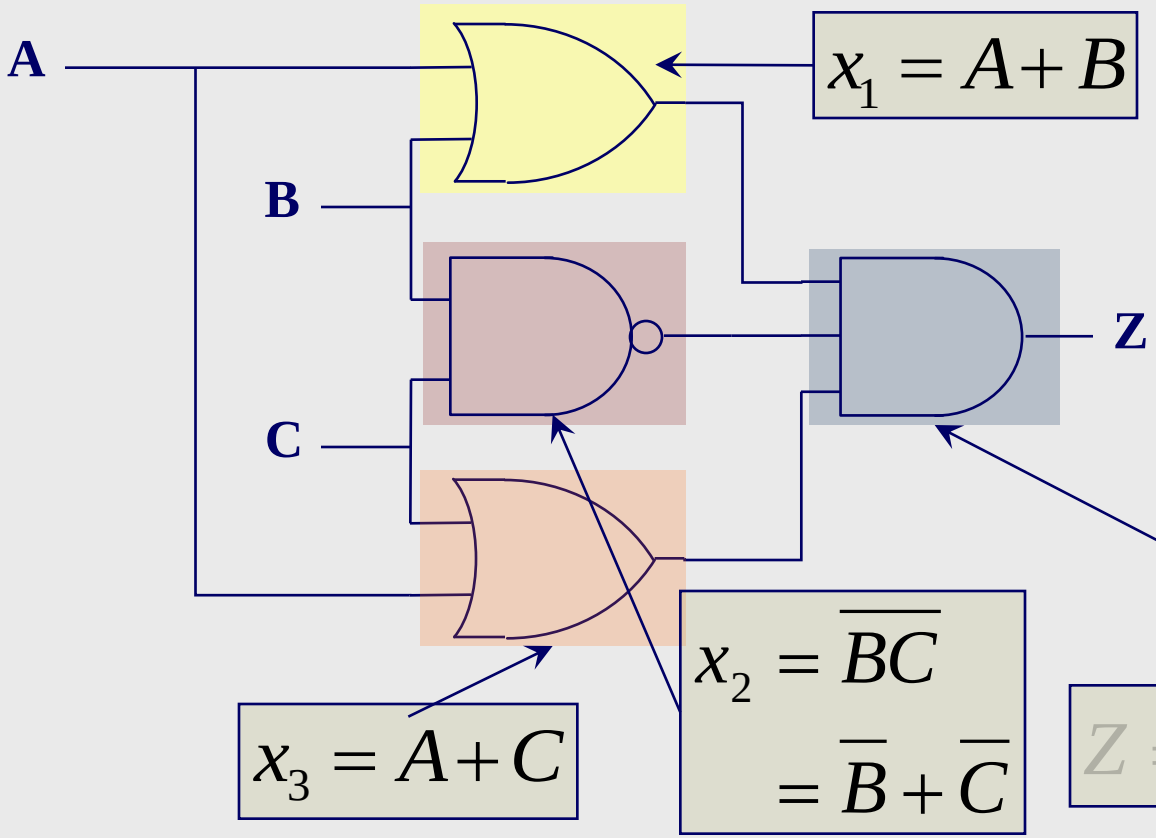
Example8: Determine the truth table



A	B	C	x_1	x_2	x_3	Z
0	0	0	0	1	0	?
0	0	1	0	1	1	?
0	1	0	1	1	0	?
0	1	1	1	0	1	?
1	0	0	1	1	1	?
1	0	1	1	1	1	?
1	1	0	1	1	1	?
1	1	1	1	0	1	?

Boolean Algebra

Example8: Determine the truth table



A	B	C	x_1	x_2	x_3	Z
0	0	0	0	1	0	0
0	0	1	0	1	1	0
0	1	0	1	1	0	0
0	1	1	1	0	1	0
1	0	0	1	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	0	1	0

Boolean Algebra

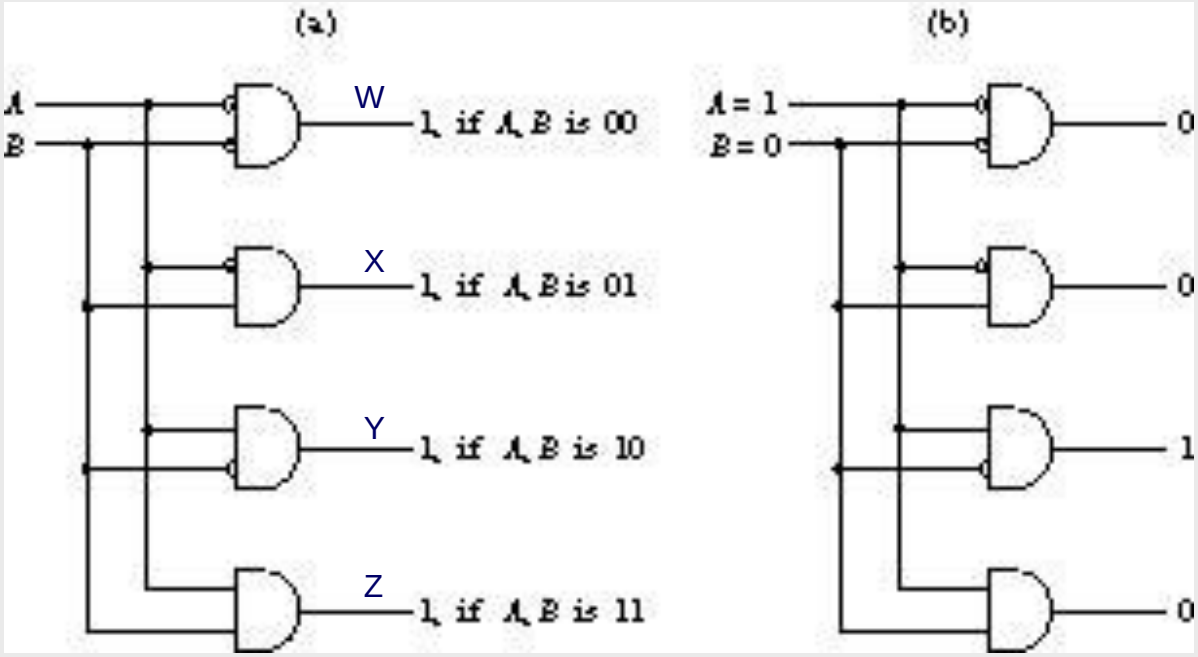
Example8: Determine the truth table

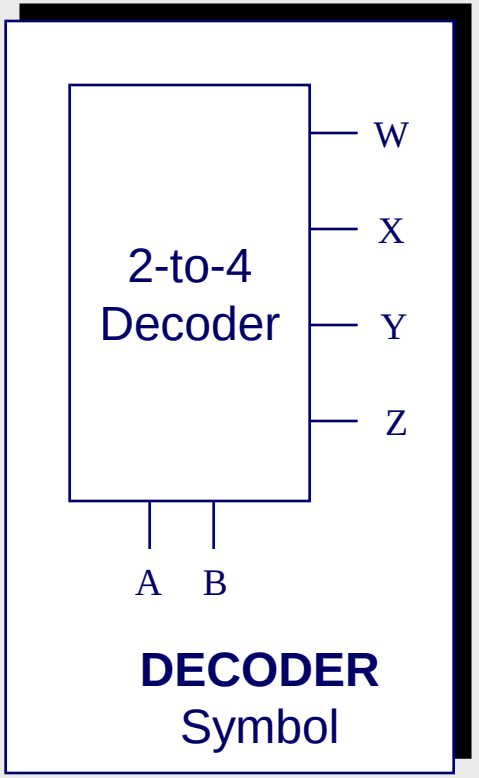
$$
\begin{aligned}
Z &= (A+B)\overline{(BC)}(A+C) \\
&= A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} \\
&= A\overline{C}(\overline{B}+B) + A\overline{B}C \\
&= A\overline{C} + A\overline{B}C \\
&= A(\overline{C}+\overline{B}C) \\
&= A(\overline{C}+\overline{B})
\end{aligned}
$$

A	B	C	x_1	x_2	x_3	Z
0	0	0	0	1	0	0
0	0	1	0	1	1	0
0	1	0	1	1	0	0
0	1	1	1	0	1	0
1	0	0	1	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	0	1	0

Combinational Logic – Decoders

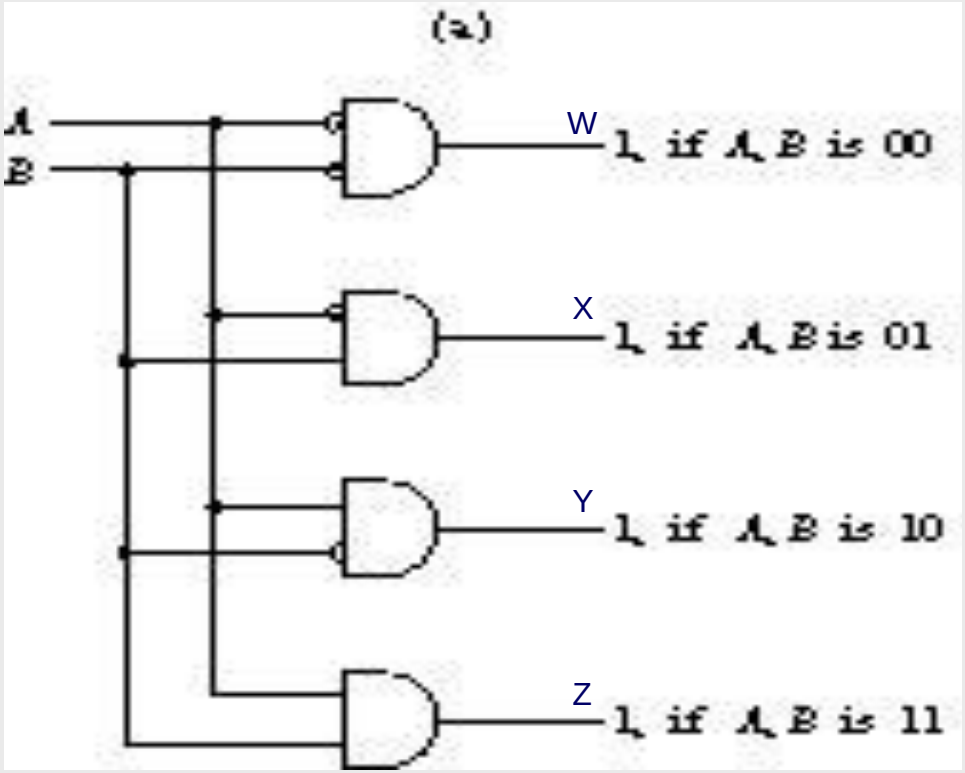
- Decode the input and signify its value by raising **just one** of its outputs.
- A decoder with **n** inputs has 2^n outputs





Combinational Logic – Decoders

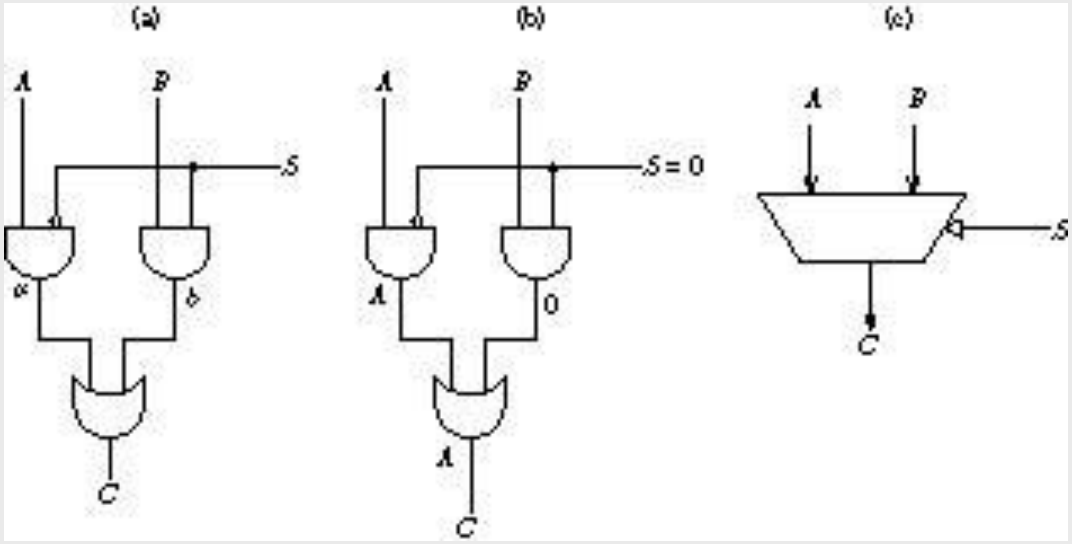
- Write the truth table

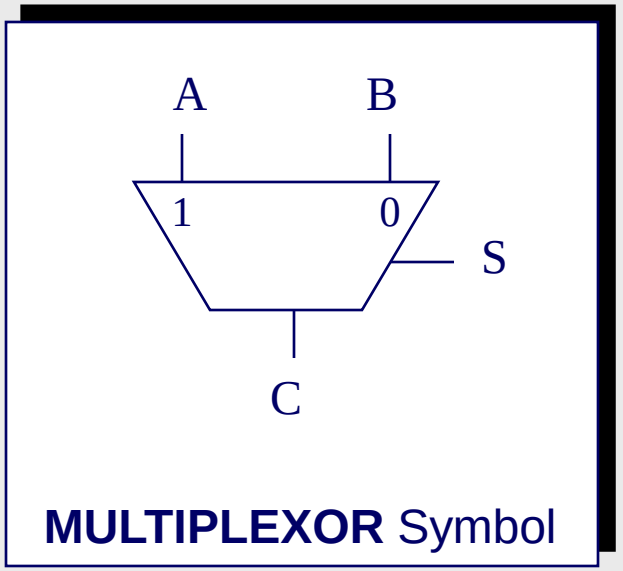


A	B	W	X	Y	Z
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Combinational Logic – Multiplexors

- Connect one of its inputs to its output according to select signals



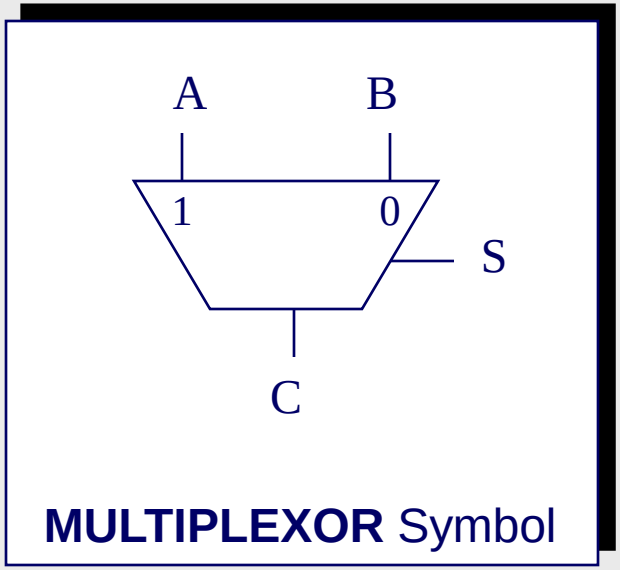


- Useful for *selecting* one from a collection of data inputs.
- Usually has 2^n inputs and **n** select lines.

Combinational Logic – Multiplexors

- Write the truth table

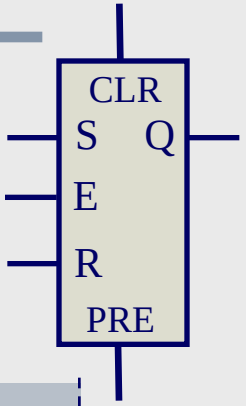
A	B	S	C
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Sequential Logic – SR Latch

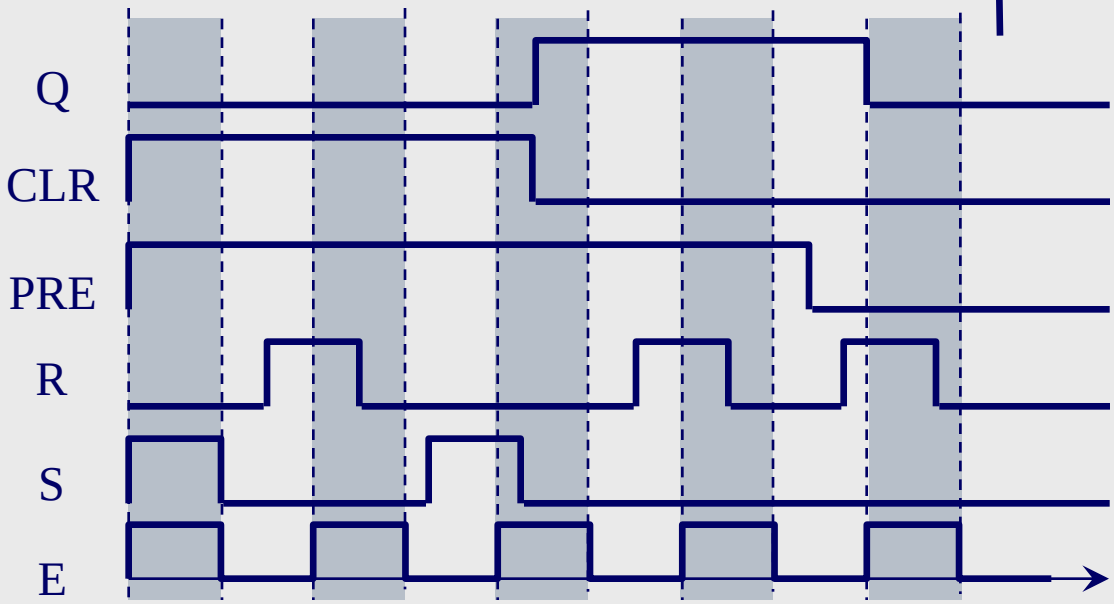
SR Latch with additional inputs:

- **Enable** (E) – S and R can only change Q when E is 1
- **Preset** (PRE) – regardless of S, R, or E, put Q to 1 when PRE is 1
- **Clear** (CLR) – regardless of S, R, E, or PRE, put Q to 0 when CLR is 1



Precedence:

1. If CLR = 1, Q = 0
2. If PRE = 1, Q = 1
3. If E = 1, Q is set based on SR:
   - If S = 0 and R = 0, Q = held
   - If S = 0 and R = 1, Q = 0
   - If S = 1 and R = 0, Q = 1
   - If S = 1 and R = 1, Q = unstable
4. Else Q is held



SR can only change **Q** only in blue regions (where **E** = 1) BUT **CLR** and **PRE** will change **Q** ANYTIME

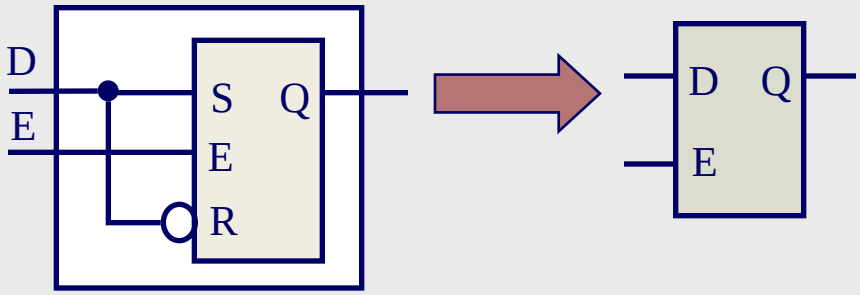
Sequential Logic – D Latch

D Latch has only 2 states:

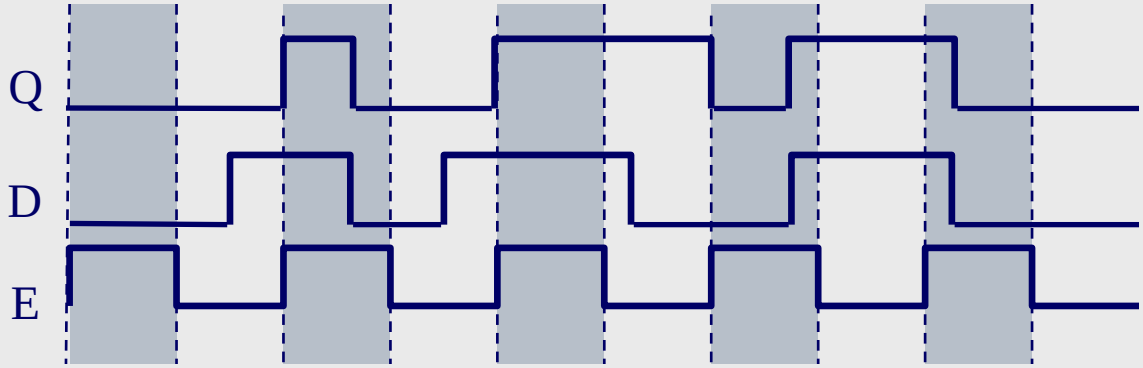
- **Set** (set Q to 1): D = 1
- **Reset** (reset Q to 0): D = 0

D Latch with enable (E):

- Q can only change when E = 1



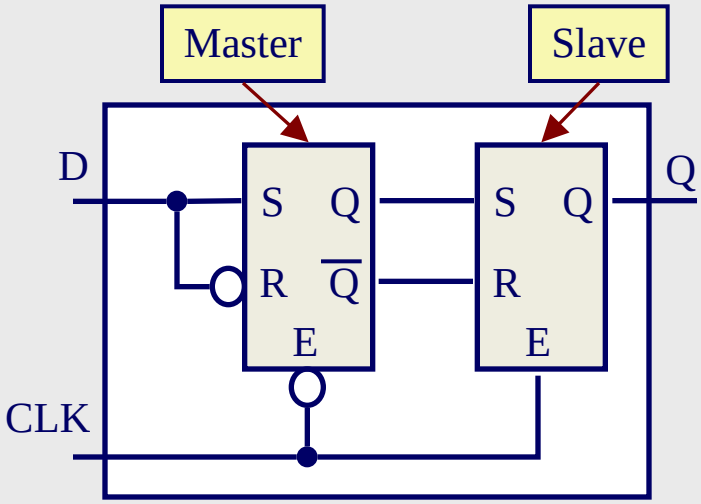
E	D	Q_{new}
0	0	Q_{old}
0	1	Q_{old}
1	0	0
1	1	1

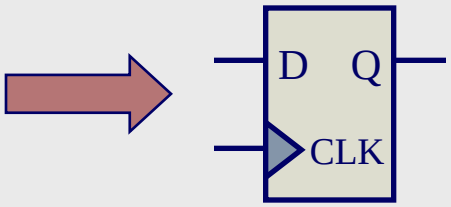


D can only change Q only in blue regions (where E = 1)

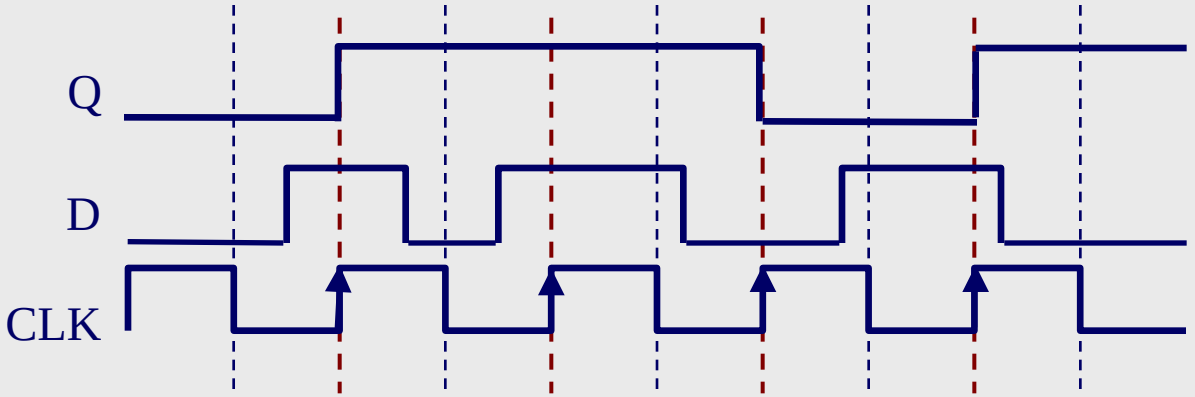
Sequential Logic – D Flip-Flop

D FF: 2 SR latches in master/slave configuration. The output (Q) changes on the **rising clock edge**





D	CLK	Q_{new}
0	↑	0
1	↑	1



D can only change Q only on rising clock edge (arrows)

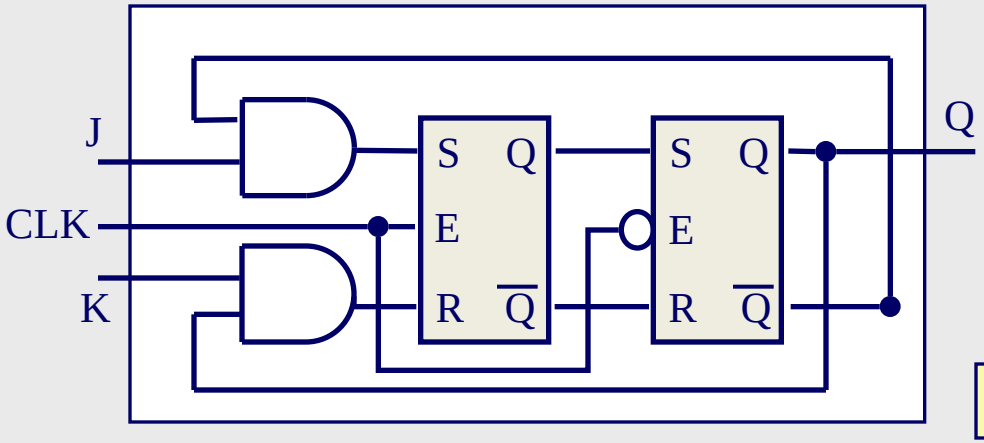
Sequential Logic – JK Flip-Flop

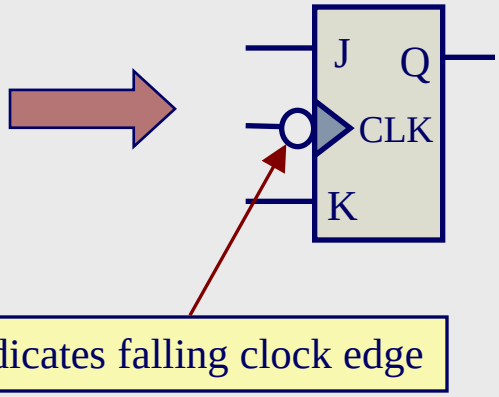
JK FF: 2 SR latches in master/slave configuration.
The output (Q) changes on the **falling clock edge**

JK FF has 4 allowed states:

- **Present state** (keep Q as is): J = 0, K = 0
- **Reset** (reset Q to 0): J = 0, K = 1
- **Set** (set Q to 1): J = 1, K = 0
- **Toggle** (set Q to $\overline{Q}$): J = 1, K = 1

J	K	CLK	Q_{new}
0	0	↓	Q_{old}
0	1	↓	0
1	0	↓	1
1	1	↓	$\overline{Q_{old}}$





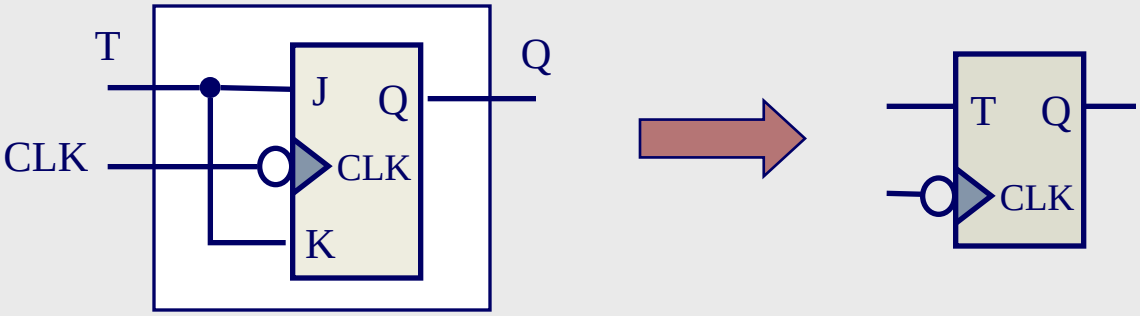
Sequential Logic – T Flip-Flop

T FF: JK FF with J and K inputs connected

T FF has 2 allowed states:

- **Present state** (keep Q as is): T = 0
- **Toggle** (set Q to $\overline{Q}$): T = 1

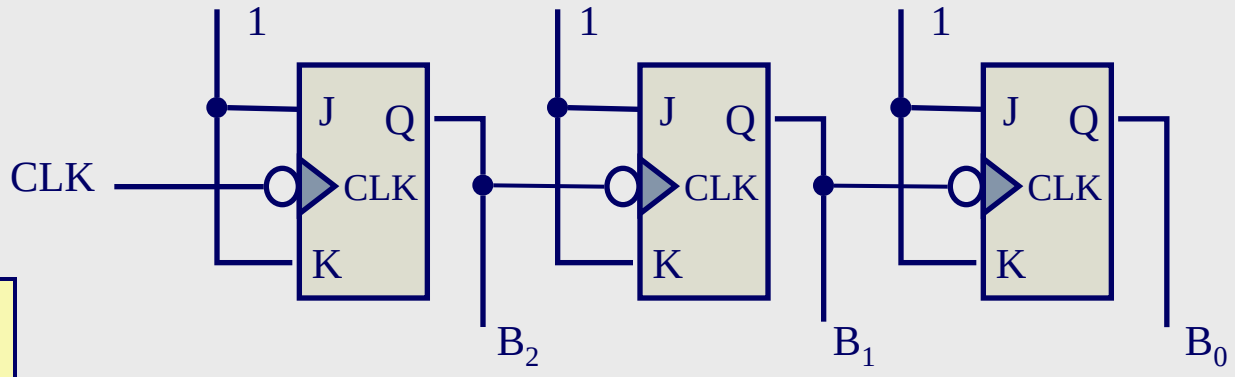
T	CLK	Q_{new}
0	↓	Q_{old}
1	↓	$\overline{Q_{old}}$



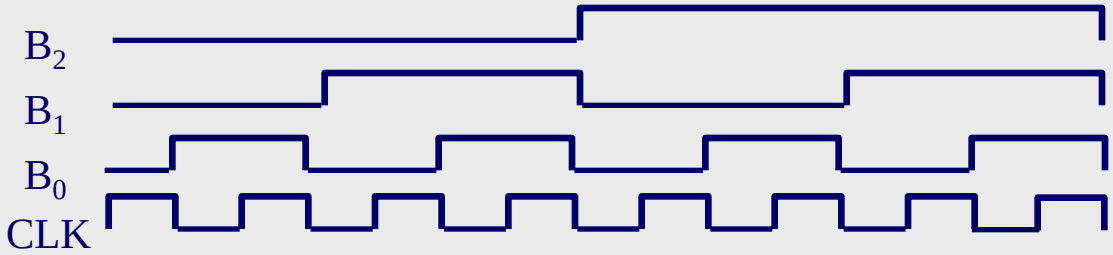
Sequential Logic – Digital Counters

Ripple counter: with N bits, cycles through the numbers from 0 to $2^N - 1$

- N JK FFs cascaded together to produce an N-bit up counter



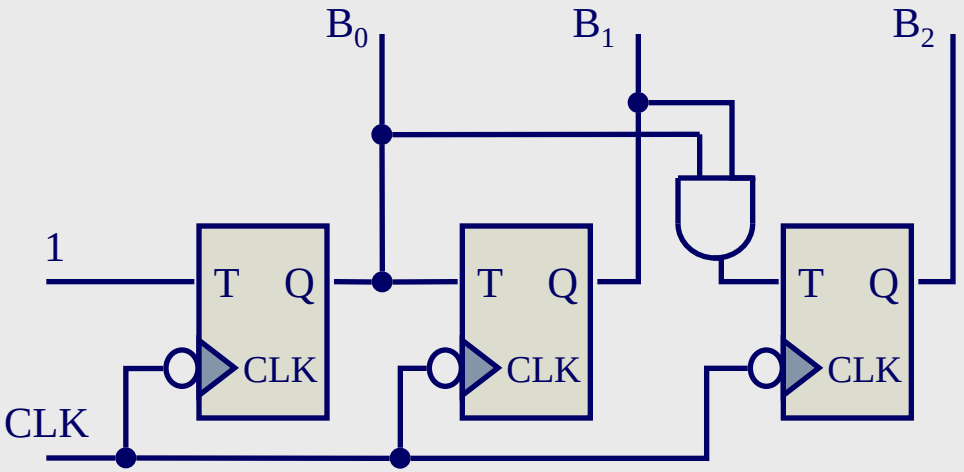
NB: for 3-bit counter we need 3 FFs

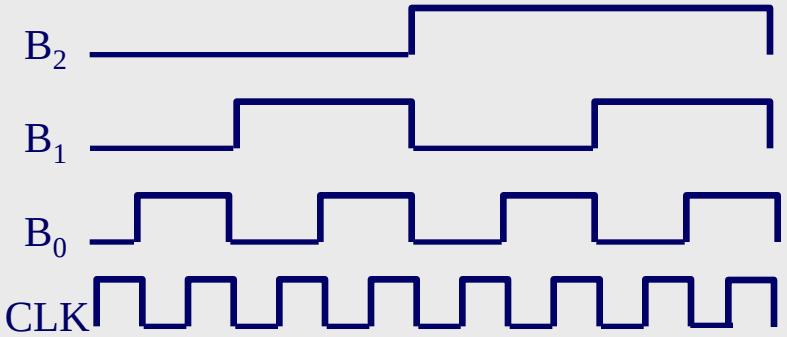


Sequential Logic – Digital Counters

Synchronous counter: with N bits, cycles through the numbers from 0 to $2^N - 1$

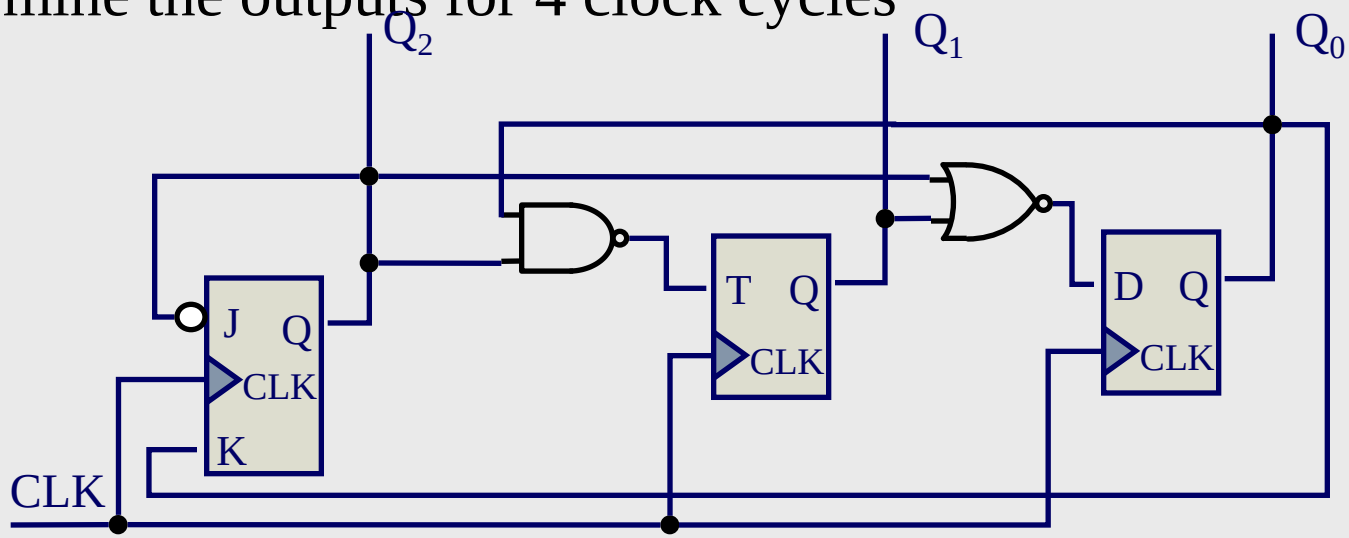
- Input clock drives all FFs simultaneously





Sequential Logic

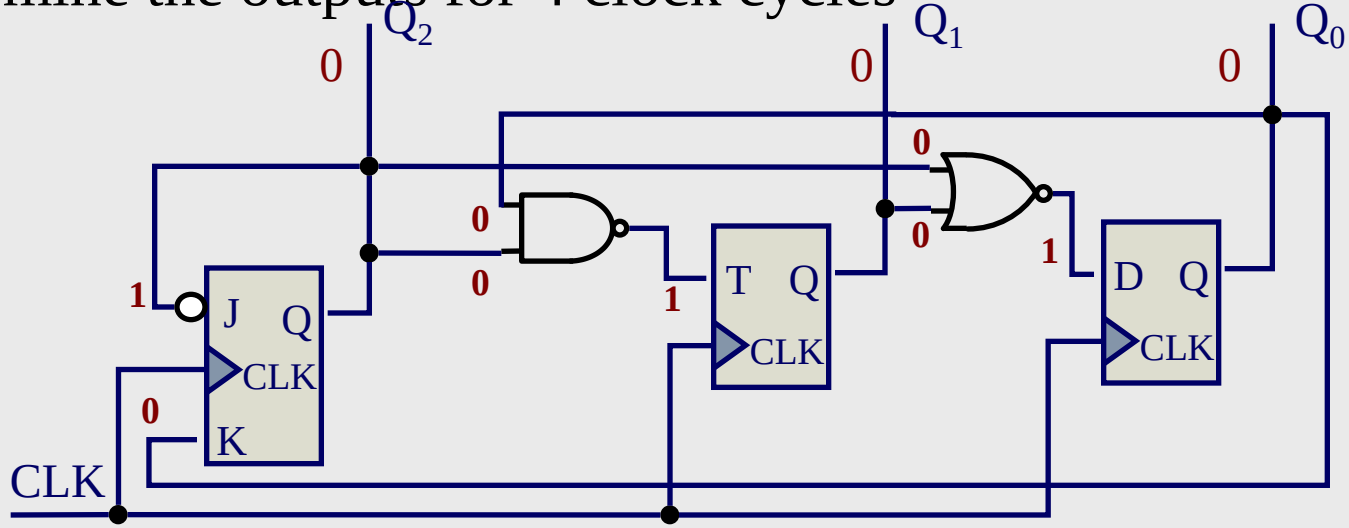
Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles



Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

1. Set outputs to 000
2. Based on output values change FF inputs
3. On each clock cycle:
   a) change FF outputs based on inputs
   b) Change FF inputs based on new outputs



J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

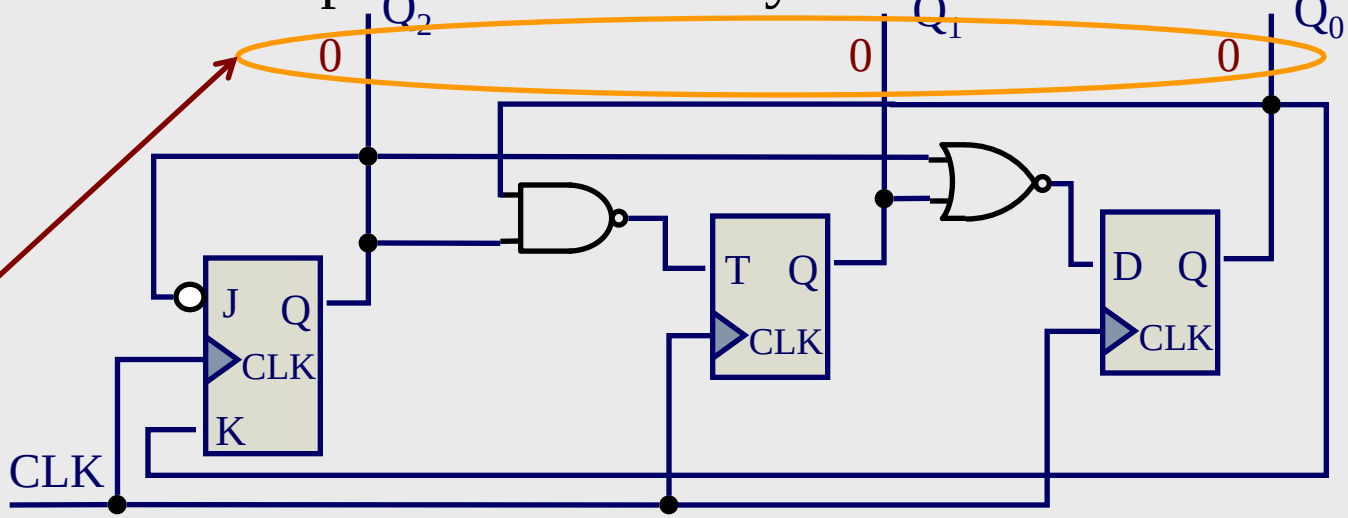
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

1. Set outputs to 000
2. Based on output values change FF inputs
3. On each clock cycle:
   a) change **ALL** FF outputs based on inputs



J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

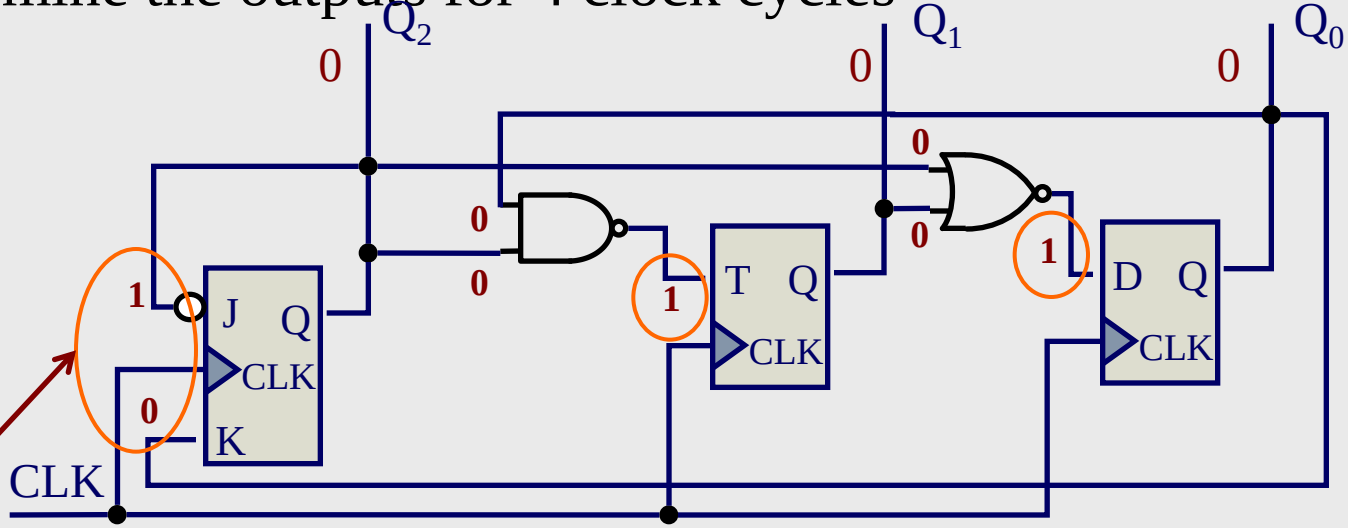
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

1. Set outputs to 000
2. Based on output values change FF inputs
3. On each clock cycle:
   a) change **ALL** FF outputs based on inputs
   b) Change **ALL** FF inputs based on new outputs



J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

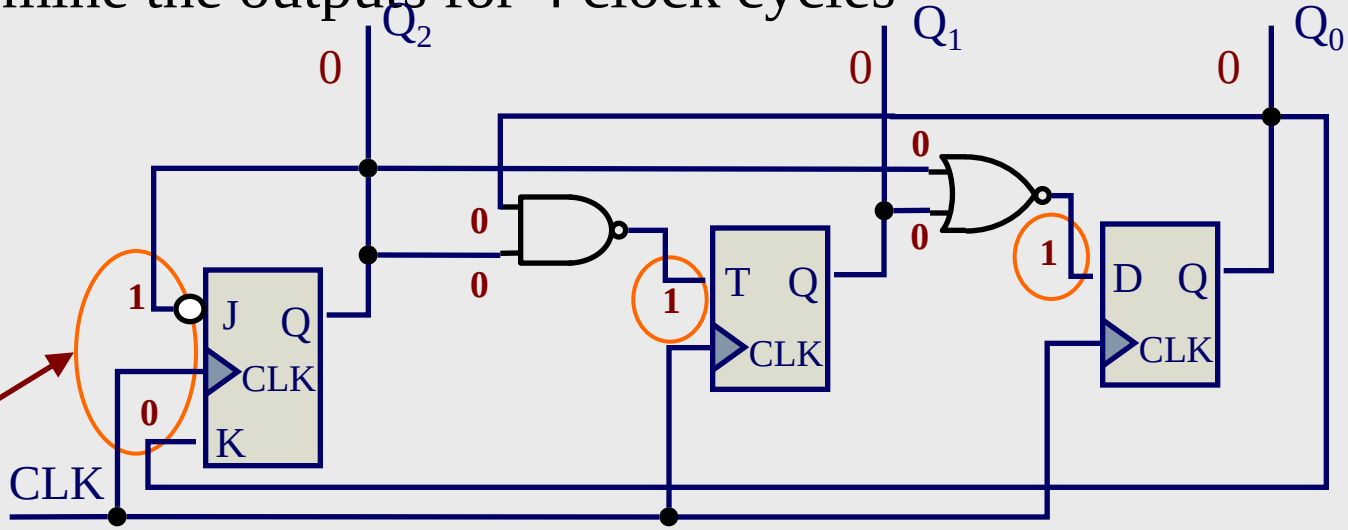
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0



Inputs changed due to outputs

J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

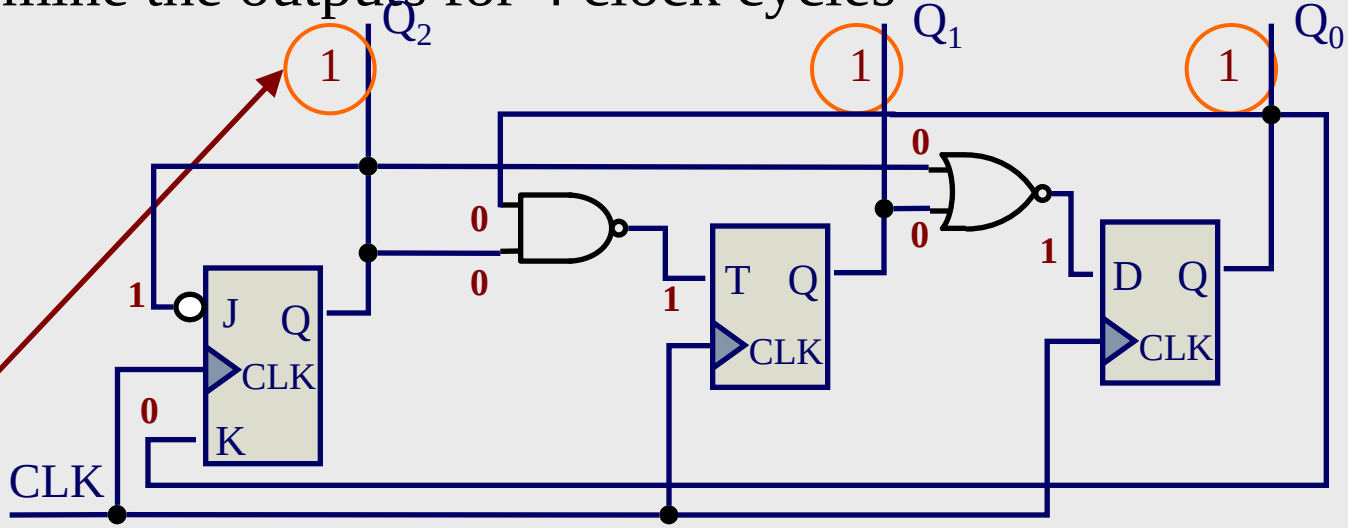
D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0
1	0	1	1

Outputs change on **new clock cycle**



J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

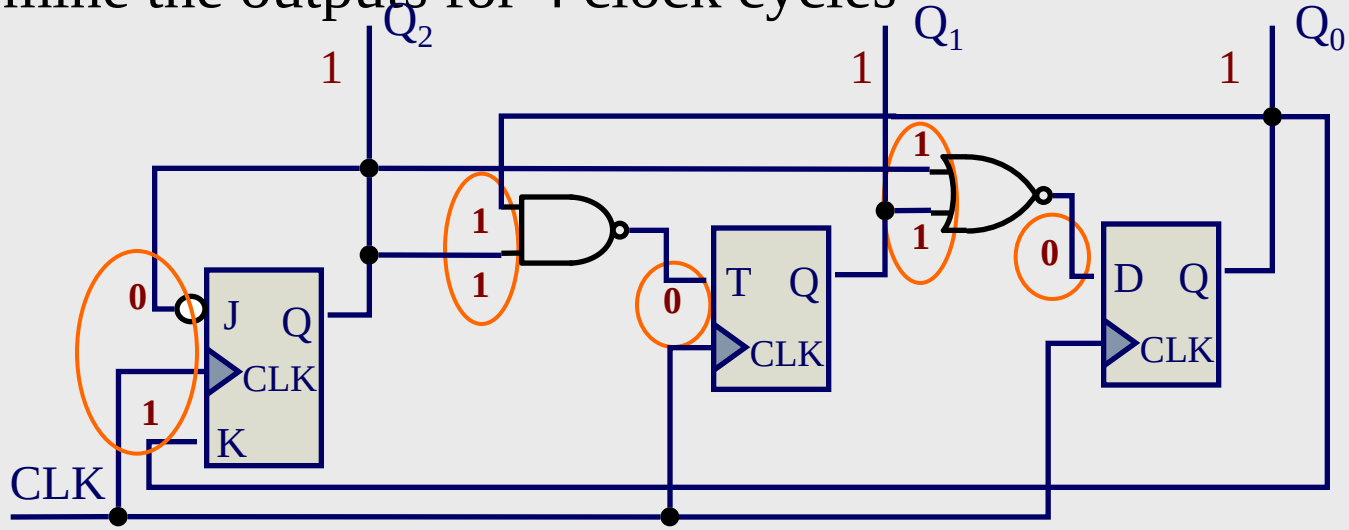
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0
1	1	1	1



Inputs changed due to outputs

J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

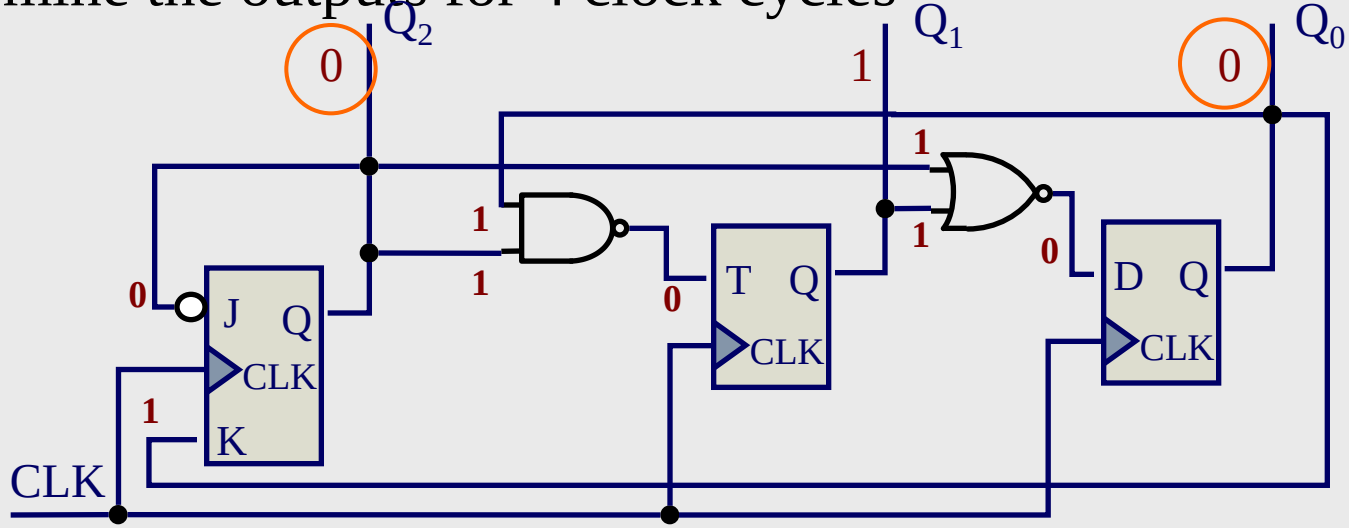
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0
1	0	1	1
2	0	1	0



Outputs change on **new clock cycle**

J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

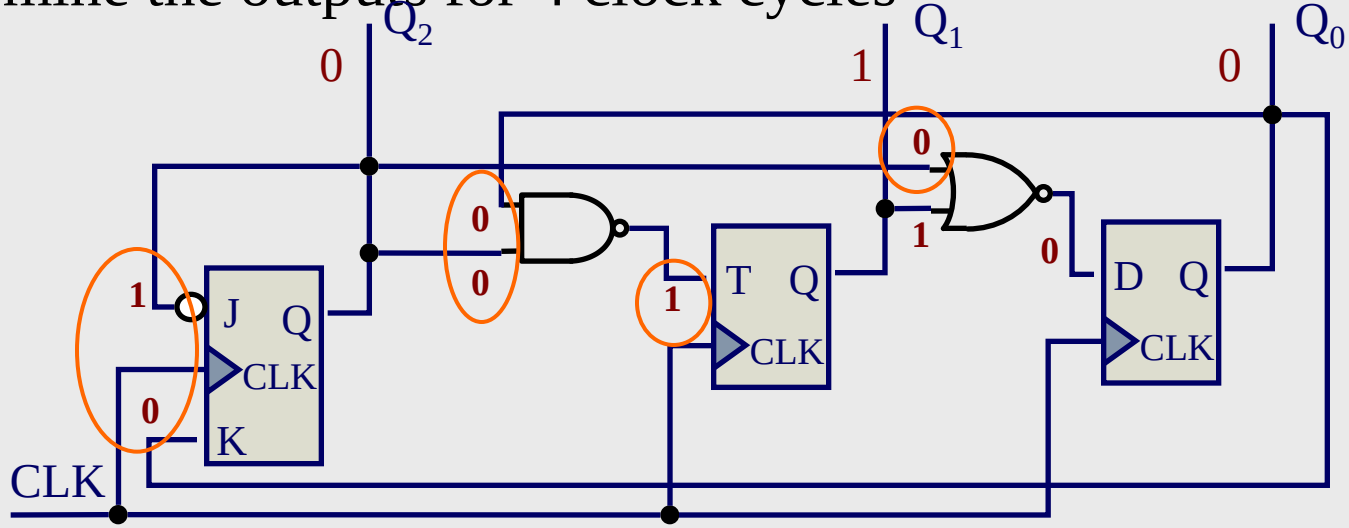
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0
1	0	1	1
2	0	1	0



Inputs changed due to outputs

J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

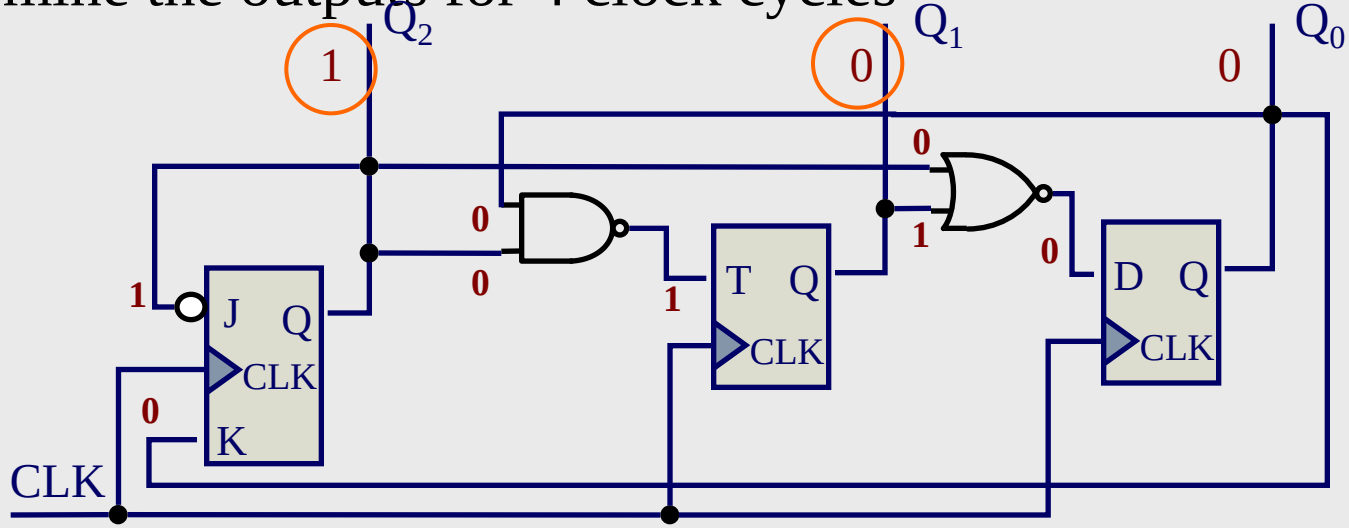
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0
1	0	1	1
2	0	1	0
3	1	0	0



Outputs change on **new clock cycle**

J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

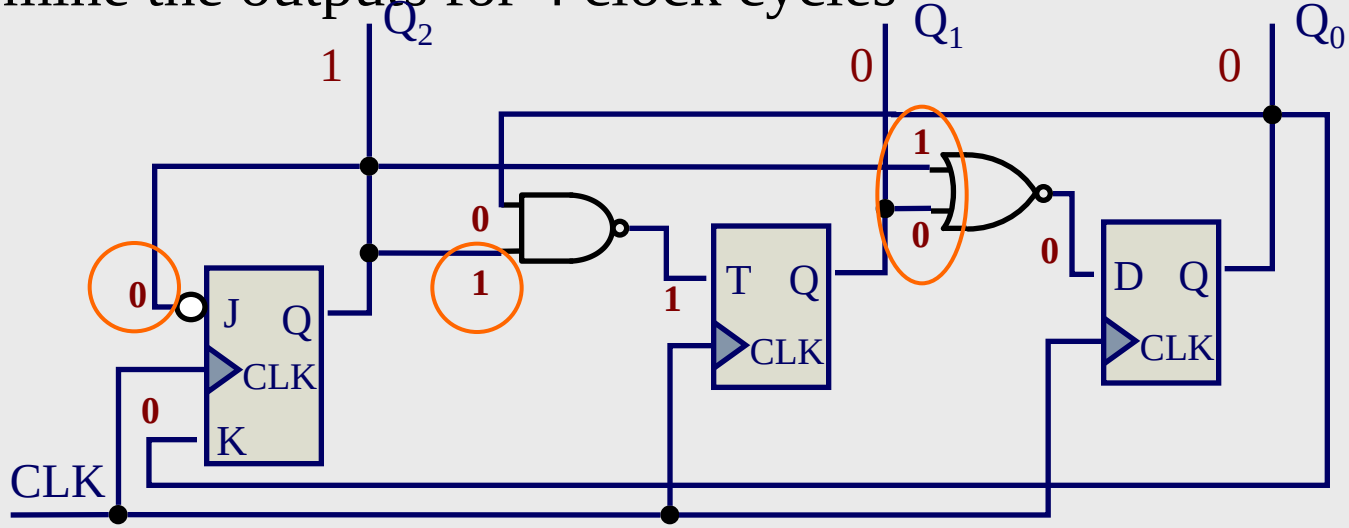
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0
1	0	1	1
2	0	1	0
3	1	0	0



Inputs changed due to outputs

J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

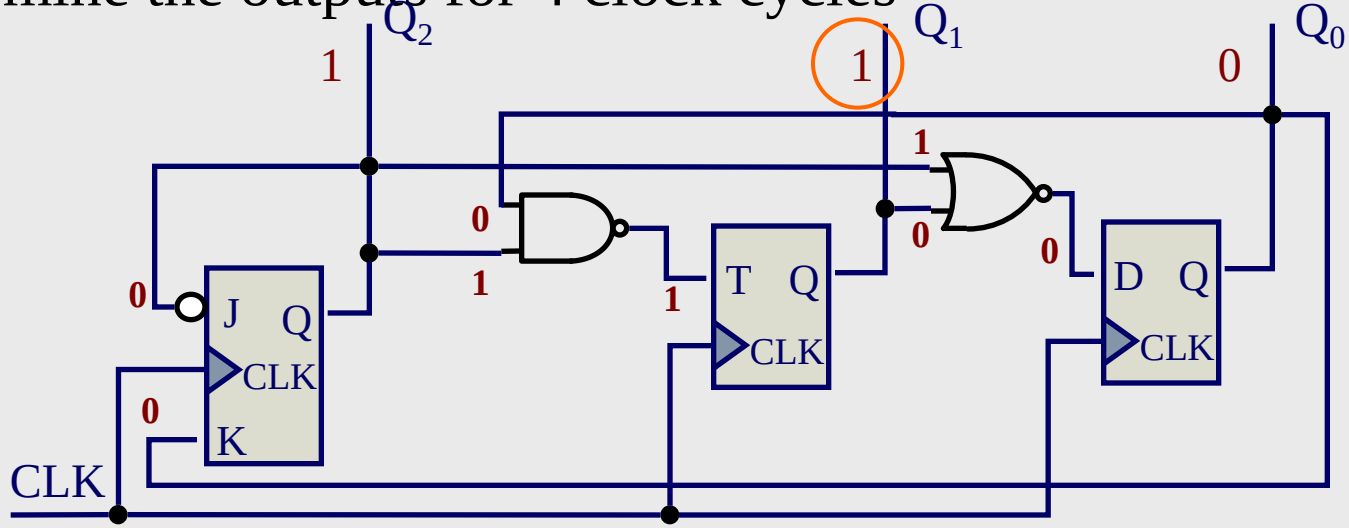
T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Sequential Logic

Example9: Assuming the outputs of the following circuit start in a 000 state, determine the outputs for 4 clock cycles

Cycle	Q_2	Q_1	Q_0
start	0	0	0
1	0	1	1
2	0	1	0
3	1	0	0
4	1	1	0



Outputs change on **new clock cycle**

J	K	CLK	Q_{new}
0	0	↑	Q_{old}
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_{old}}$

T	CLK	Q_{new}
0	↑	Q_{old}
1	↑	$\overline{Q_{old}}$

D	CLK	Q_{new}
0	↑	0
1	↑	1

Digital to Analog Converter (DAC)

DAC: converts an **unsigned binary word** to an analog output voltage or current

Resolution δv: minimum step size by which the output voltage (or current) can increment

Output voltage v_a: the analog value represented by the binary word B

- EX: let n=4

$$v_a = (2^3 \cdot b_3 + 2^2 \cdot b_2 + 2^1 \cdot b_1 + 2^0 \cdot b_0)\delta v$$

Max output voltage v_{aMax}: the maximum analog value

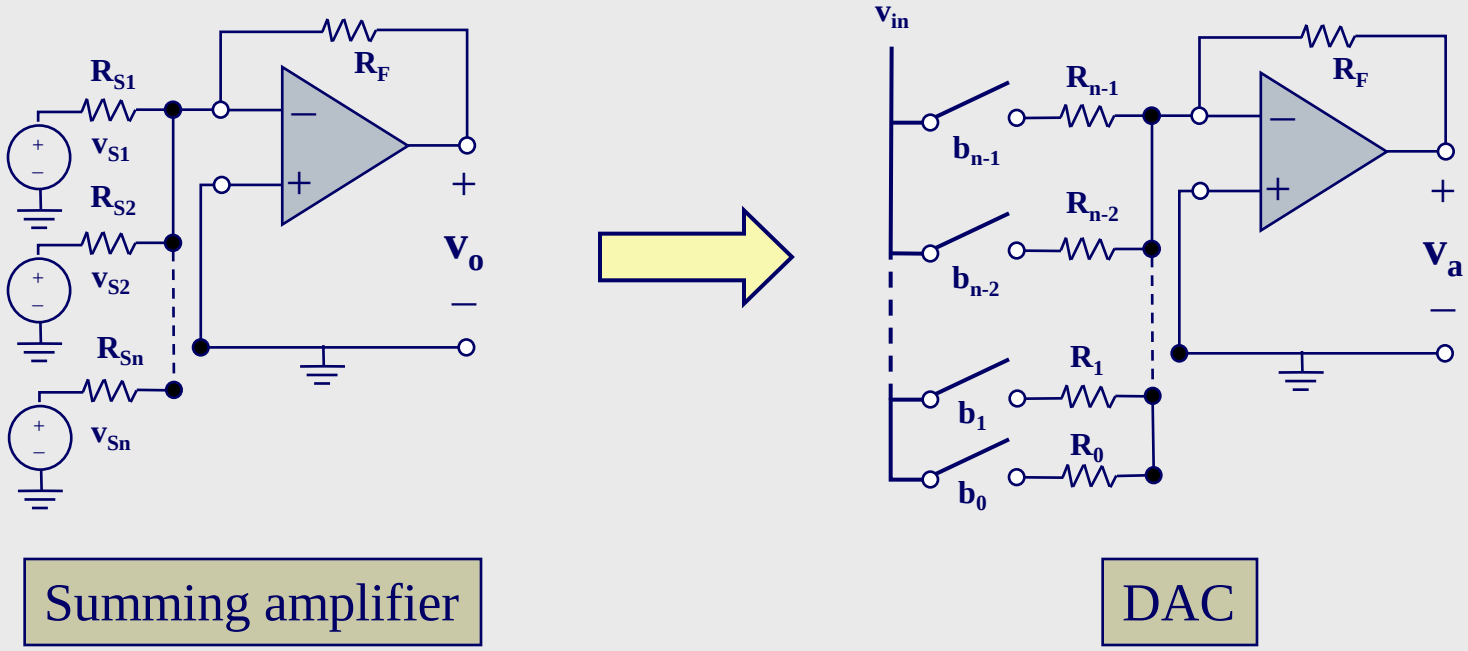
- EX: let n=4

$$v_{aMax} = (2^3 + 2^2 + 2^1 + 2^0)\delta v$$
$$= (2^n - 1)\,\delta v$$

Digital to Analog Converter (DAC)

Building a DAC:

use a summing amplifier



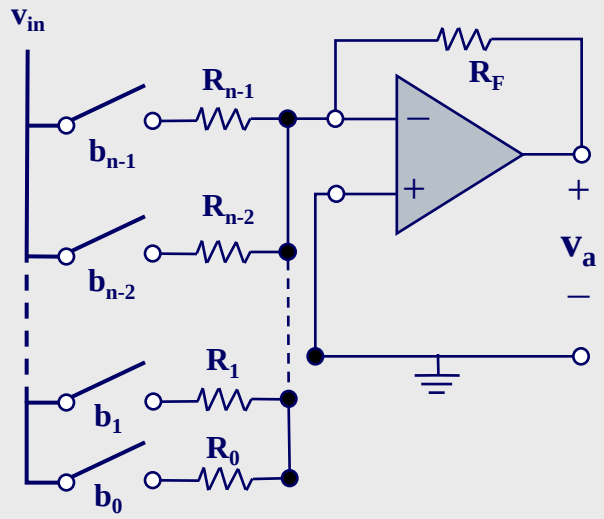
Summing amplifier

DAC

Digital to Analog Converter (DAC)

Building a DAC:

the analog output ($\mathbf{v_a}$) is proportional to the binary word **B**



$$v_a = -\sum_{i=0}^{n-1} \frac{R_F}{R_i} \cdot b_i \cdot v_{in}$$

$\mathbf{v_{si}}$ in summing amplifier

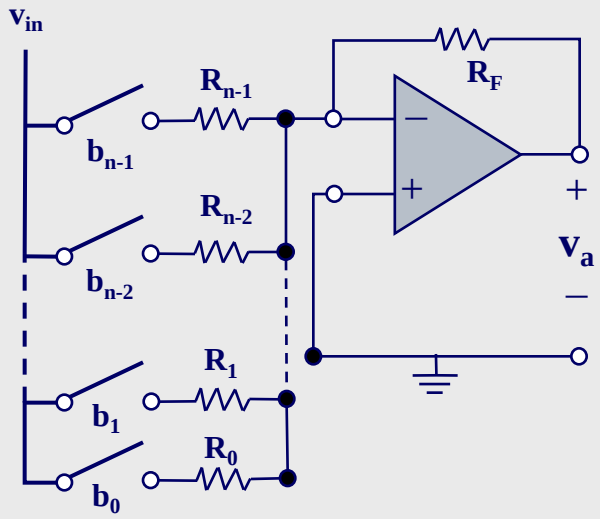
Choose $R_i = \frac{R_0}{2^i}$

THUS $v_a = -\frac{R_F}{R_0} \cdot v_{in} \sum_{i=0}^{n-1} 2^i \cdot b_i$

Digital to Analog Converter (DAC)

Example10: find the smallest resolution δv of an 8-bit DAC

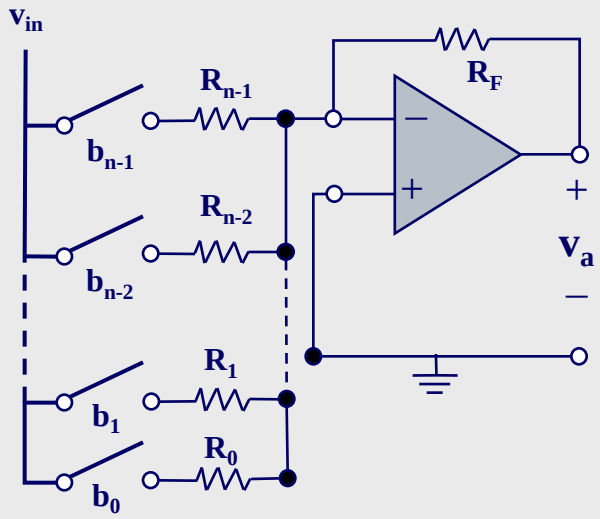
$v_{aMax} = 12V$



Digital to Analog Converter (DAC)

Example10: find the smallest resolution δv of an 8-bit DAC

$v_{aMax} = 12V$



$$v_{a\max} = \left(2^n - 1\right)\delta v$$

$$\delta v = \frac{v_{a\max} - v_{a\min}}{\left(2^n - 1\right)}$$

$$= \frac{12 - 0}{\left(2^8 - 1\right)}$$

$$= 47.1\,mV$$

Digital to Analog Converter (DAC)

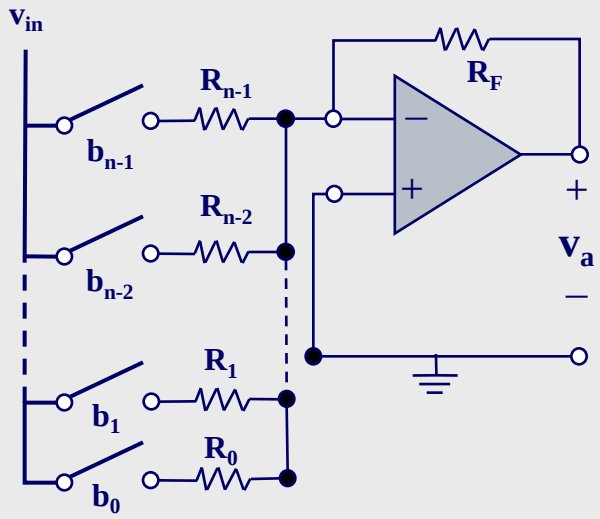
Example11: find the resistor values for a DAC with:

range = 15V

δv = 1V

v_{in} = 5V

R_F = 2kΩ



Digital to Analog Converter (DAC)

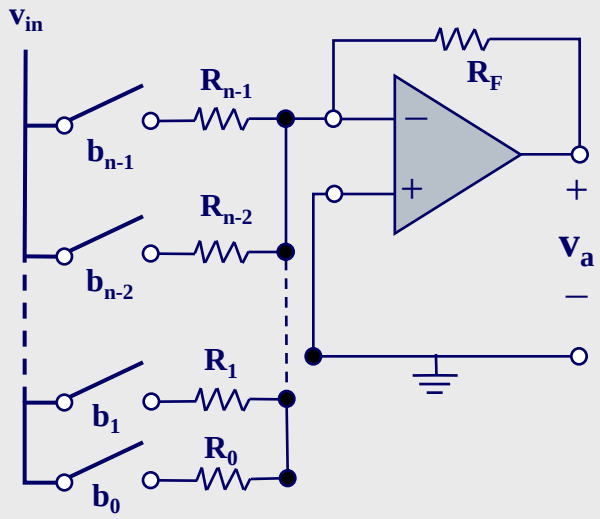
Example11: find the resistor values for a DAC with:

range = 15V

δv = 1V

v_{in} = 5V

R_F = 2kΩ



$$n = \left\lceil \frac{\log\left[(v_{a\max} - v_{a\min})/\delta v + 1\right]}{\log 2} \right\rceil$$

$$= \left\lceil \frac{\log(15/1+1)}{\log 2} \right\rceil$$

$$= \lceil 4 \rceil$$

$$= 4$$

Digital to Analog Converter (DAC)

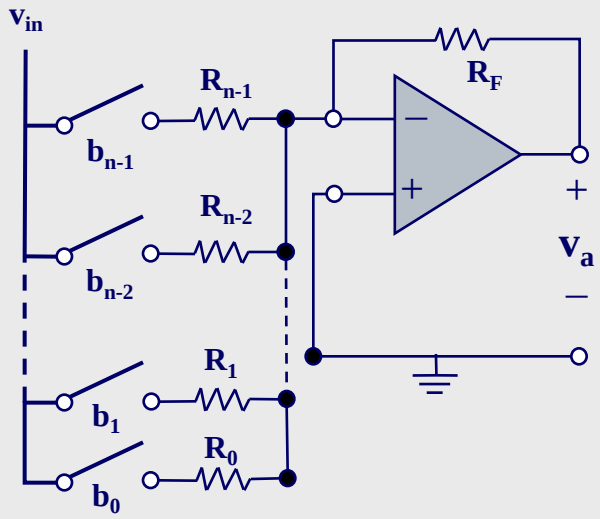
Example11: find the resistor values for a DAC with:

range = 15V

δv = 1V

v_{in} = 5V

R_F = 2kΩ



$$v_a = -\frac{R_F}{R_0} \cdot v_{in} \sum_{i=0}^{n-1} 2^i \cdot b_i$$

$$v_{a\max} = \frac{R_F}{R_0} \cdot v_{in} \left(2^n - 1\right)$$

$$R_0 = \frac{R_F}{v_{a\max}} \cdot v_{in} \left(2^n - 1\right)$$

$$= \frac{2 \times 10^3}{15} \left(5\right)\left(2^4 - 1\right)$$

$$= 10\ k\Omega$$

Digital to Analog Converter (DAC)

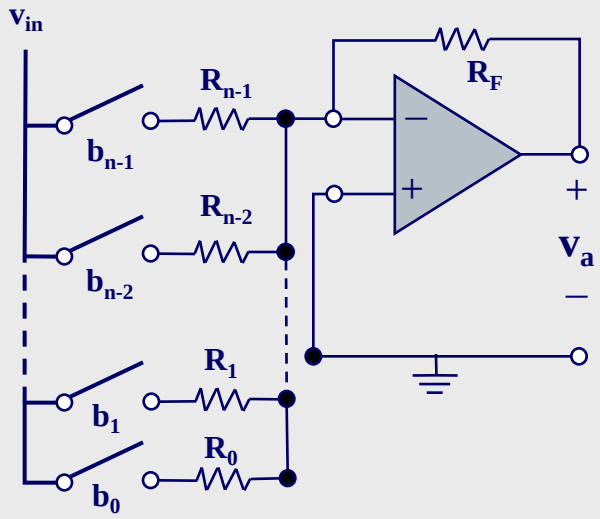
Example11: find the resistor values for a DAC with:

range = 15V

δv = 1V

v_{in} = 5V

R_F = 2kΩ



$$R_i = \frac{R_0}{2^i}$$

$$R_1 = \frac{R_0}{2^1} = \frac{10^3}{2} = 5k\Omega$$

$$R_2 = \frac{R_0}{2^2} = \frac{10^3}{4} = 2.5k\Omega$$

$$R_3 = \frac{R_0}{2^3} = \frac{10^3}{8} = 1.25k\Omega$$