Ch5: Digital Circuits – Part 1 (Combinational Logic)

Contents:

Number Systems

Logic Gates

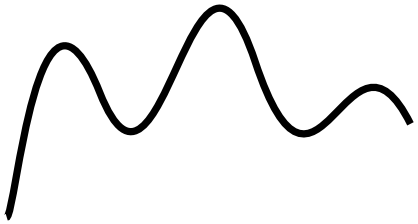
Boolean Algebra

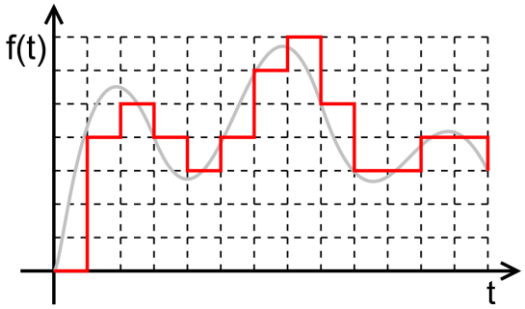
Application

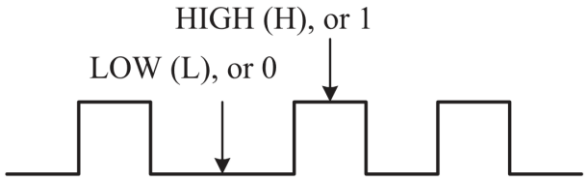
Number Systems

Analog and Digital Signals

In contrast to an **analog signal** that changes in a **continuous** manner, a **digital signal** exists only at specific **discrete** steps. Most digital signals have only two states (HIGH and LOW) and it is called a **binary signal** or **logic signal**.







Digital devices convert digital inputs into one or more digital outputs, and they are categorized as **Combinational Logic** or **Sequential Logic**.

Number Systems

The **base** (b) of a number system indicates the number of different symbols (d_i) that can be used to represent a digit.

$$\underbrace{(d_{n-1} \dots d_3 d_2 d_1 d_0)_b}_{\text{Base } b \text{ number system}} = \underbrace{(d_{n-1} \cdot b^{n-1} + \cdots + d_2 \cdot b^2 + d_1 \cdot b^1 + d_0 \cdot b^0)}_{\text{Decimal number}}$$

- To convert a number **from decimal to an arbitrary base**, the procedure is to successively divide the decimal number by the base and record the remainders after each division. The remainders, when written in reverse order from left to right, form the digits of the number represented in the new base.

Decimal number: $b = 10$, $d_i = 0 - 9$, Example: $123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

(Fractions may also be included if digits for negative powers of 10 are included: d_{-1}, d_{-2},…)

Binary Number System

Since the operation of **digital devices** is based on transistors that switch between two states: the **saturated state** (**ON**, **High**, **True**, or **1**) and the **cutoff state** (**OFF**, **Low**, **False**, or **0**), a base 2 number system called the **Binary Number System** is used to represent and manipulate numbers with digital devices.

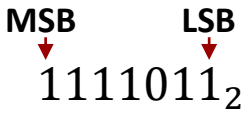
Binary number: $b = 2, d_i = 0,1,$ Example:

$$1111011_2$$
$$= 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$= 123$$

Decimal to Binary:

Successive Divisions	Remainder	
123/2	1	LSB
61/2	1	
30/2	0	
15/2	1	
7/2	1	
3/2	1	
1/2	1	MSB
Result	1111011	

The digits of a binary number are also called **bits**. The first, or leftmost, bit is known as the **most significant bit** (**MSB**) because it represents the **largest** power of 2. The last, or rightmost, bit is known as the **least significant bit** (**LSB**) because it represents the **smallest** power of 2. A **group of 8 bits** is called a **byte** (0→255).



Example: Binary Number System

Binary-to-Decimal Conversion

109_{10} to binary

109/2 = 54 w/ remainder 1 (LSB)
54/2 = 27 w/ remainder 0
27/2 = 13 w/ remainder 1
13/2 = 6 w/ remainder 1
6/2 = 3 w/ remainder 0
3/2 = 1 w/ remainder 1
1/2 = 0 w/remainder 1 (MSB)

Answer: 1101101
8-bit answer: 01101101

Take decimal number and keep dividing by 2, while keeping the remainders. The first remainder becomes the LSB, while the last one becomes the MSB.

Decimal-to-Binary Conversion

10100100 to decimal

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
(MSB) 1	0	1	0	0	1	0	0 (LSB)

$0 \times 2^0 = 0$
$0 \times 2^1 = 0$
$1 \times 2^2 = 4$
$0 \times 2^3 = 0$
$0 \times 2^4 = 0$
$1 \times 2^5 = 32$
$0 \times 2^6 = 0$
$1 \times 2^7 = 128$

Expand the binary number as shown and add up the terms. The result will be in decimal form.

Answer: 164_{10}

Note that most digital systems deal with 4, 8, 16, or 32 bits at a time. Hence, in decimal-to-binary conversions, you may need to put an additional 0 in front of the MSB (e.g., 1101101 -> 01101101).

Octal Number System

Because binary numbers can be long and cumbersome to write and display, often the **Octal (Base 8) Number System** or **Hexadecimal (Base 16) Number System** is used as an alternative representation.

$$247_8 \text{ (octal)} = 2 \times 8^2 + 4 \times 8^1 + 7 \times 8^0 = 167_{10} \text{ (decimal)}$$

Octal to Binary

537_8 to binary

5 3 7

1 0 1 0 1 1 1 1 1

Answer: 101011111_2

Binary to Octal

$111\ 001\ 100_2$ to octal

1 1 1 0 0 1 1 0 0

7 1 4

Answer: 714_8

Decimal	Binary	Octal
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	10
9	1001	11
10	1010	12
11	1011	13
12	1100	14
13	1101	15
14	1110	16
15	1111	17

To convert a **binary number to octal**, divide the number into groups of **three digits** beginning with the LSB and replace each group with its octal equivalent.

$$123_{10} = 001\ 111\ 011_2 = 173_8$$

Hexadecimal Number System

In **Hexadecimal (Base 16) Number System**, the letters A through F are used to represent the digits larger than 9.

$$2D5_{16}\,(\text{hex}) = 2 \times 16^2 + D\,(=13_{10}) \times 16^1 + 9 \times 16^0 = 725_{10}$$

Hex to Binary

$3E9_{16}$ to binary

3 E 9

0 0 1 1 1 1 1 0 1 0 0 1

Answer: $0011\ 1110\ 1001_2$

Binary to Hex

$1001\ 1111\ 1010\ 0111_2$ to hex

1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 1

9 F A 7

Answer: $9FA7_{16}$

To convert a **binary number to hexadecimal**, divide the number into groups of **four digits** beginning with the LSB and replace each group with its hexadecimal equivalent.

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

$$123_{10} = 0111\ 1011_2 = 7B_{16}$$

Binary Coded Decimal (BCD)

Binary Coded Decimal (BCD) system is used to represent each digit of a decimal number as a 4-bit binary number.

```
   1     5     0
  /      |      \
0001   0101   0000
```

$150_{10} = 0001\ 0101\ 0000_{(BCD)}$

$123_{10} = 0001\ 0010\ 0011_{(BCD)}$

$123_{10} = 0111\ 1011_{2}$

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

- Although BCD is a **convenient** way to represent decimal numbers in a binary number format, it is **inefficient** for storing or transmitting multiple-digit numbers because only 10 of the 16 (2^4) possible states of the 4-bit number are used.

- BCD is commonly used when outputting to decimal (0–9) displays, such as those (seven-segments) found in digital clocks and multimeters.

ASCII Codes

American Standard Code for Information Interchange (**ASCII**) is an alphanumeric code used to transmit letters, symbols, numbers, and special nonprinting characters (e.g., ESC, DEL) represented in binary form between computers and computer peripherals (such as printers and keyboards).

- ASCII consists of 128 different 7-bit codes.
- The 7-bit codes are usually stored in an 8-bit byte.

$$\text{A: } 0100\ 0001 = 41_{16} = 65_{10}$$
$$\text{B: } 0100\ 0010 = 42_{16} = 66_{10}$$
$$\text{0: } 0011\ 0000 = 30_{16} = 48_{10}$$
$$\text{1: } 0011\ 0001 = 31_{16} = 49_{10}$$

CHAR	DEC	HEX	7-BIT
@	64	40	100 0000
A	65	41	100 0001
B	66	42	100 0010
C	67	43	100 0011
D	68	44	100 0100
E	69	45	100 0101
⋮			

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

Binary Arithmetic: Binary Addition

Adding binary numbers ($x_b + y_b$) is just like adding decimal numbers. Whenever the result of adding one column of numbers is greater than one digit, a 1 is carried over to the next column to be added. Here are the possibilities:

- 0 + 0 = 0
- 1 + 0 = 1
- 1 + 1 = 2 which is 10 in binary which is 0 with a carry of 1
- 1 + 1 + 1 (carry) = 3 which is 11 in binary which is 1 with a carry of 1

$$\begin{array}{r} \\ 9 \\ +\ 3 \\ \hline 12 \end{array} \qquad \begin{array}{r} \\ 1001 \\ +\ 0011 \\ \hline 1100 \end{array}$$

(11 carried)

$$+ \quad \begin{array}{rl} 5_{10} = & 0\ 1\ 0\ 1 \\ 3_{10} = & 0\ 0\ 1\ 1 \\ \hline & 1\ 0\ 0\ 0 \end{array}$$

(1 1 1 carried)

$$\begin{array}{r} 11010 \\ 10111 + \\ \hline 110001 \end{array}$$

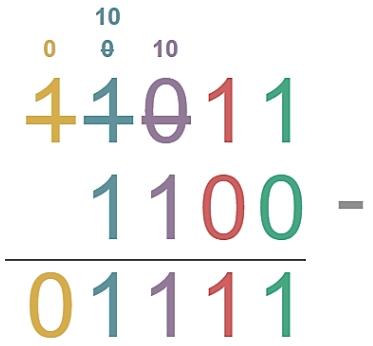
(1 1 1 1 carried)

Binary Arithmetic: Binary Subtraction (1st Method)

Similar to binary addition, for binary subtraction ($x_b - y_b$), we will work through the numbers, column by column, starting on the far right. Instead of carrying forward however, we will borrow backwards (when necessary). Here are the possibilities:

- 0 - 0 = 0
- 1 - 0 = 1
- 1 - 1 = 0
- 0 - 1 we can't do so we borrow 1 from the next column. This makes it 10 - 1 which is 1.

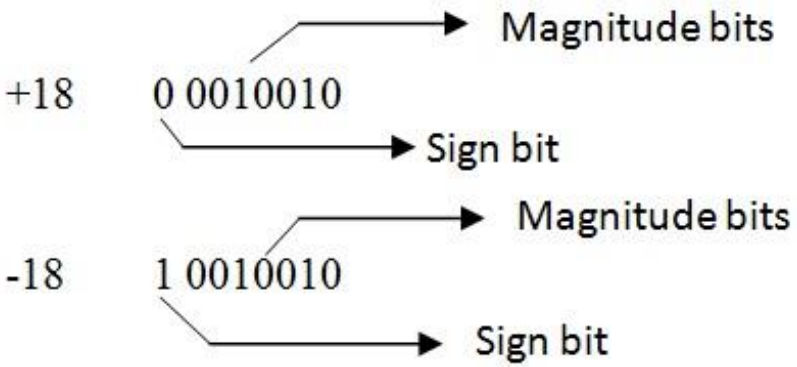
$$\begin{array}{r} 4_{10} \\ -\ 1_{10} \\ \hline 3_{10} \end{array} \qquad \begin{array}{r} \ \ 1 \\ 0\ \ \not{1}\ \ \not{0}\ \ \not{0} \\ -\ 0\ \ 0\ \ 0\ \ 1 \\ \hline 0\ \ 0\ \ 1\ \ 1 \end{array}$$



This method can be used only when $x_b \geq y_b$. Computers subtract binary digits using another approach that is to find the **negative value** of the second number and then add the numbers: $x_b + (-y_b)$.

Negative Binary Numbers: Sign-Magnitude Representation

One of the simplest methods is to represent negative binary numbers is **Sign-Magnitude Representation**. In this method, a bit, usually the MSB, to act as a **sign bit** is reserved. If the sign bit is **0**, the number is **positive**; if the sign bit is **1**, the number is **negative**.



Occasionally, you will see sign-magnitude numbers used in display and analog-to-digital applications, but you will hardly ever see them in circuits that perform arithmetic. Instead, **Two's Complement Representation** is usually used.

Negative Binary Numbers: Two's Complement Representation

In Two's Complement Representation,

- A **Positive** Number is the **same** as the actual binary number.
- A **Negative** Number is represented by a binary number, which when added to its corresponding positive equivalent results in zero.

Decimal to 2's Complement:

(Positive Number)

$+41_{10}$ to 2's Complement

- True Binary = 0010 1001
- 2's Complement = 0010 1001

(Negative Number)

-41_{10} to 2's Complement

- True Binary (41_{10}) = 0010 1001
- 1's Complement = 1101 0110 (1)
- Add 1: +1 (2)
- 2's Complement = 1101 0111

1) Take the 1's Complement: Complementing each bit of the true binary ($1 \rightarrow 0$, $0 \rightarrow 1$).
2) Add 1 to the 1's Complement number to get the magnitude bits. The **sign bit** will always end up being **1**.

Negative Binary Numbers: Two's Complement Representation

2's Complement to Decimal:

- If the 2's complement number is **positive** (sign bit = 0), perform a regular binary-to-decimal conversion.

- If the 2's complement number is **negative** (sign bit = 1), the decimal sign will be negative.

1) Complement each bit of the 2's complement number.
2) Add 1 to get the true binary equivalent.
3) Perform a true binary-to-decimal conversion and including negative sign.

2's complement =	1100 1101	
Complement:	0011 0010	(1)
Add 1:	+1	(2)
True Binary =	0011 0011	(3)
Decimal =	-51_{10}	

DECIMAL	2'S COMPLEMENT
+7	0000 0111
+6	0000 0110
+5	0000 0101
+4	0000 0100
+3	0000 0011
+2	0000 0010
+1	0000 0001
0	0000 0000
–1	1111 1111
–2	1111 1110
–3	1111 1101
–4	1111 1100
–5	1111 1011
–6	1111 1010
–7	1111 1001
–8	1111 1000

Binary Arithmetic: Binary Subtraction (2nd Method)

The second method to subtract binary numbers $(a - b)$ is to use the **two's complement representation**. That is, adding the numbers in this way: $a + (-b)$.

(i) Find the 2's complement representation of the number $(-b)$.
(ii) Add additional 0/1 to the left side of the positive/negative numbers if they don't have the same number of bits.
(iii) Add the numbers $a + (-b)$.
(iv) If the final carry over of the sum is 1 (overflow bit), it is dropped, and the result is positive.
(v) If there is no carry over, the result is negative.

69	01000101	
-35	11011101	+
	1 1 1 1 1 1 1	
34	100100010	
	Discard final bit	

55	00110111	
-95	10100001	+
	1 1 1 1	
-40	11011000	
	Finished	

Binary Arithmetic: Binary Multiplication

Binary multiplication is carried out in the same way as the decimal arithmetic. Here are the possibilities:

- $0 \times 0 = 0$
- $1 \times 0 = 0$
- $1 \times 1 = 1$

$$\begin{array}{r} 9 \\ \times\ 3 \\ \hline 27 \end{array} \qquad \begin{array}{r} 1001 \\ \times\ 0011 \\ \hline 1001 \\ +\ 1001 \\ +\ 0000 \\ +\ 0000 \\ \hline 11011 \end{array} \qquad \begin{array}{r} 11001 \\ 110\ \text{x} \\ \hline 00000 \\ 110010 \\ 1100100 \\ \hline 10010110 \end{array}$$

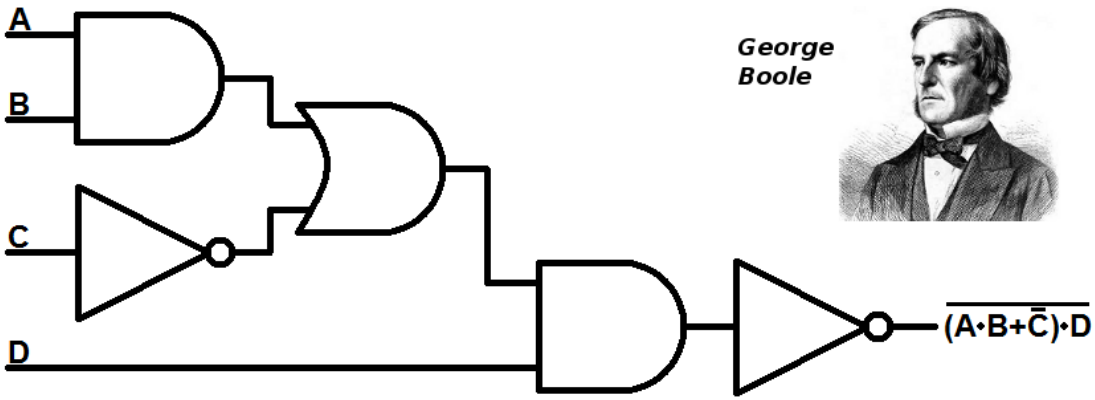
Binary division is also carried out in the same way as the decimal arithmetic.

Logic Gates

Logic Gates

Logic Gates are the building blocks of digital electronics. The fundamental logic gates include the NOT (INV), AND, NAND, OR, NOR, exclusive OR (XOR), and exclusive NOR (XNOR) gates.

The term **Combinational Logic** is used for the combining of two or more basic **Logic Gates** to form a required function. Outputs of **combinational logic** devices depend only on the **instantaneous values** of the inputs.



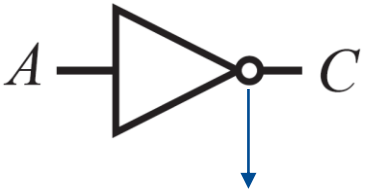
An online digital circuit simulator:
https://logic.ly/demo/

NOT (or INV) Gate

A NOT gate or INVerter outputs a logic level that is the opposite of the input logic level.
Operation: Inverting signal (complement)
Expression: $C = \bar{A}$
Truth Table: A table displaying all combinations of inputs and their corresponding outputs

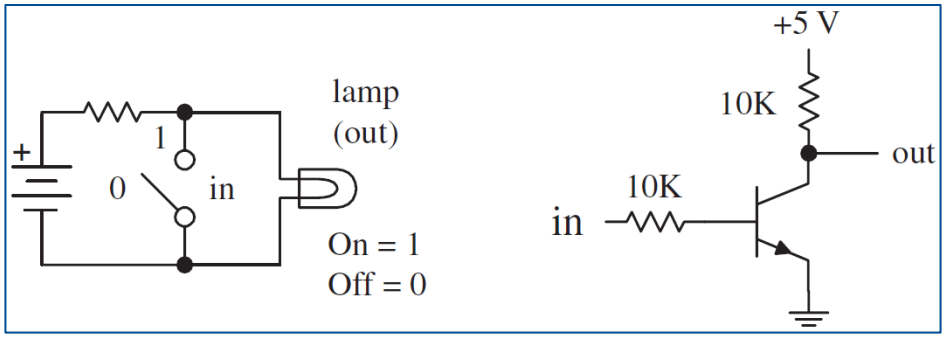
0 = LOW voltage level
1 = HIGH voltage level

A	C
0	1
1	0



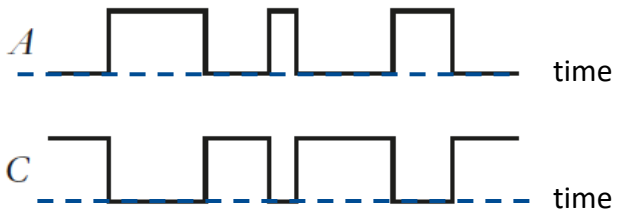
A small circle at the input or output denotes signal inversion (0→1, 1→0).

Switch Analogy



Transistor Analogy

- **Timing Diagram** helps to analyze complex logic circuits. It shows the simultaneous levels of every possible combination of inputs and corresponding outputs in a circuit vs. time.



AND Gate

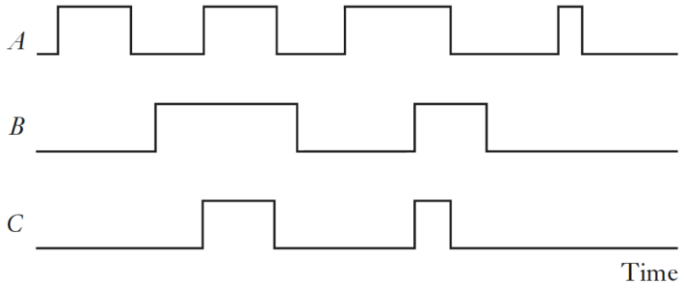
The output of an AND gate is HIGH only when both inputs are HIGH.

Operation: AND logic

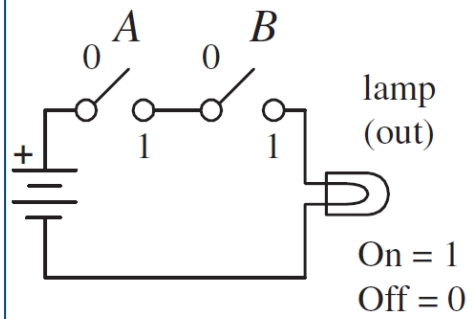
Expression: $C = A \cdot B$

Truth Table:

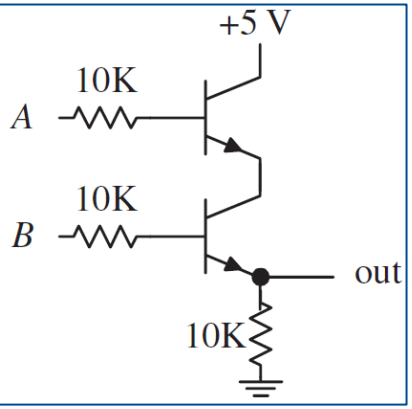
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



Switch Analogy



Transistor Analogy



NAND Gate

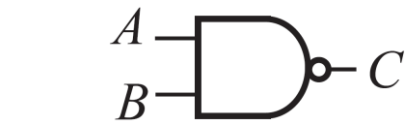
Combines the NOT function with an AND gate; output only goes LOW when both inputs are HIGH.

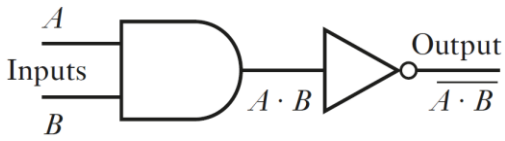
Operation: Inverted AND logic

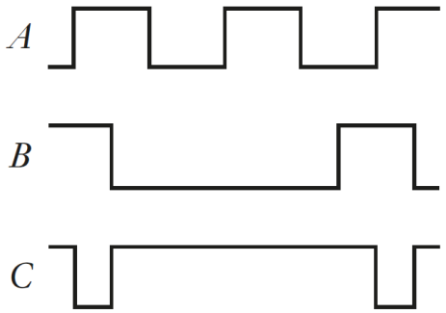
Expression: $C = \overline{A \cdot B}$

Truth Table:

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

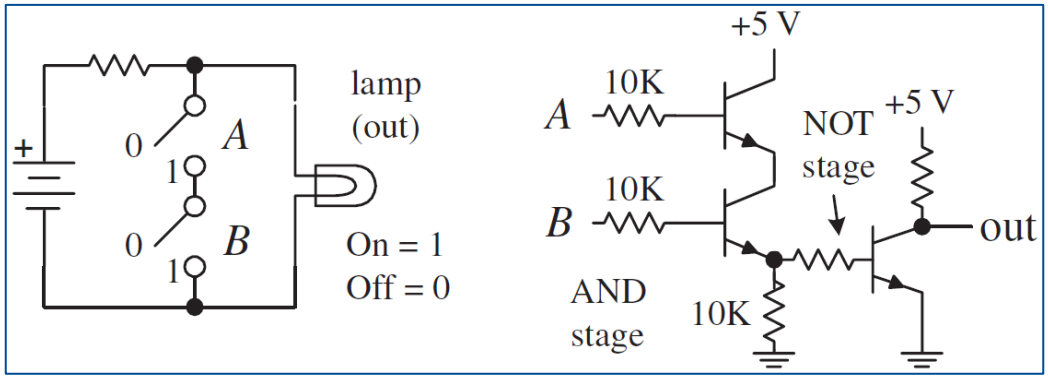






Switch Analogy

Transistor Analogy



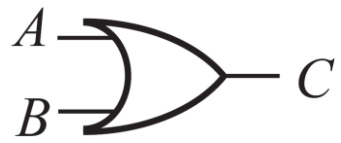
OR Gate

The output of an OR gate will go HIGH if one or both inputs goes HIGH. The output only goes LOW when both inputs are LOW.

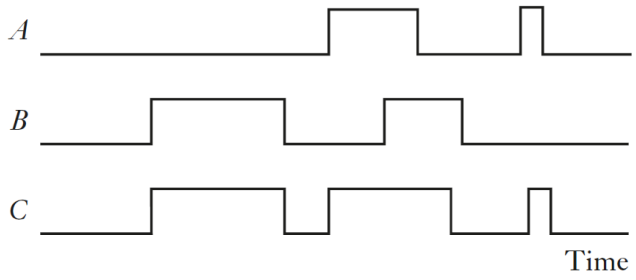
Operation: OR logic

Expression: $C = A + B$

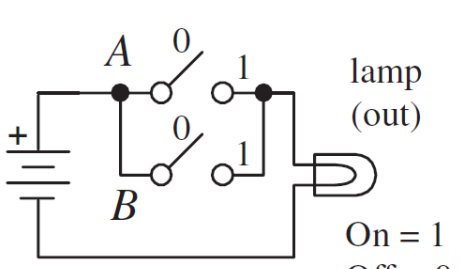
Truth Table:



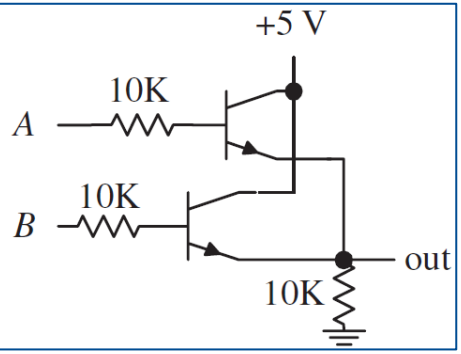
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1



Switch Analogy



Transistor Analogy



NOR Gate

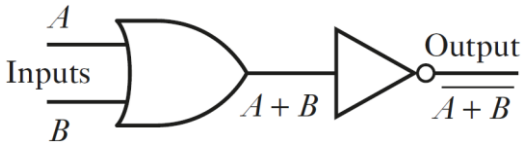
Combines the NOT function with an OR gate; output goes LOW if one or both inputs are LOW, output goes HIGH when both inputs are LOW.

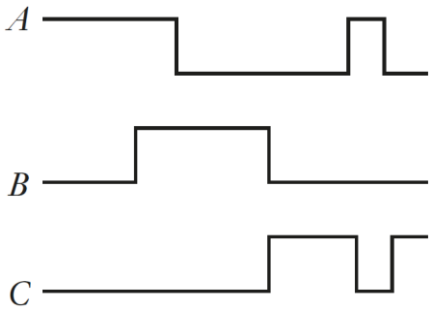
Operation: Inverted OR logic

Expression: $C = \overline{A + B}$

Truth Table:

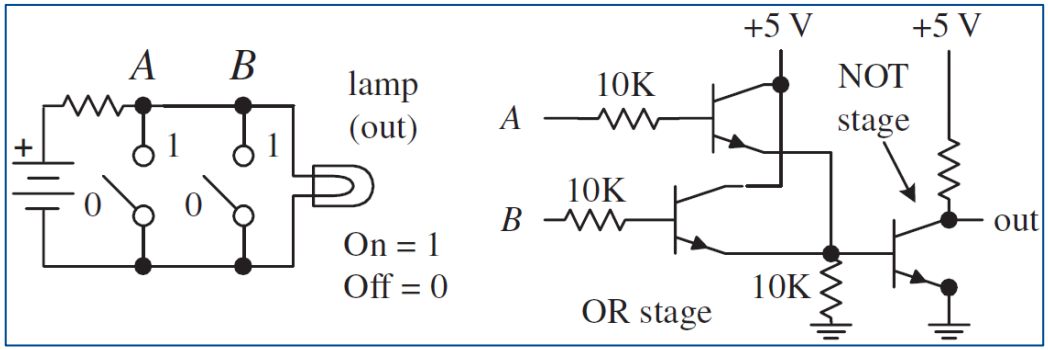
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0





Switch Analogy

Transistor Analogy



XOR (Exclusive OR) Gate

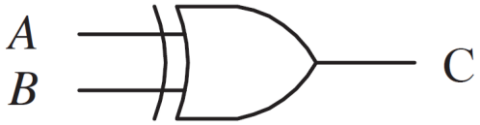
The output of an XOR gate goes HIGH if the inputs are different from each other.

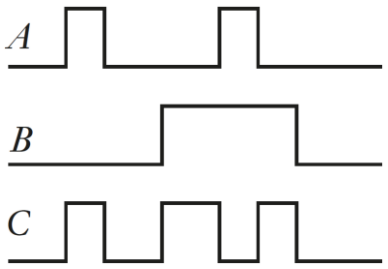
Operation: Exclusive OR logic

Expression: $C = A \oplus B = (A + B) \cdot (\bar{A} + \bar{B}) = A \cdot \bar{B} + \bar{A} \cdot B$

Truth Table:

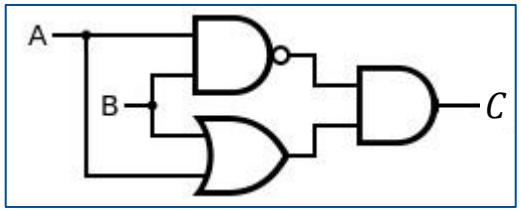
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0



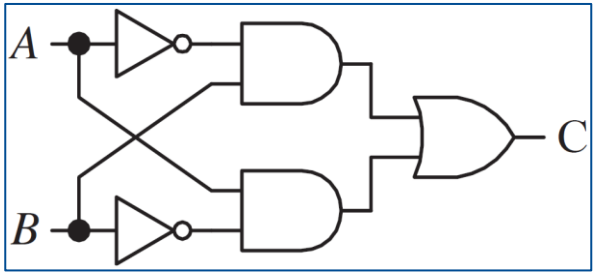


Equivalent Circuits

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



$A \cdot \bar{B} + \bar{A} \cdot B$



XNOR (Exclusive NOR) Gate

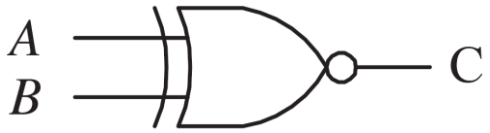
Combines the NOT function with an XOR gate; output goes HIGH if the inputs are the same.

Operation: Exclusive NOR logic

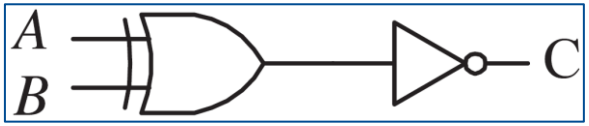
Expression: $C = \overline{A \oplus B} = A \odot B = \overline{(A + B) \cdot (\bar{A} + \bar{B})} = A \cdot B + \bar{A} \cdot \bar{B}$

Truth Table:

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

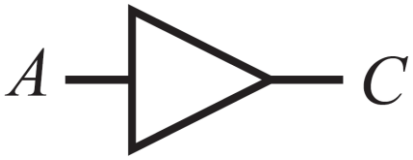


Equivalent Circuit



Buffer

- A **Buffer** is used to **increase the current** supplied at the output while **retaining** the digital state.
- This is important if you wish **to drive multiple digital inputs from a single output**.
- A normal digital gate has a limited **fan-out**, **which defines the maximum number of similar digital inputs that can be driven by the gate's output**. The buffer overcomes fan-out limitations by providing more output current.

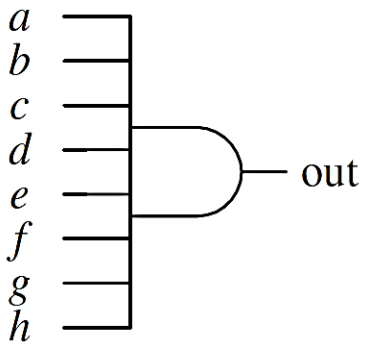


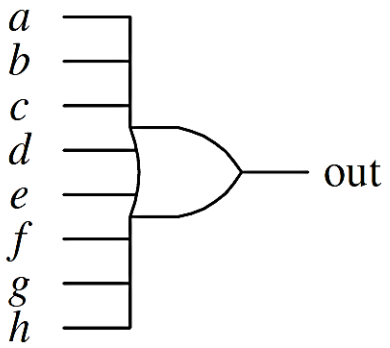
A	C
0	0
1	1

Multiple-Input Logic Gates

The **standard** AND, NAND, OR, NOR, and XOR gates have only two inputs, but other forms are available with more than two inputs.

- In the case of AND gate, the output is 1 if and only if all inputs are 1; otherwise, the output is 0.
- In the case of OR gate, the output is 0 if and only if all inputs are 0; otherwise, it is 1.
- In the case of XOR gate, the output is 0 if all of the inputs are 0 or if all of the inputs are 1; otherwise, it is 1.

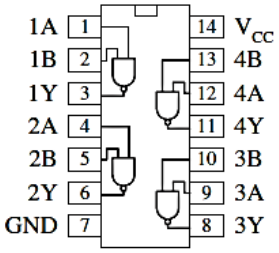




Digital Logic Gate ICs
(Examples of TTL & CMOS 74XX Series)

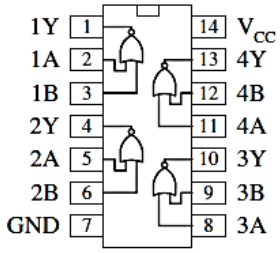
The logic gates are manufactured as **Integrated Circuits**. The different manufacturers have standardized their numbering schemes so that the part numbers are the same regardless of the manufacturer.

Quad 2-input NAND



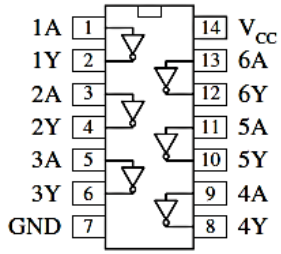
7400, 74LS00,
74F00, 74HC00, etc.

Quad 2-input NOR



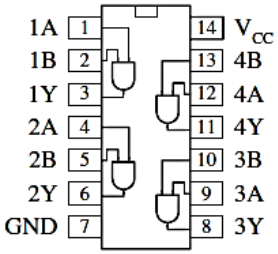
7402, 74LS02,
74F02, 74HC02, etc.

Hex Inverter



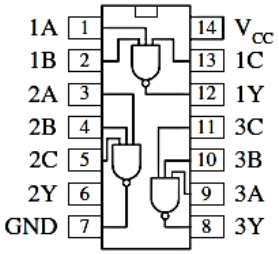
7404, 74LS04,
74F04, 74HC04, etc.

Quad 2-input AND



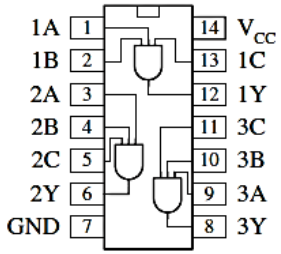
7408, 74LS08,
74F08, 74HC08, etc.

Triple 3-input NAND



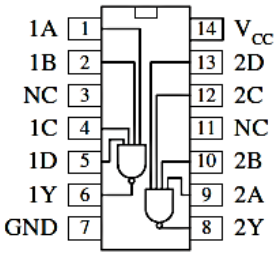
7410, 74LS10,
74F10, 74HC10, etc.

Triple 3-input AND



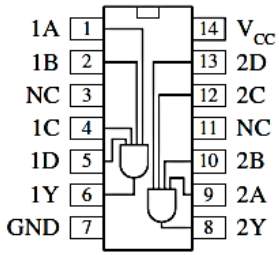
7411, 74LS11,
74F11, 74HC11, etc.

Dual 4-input NAND



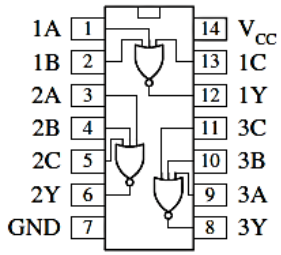
7420, 74LS20,
74F20, 74HC20, etc.

Dual 4-input AND



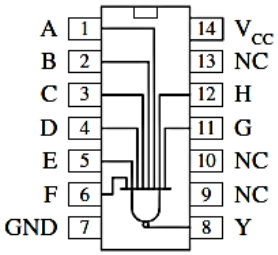
7421, 74LS21,
74F21, 74HC21, etc.

Triple 3-input NOR



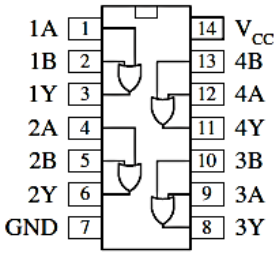
7427, 74LS27,
74F27, 74HC27, etc.

8-input NAND



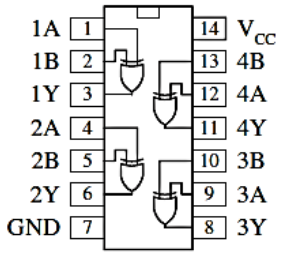
7430, 74LS30,
74F30, 74HC30, etc.

Quad 2-input OR



7432, 74LS32,
74F32, 74HC32, etc.

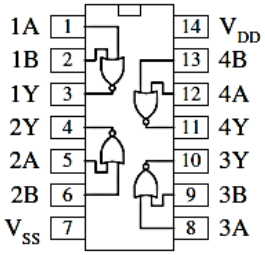
Quad 2-input XOR



7486, 74LS86,
74F86, 74HC86, etc.

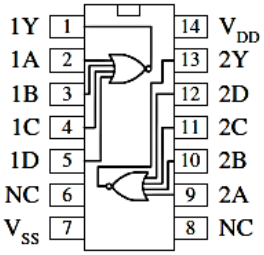
Digital Logic Gate ICs
(Examples of CMOS 40XXB Series)

Quad 2-input NOR



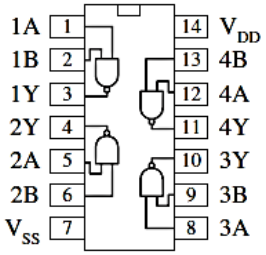
CMOS 4001(B)

Dual 4-input NOR



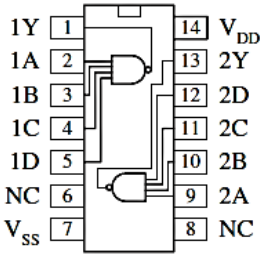
CMOS 4002(B)

Quad 2-input NAND



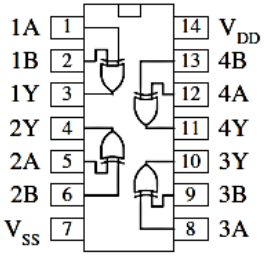
CMOS 4011(B)

Dual 4-input NAND



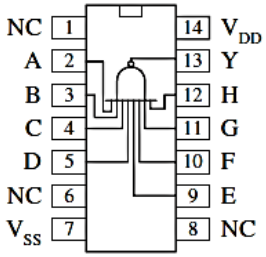
CMOS 4012(B)

Quad 2-input XOR



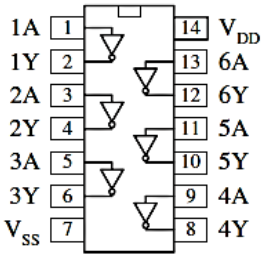
CMOS 4030(B)

8-input NAND



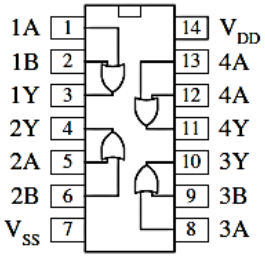
CMOS 4068(B)

Hex inverter



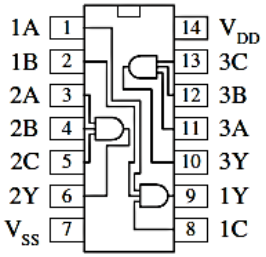
CMOS 4069(B)

Quad 2-input OR



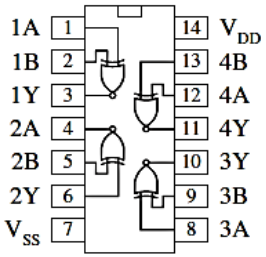
CMOS 4071(B)

Triple 3-input AND



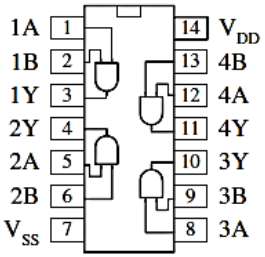
CMOS 4073(B)

Quad 2-input XNOR



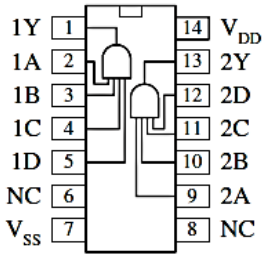
CMOS 4077(B)

Quad 2-input AND



CMOS 4081(B)

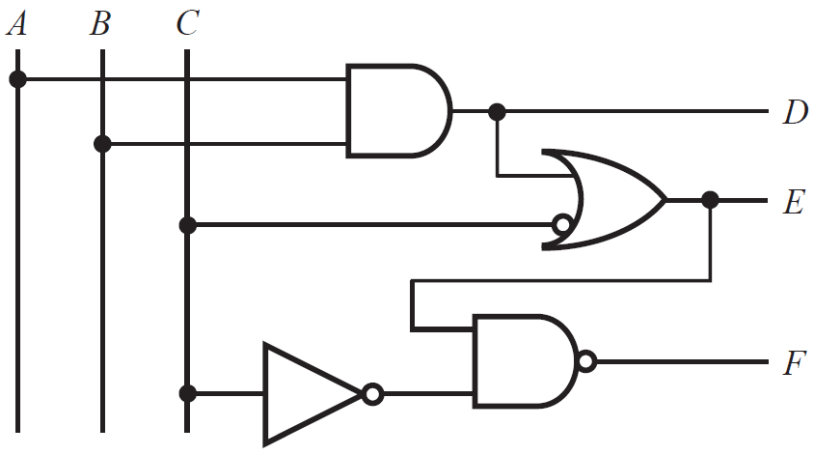
Dual 4-input AND



CMOS 4082(B)

Example

Determine signal expressions and values in the following logic diagram.



Boolean Algebra

Boolean Algebra

Boolean Algebra is important in formulating mathematical expressions for logic circuits.

(+) $\Rightarrow$ OR , (·) $\Rightarrow$ AND , ($\bar{X}$) $\Rightarrow$ NOT

Fundamental Laws

OR	AND	NOT
$A + 0 = A$	$A \cdot 0 = 0$	
$A + 1 = 1$	$A \cdot 1 = A$	$\bar{\bar{A}} = A$ (double inversion)
$A + A = A$	$A \cdot A = A$	
$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$	

Commutative Laws

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

Associative Laws

$$(A + B) + C = A + (B + C)$$

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

Distributive Laws

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Boolean Algebra

De Morgan's Laws:

$$\overline{A + B + C + \cdots} = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \cdots$$

$$\overline{A \cdot B \cdot C \cdots} = \bar{A} + \bar{B} + \bar{C} + \cdots$$

or ⇓

$$A + B + C + \cdots = \overline{\bar{A} \cdot \bar{B} \cdot \bar{C} \cdots}$$

$$A \cdot B \cdot C \cdots = \overline{\bar{A} + \bar{B} + \bar{C} + \cdots}$$

Other Useful Identities:

$$A + (A \cdot B) = A$$

$$A \cdot (A + B) = A$$

$$A + (\bar{A} \cdot B) = A + B$$

$$(A + B) \cdot (A + \bar{B}) = A$$

$$A + B + (A \cdot \bar{B}) = A + B$$

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

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

Boolean Algebra

Truth tables can be very helpful in verifying an identity.

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

A	$\overline{A}$	B	$\overline{A} \cdot B$	$A + \overline{A} \cdot B$	$A + B$
1	0	0	0	1	1
1	0	1	0	1	1
0	1	0	0	0	0
0	1	1	1	1	1

Example

Simplify the following expression using Boolean laws and identities.

$$X = (A \cdot B \cdot C) + (B \cdot C) + (\overline{A} \cdot B)$$

Logic Networks Design: A Practical Example

An application of combinational logic to a real engineering problem:
Design a circuit for a simple security system for a home in a way that an alarm to sound if someone breaks into the house through a door/window or if something is moving around the house. The user may also want to disable portions of the alarm system.

I. Defining Boolean variables for inputs and outputs of the circuit:

A: state of the door/window sensors
B: state of the motion detector
$C\ D$: 2-bit code to select the operating state by user defined by

- **0 1** the alarm will sound only if the doors/windows are disturbed (e.g., when occupants are sleeping).
- **1 0** the alarm will sound if the doors/windows are disturbed or if there is motion in the house (e.g., when occupants are away).
- **0 0** the alarm will not sound (normal state).

Y: output used to sound the alarm

(Inputs: A, B, C, and D; Output: Y)

Logic Networks Design: A Practical Example

II. Quasi-Logic Statement:

Activate the alarm ($Y = 1$) if A is high and the code $C\ D$ is 0 1 or activate the alarm if A or B is high and the code is 1 0.

III. Boolean Expression:

$$Y = A \cdot (\bar{C} \cdot D) + (A + B) \cdot (C \cdot \bar{D})$$

Note: To create a product of 1 for the active control code 0 1, we need to form the expression $\bar{C} \cdot D$.

For this particular problem, by looking at a truth table, the equation can be **simplified**:

C	D	$(\bar{C} \cdot D)$	$(C \cdot \bar{D})$
0	0	0	0
1	0	0	1
0	1	1	0

$\Rightarrow \bar{C} \cdot D = D\ , C \cdot \bar{D} = C$

$$\Rightarrow Y = (A \cdot D) + (A + B) \cdot C$$

Logic Networks Design: A Practical Example

IV. Realization:

After simplification, it may be desirable to manipulate the result further in order to convert all operations to a preferred type of gate (e.g., NOT, AND, NAND, OR, or NOR) to minimize the number of required logic gate IC components.

Converting from one gate type to another is easily accomplished with a repeated application of **De Morgan's Laws**.

$$Y = (A \cdot D) + (A + B) \cdot C \xrightarrow{A + B + C + \cdots = \overline{\bar{A} \cdot \bar{B} \cdot \bar{C} \cdots}} Y = (A \cdot D) + (\overline{\bar{A} \cdot \bar{B}}) \cdot C$$

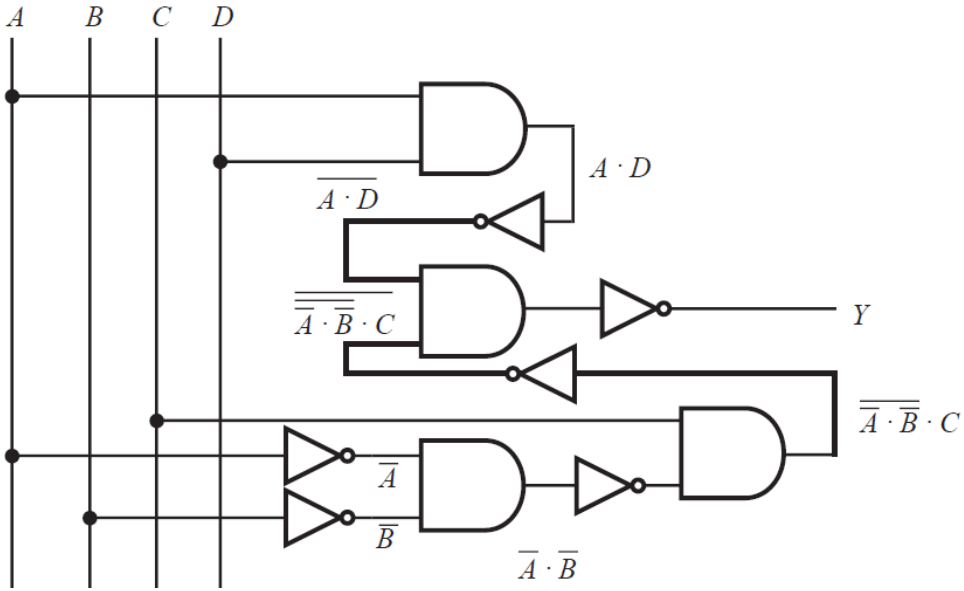
$$Y = \overline{\overline{A \cdot D} \cdot \overline{(\bar{A} \cdot \bar{B}) \cdot C}}$$

Logic Networks Design: A Practical Example

V. Drawing the Circuit Diagram:

Since there are a total of 4 AND gates and 6 inverters, the circuit can be made with 2 ICs:

- one quad AND gate IC (e.g., the 7408), which contains four AND gates,
- one hex inverter IC (e.g., the 7404), which contains six inverters.



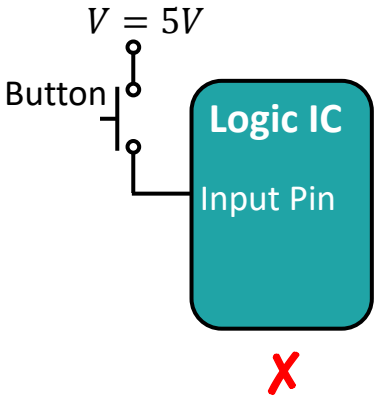
$$Y = \overline{\overline{A \cdot D} \cdot \overline{\overline{\overline{A} \cdot \overline{B}} \cdot C}}$$

❖ This solution is known as a **hardware solution** because it uses ICs to provide the desired logic. A **software solution** is to implement the logic using a program running on a **microcontroller**.

Application

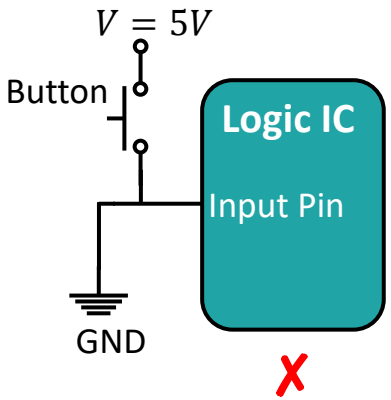
Pullup and Pulldown Resistors

To connect buttons to logic ICs, **Pullup** resistors are used to keep an input normally high and **Pulldown** resistors are used to keep an input normally low; otherwise, **Floating** or **Short Circuit** happens.



When the button is not pressed, the input pin is connected to nothing (the pin is said to be **Floating**). Thus, due to electrical noise on nearby pins, value of input pin fluctuates between high and low.

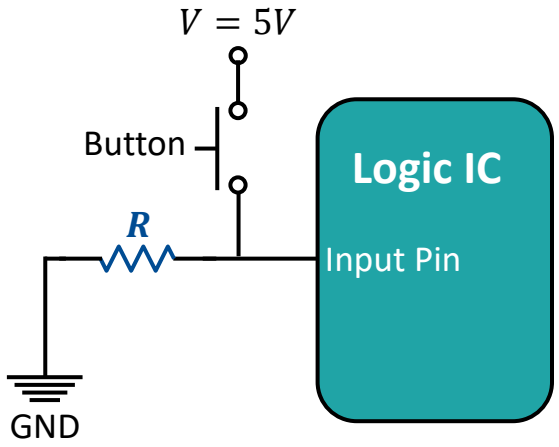
→ Unexpected results is read on the pin!



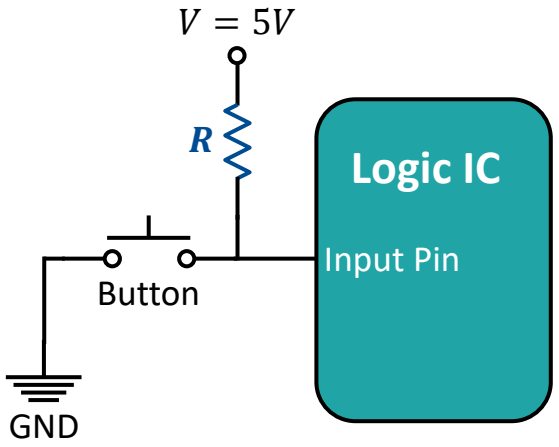
When the button is pressed, **Short Circuit** happens!

Pullup and Pulldown Resistors

Nearly all digital inputs use a **Pullup** or **Pulldown** **Resistor** to set the **default state** of the input pin.



Pulldown Resistor is used to make the initial state of input pin **LOW**

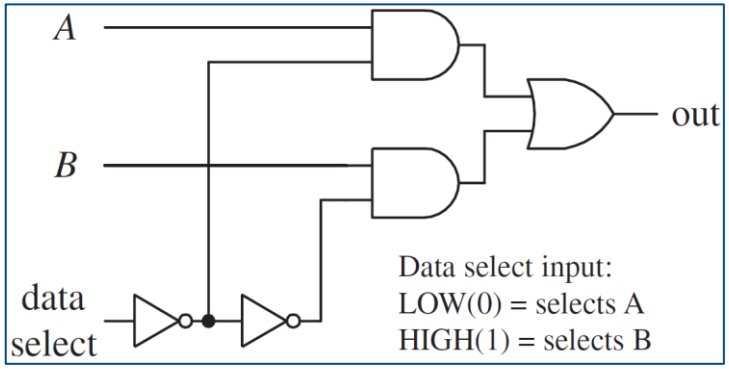


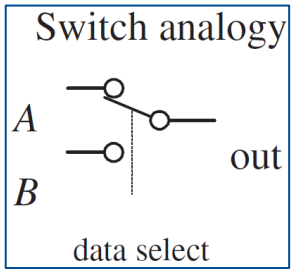
Pullup Resistor is used to make the initial state of input pin **HIGH**

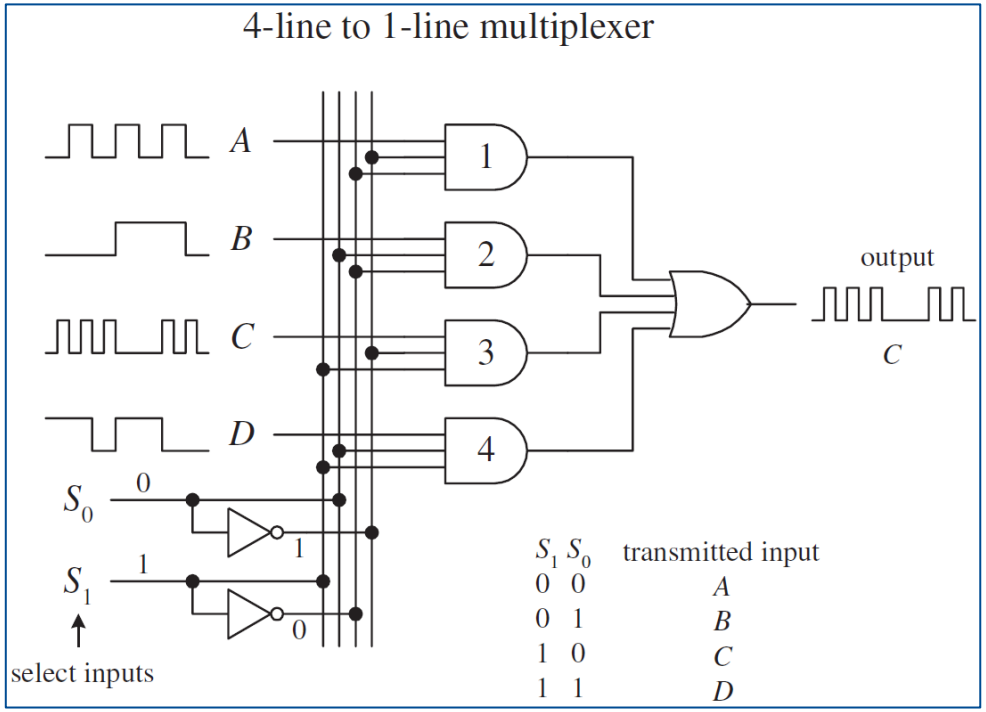
Typically, a pulldown resistor is around 100 to 1k and a pullup resistor is around 10k.

Multiplexers (Data Selectors)

Multiplexers (**Data Selectors**) act as digitally controlled switches similar to SPDT, SP4T, SP8T etc. switches.

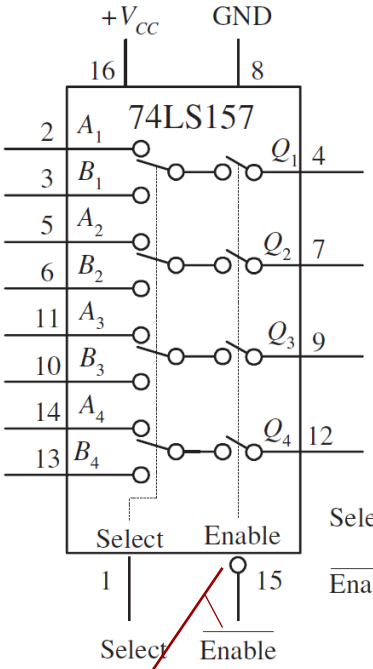


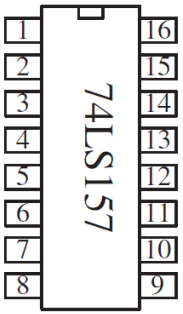




Multiplexers (Data Selectors)

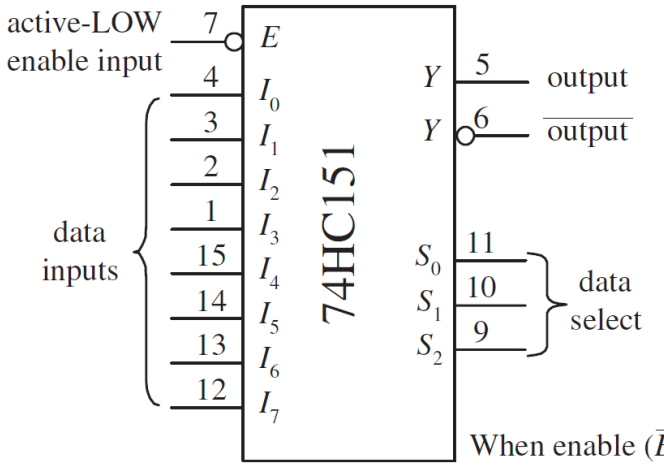
74LS157 quad 1-of-2 data selector





Select: HIGH (1) = A inputs selected
LOW (0) = B inputs selected
$\overline{\text{Enable}}$: HIGH (1) = outputs disabled
LOW (0) = outpus enabled

8-line to 1-line multiplexer



S_1	S_2	S_3	data out (Y)
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

V_{CC} = pin 16
GND = pin 8

When enable ($\overline{E}$) is set HIGH (1), all inputs are disabled—output (Y) is forced LOW (0) regardless of all other inputs.

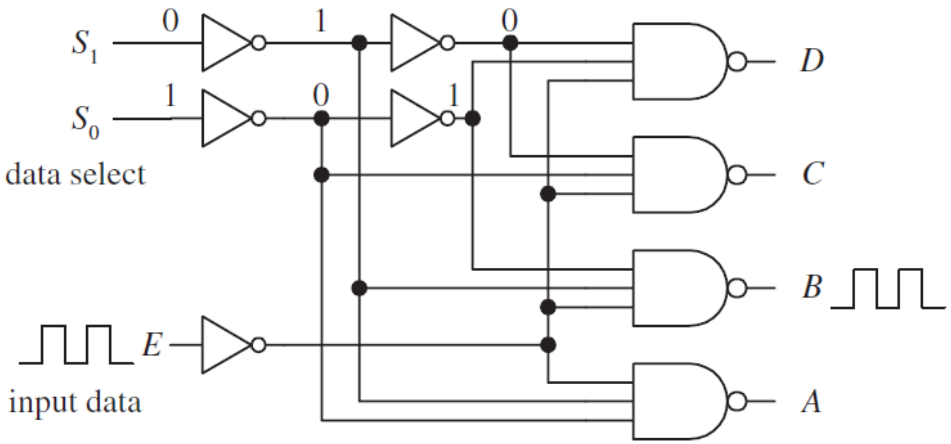
This type of enable control is referred to as **active-low Enable**.

These two conditions depends on the state of the **Enable** input.

Demultiplexers (Data Distributors)

Demultiplexer (or **Data Distributor**) is the opposite of Multiplexer. It takes a single data input and routes it to one of several possible outputs.

4-line demultiplexer logic circuit



Control logic

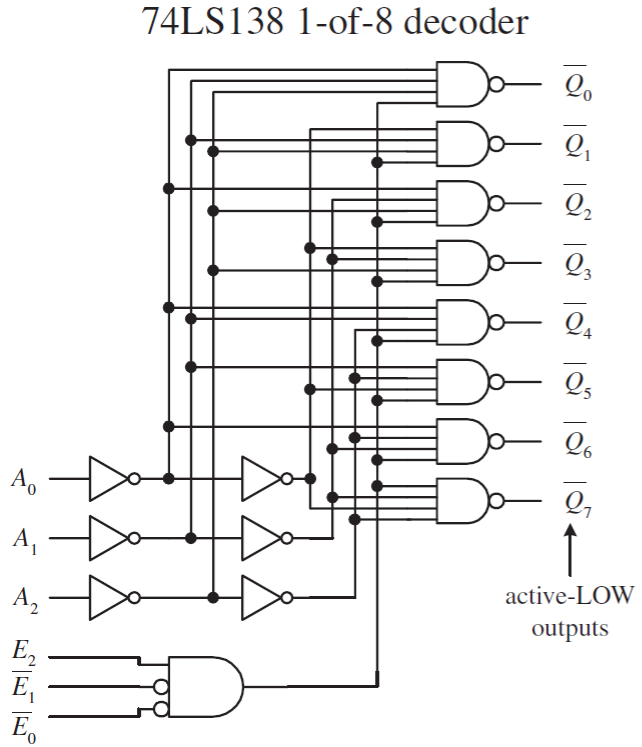
S_1	S_0	input routed to:
0	0	A
0	1	B
1	0	C
1	1	D

disabled outputs are held HIGH

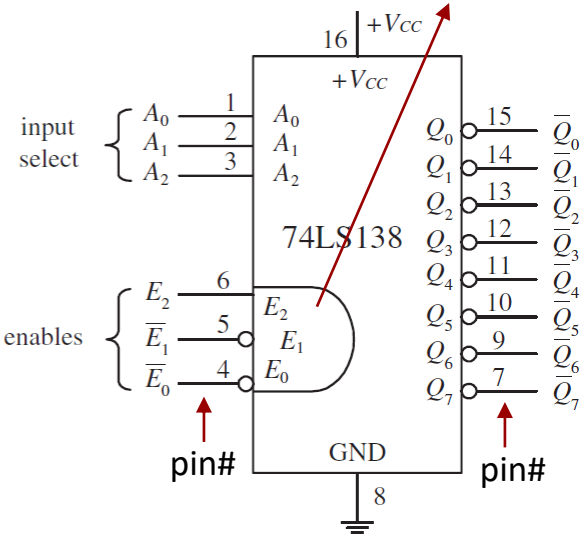
Decoders

Decoder simply uses the data select inputs to choose which outputs, among many, are to be made high or low.

- 74138 uses 3 inputs to select which of eight outputs will be made low; all other outputs are held high (**active-low output**).

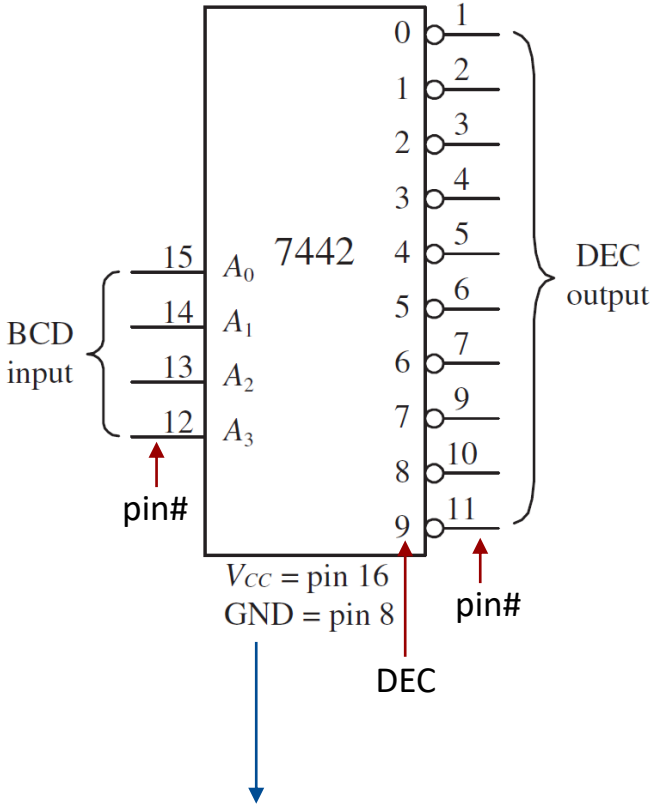


Make the active-low inputs $\overline{E_0}$ and $\overline{E_1}$ **low** and the active-high input E_2 **high**; otherwise, the decoder is disabled, making all active-low outputs high regardless of the selected inputs.

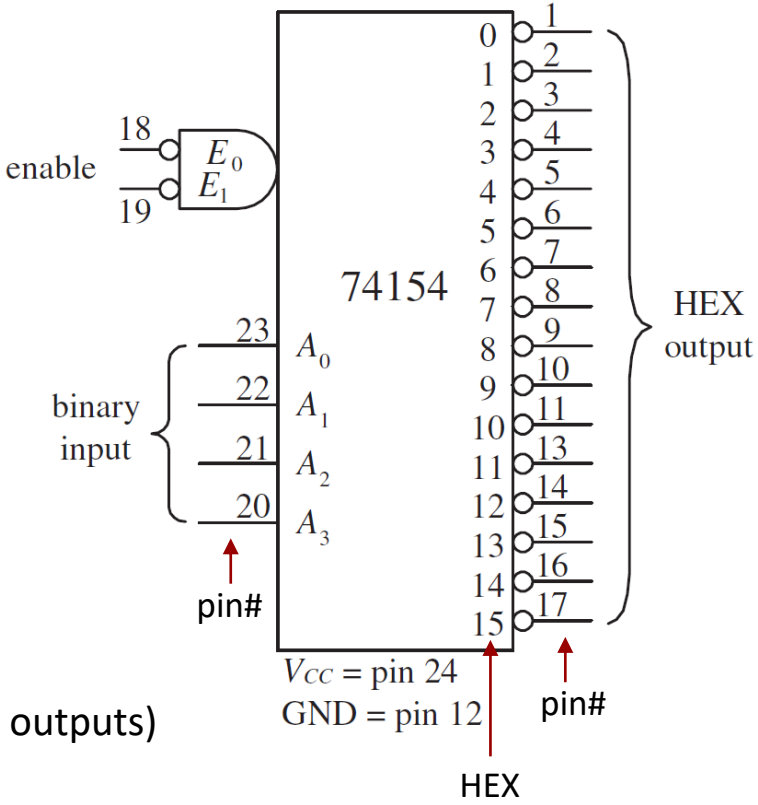


Decoders

7442 BCD-to-DEC Decoder



74154 Binary-to-HEX Decoder

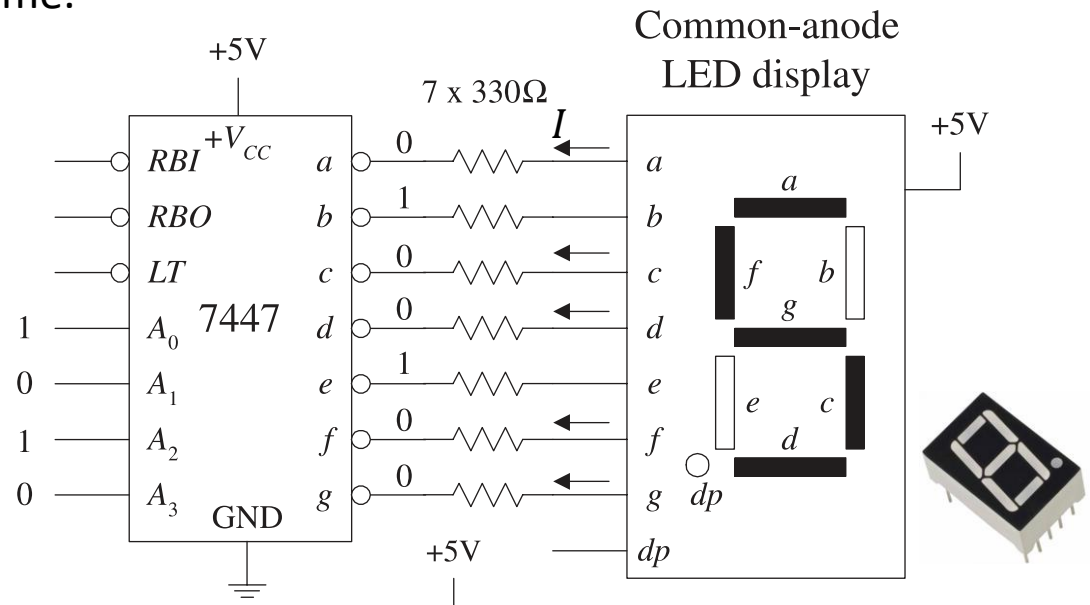


(active-low outputs)

- Four inputs to select which of ten outputs will be made low; all other outputs are held high (**active-low output**).

Decoders

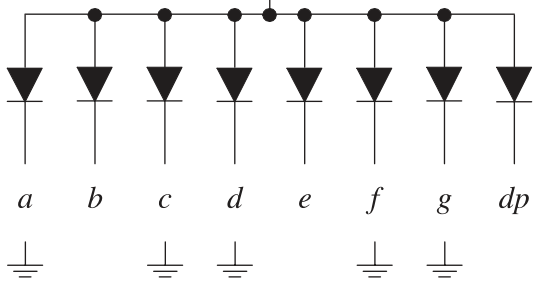
With 7447 **BCD-to-Seven-Segment decoder**, more than one output can be driven low at a time.

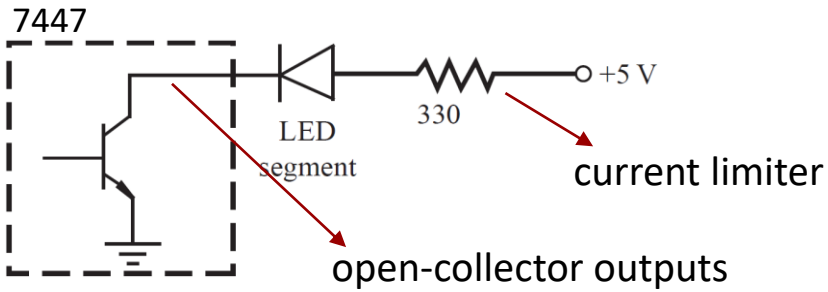


Inputs $A_3 A_2 A_1 A_0$	Digit	7447 outputs $\bar{a}$	$\bar{b}$	$\bar{c}$	$\bar{d}$	$\bar{e}$	$\bar{f}$	$\bar{g}$
0 0 0 0	0	0	0	0	0	0	0	1
0 0 0 1	1	1	0	0	1	1	1	1
0 0 1 0	2	0	0	1	0	0	1	0
0 0 1 1	3	0	0	0	0	1	1	0
0 1 0 0	4	1	0	0	1	1	0	0
0 1 0 1	5	0	1	0	0	1	0	0
0 1 1 0	6	0	1	0	0	0	0	0
0 1 1 1	7	0	0	0	1	1	1	1
1 0 0 0	8	0	0	0	0	0	0	0
1 0 0 1	9	0	0	0	1	1	0	0

1 = HIGH voltage level,
0 = LOW voltage level

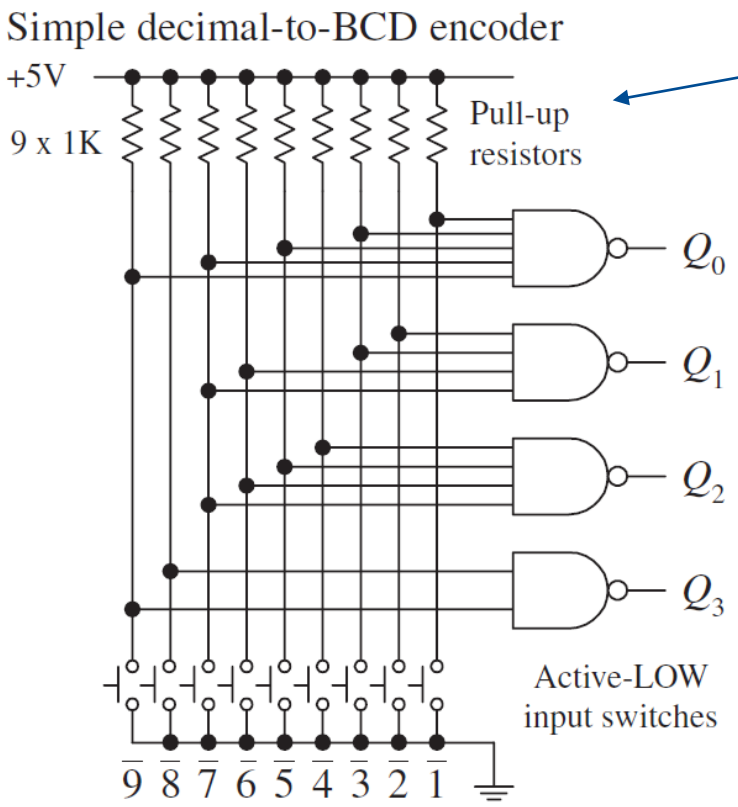
Internal wiring of display





Encoders

Encoders are the opposite of decoders. They are used to generate a **coded output** from a single active numeric input.



Normally all lines are held high by the pullup resistors connected to +5 V.

Truth Table

$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{8}$	$\overline{9}$	Q_3	Q_2	Q_1	Q_0	BCD
H	H	H	H	H	H	H	H	H	L	L	L	L	0000 (0_{10})
L	H	H	H	H	H	H	H	H	L	L	L	H	0001 (1_{10})
H	L	H	H	H	H	H	H	H	L	L	H	L	0010 (2_{10})
H	H	L	H	H	H	H	H	H	L	L	H	H	0011 (3_{10})
H	H	H	L	H	H	H	H	H	L	H	L	L	0100 (4_{10})
H	H	H	H	L	H	H	H	H	L	H	L	H	0101 (5_{10})
H	H	H	H	H	L	H	H	H	L	H	H	L	0110 (6_{10})
H	H	H	H	H	H	L	H	H	L	H	H	H	0111 (7_{10})
H	H	H	H	H	H	H	L	H	H	L	L	L	1000 (8_{10})
H	H	H	H	H	H	H	H	L	H	L	L	H	1001 (9_{10})

H = High voltage level, L = Low voltage level

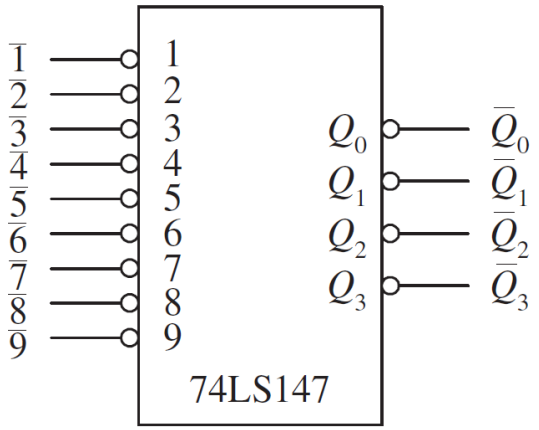
Encoders

74LS147 decimal-to-BCD **priority encoder**:

The term **priority** means that it is designed so that if two or more inputs are selected at the same time, it will select only the larger-order digit (e.g., if 3, 5, and 8 are selected at the same time, only the 8 will be output).

$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{8}$	$\overline{9}$	$\overline{Q}_3$	$\overline{Q}_2$	$\overline{Q}_1$	$\overline{Q}_0$	BCD (neg. logic)
H	H	H	H	H	H	H	H	H	H	H	H	H	1111 (0_{10})
X	X	X	X	X	X	X	X	L	L	H	H	L	0110 (9_{10})
X	X	X	X	X	X	X	L	H	L	H	H	H	0111 (8_{10})
X	X	X	X	X	X	L	H	H	H	L	L	L	1000 (7_{10})
X	X	X	X	X	L	H	H	H	H	L	L	H	1001 (6_{10})
X	X	X	X	L	H	H	H	H	H	L	H	L	1010 (5_{10})
X	X	X	L	H	H	H	H	H	H	L	H	H	1011 (4_{10})
X	X	L	H	H	H	H	H	H	H	H	L	L	1100 (3_{10})
X	L	H	H	H	H	H	H	H	H	H	L	H	1101 (2_{10})
L	H	H	H	H	H	H	H	H	H	H	H	L	1110 (1_{10})

H = High voltage level, L = Low voltage level, X = don't care

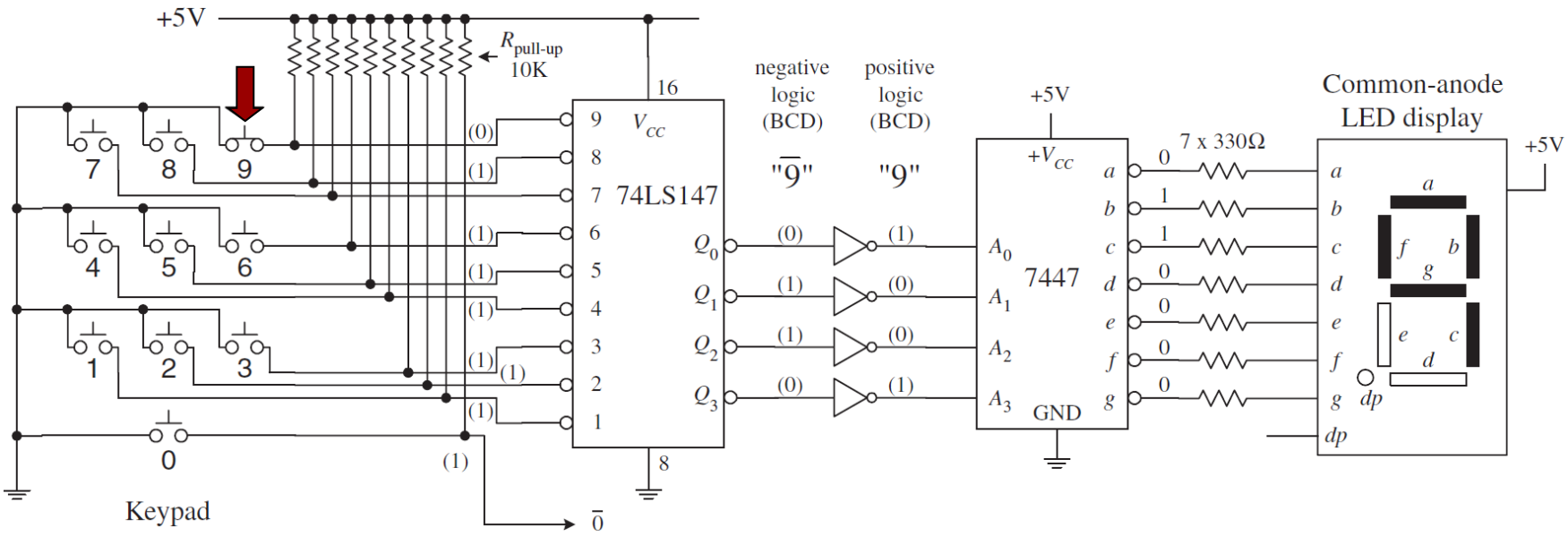


- It has **active-low outputs**, i.e., output is expressed in **negative true logic**.

Other encoders:
- 74148 octal-to-binary encoder
- 74184 BCD-to-binary encoder
- 74185 binary-to-BCD encoder
- …

Encoders

Using an encoder and a decoder together to drive an LED display via a 0 through 9 keypad:



- **Note**: For all logic ICs, by placing a load (e.g., LED) between +VCC and an output LOW, you can sink current through the load and into the output. By placing a load between an output HIGH and ground, you can source current from the output and sink it through the load. The limits to how much current an IC can source, or sink should be considered.