Chapter 7

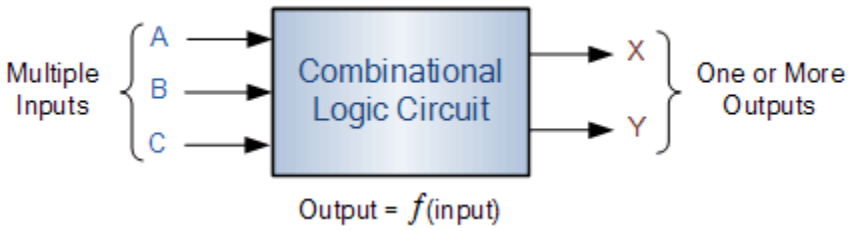
Combination of Logical Circuit: Data Control Circuit

The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependent at all times on the combination of its inputs. Thus a combinational circuit is *memoryless.*

So if one of its inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have "no memory", "timing" or "feedback loops" within their design.

Combinational Logic



Combinational Logic Circuits are made up from basic logic AND, OR, NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of combinational logic circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.

Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.

The three main ways of specifying the function of a combinational logic circuit are:

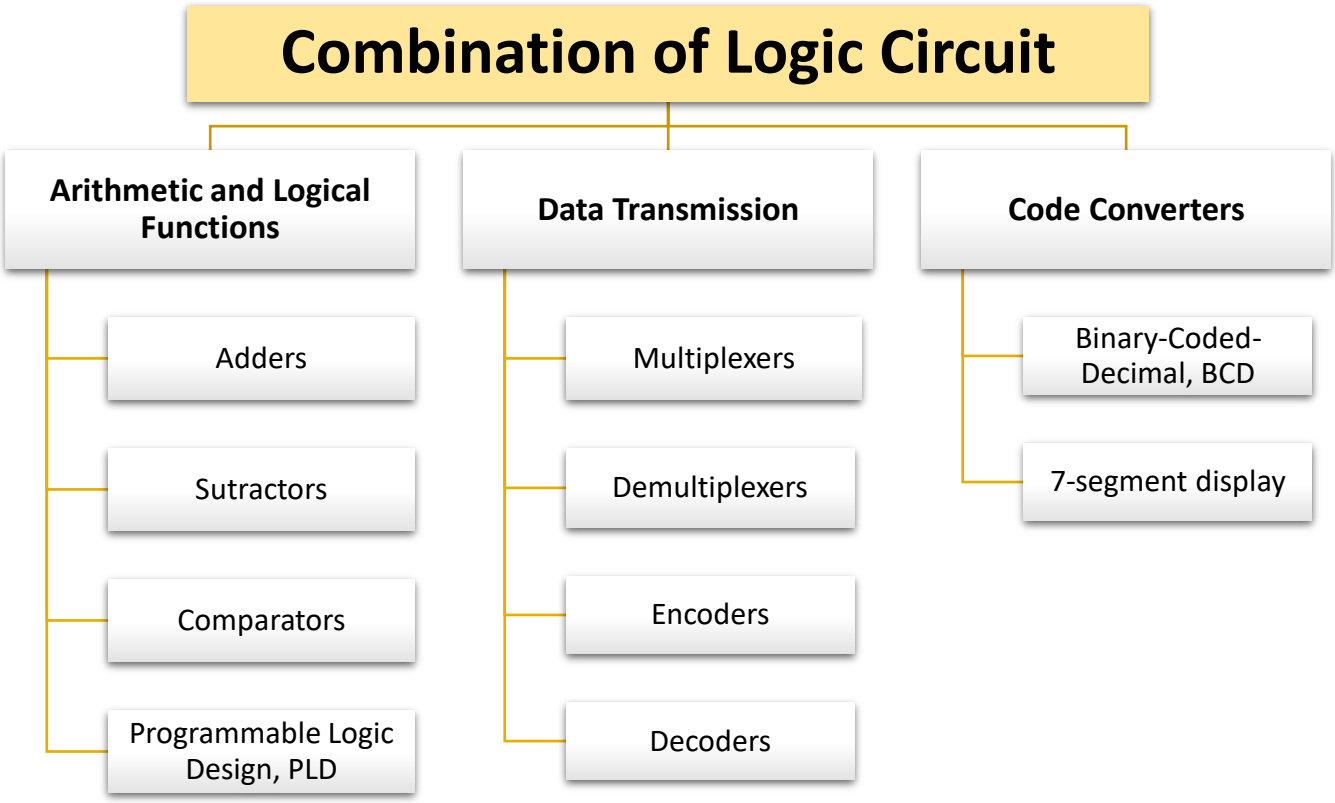
1. Boolean Algebra – This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic "1" output.

2. Truth Table – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.

3. Logic Diagram – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.

As combinational logic circuits are made up from individual logic gates only, they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application include *Multiplexers*, *De-multiplexers*, *Encoders*, *Decoders*, *Full* and *Half Adders* etc.

Classification of Combinational Logic Circuit



One of the most common uses of combinational logic is in Multiplexer and De-multiplexer type circuits. Here, multiple inputs or outputs are connected to a common signal line and logic gates are used to decode an address to select a single data input or output switch.

A multiplexer consist of two separate components, a logic decoder and some solid state switches, but before we can discuss multiplexers, decoders and de-multiplexers in more detail we first need to understand how these devices use these "solid state switches" in their design.

(Electronics Tutotials, 2019)

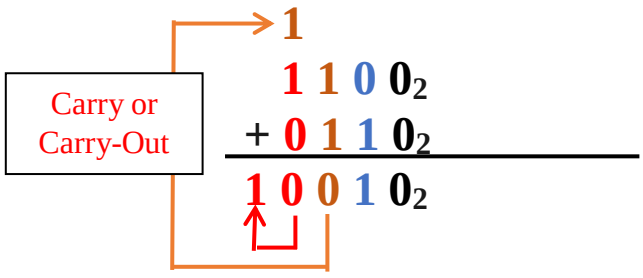
Binary Adder

The addition of two binary numbers is performed as addition in decimal numbers:

$$\begin{array}{r} 0_2 \\ +\ 0_2 \\ \hline 0_2 \end{array} \qquad \begin{array}{r} 0_2 \\ +\ 1_2 \\ \hline 1_2 \end{array} \qquad \begin{array}{r} 1_2 \\ +\ 0_2 \\ \hline 1_2 \end{array} \qquad \begin{array}{r} 1_2 \\ +\ 1_2 \\ \hline 1\ 0_2 \end{array}$$

Carry or Carry-Out to the next position in the left

Example 7.1) Perform the following binary addition



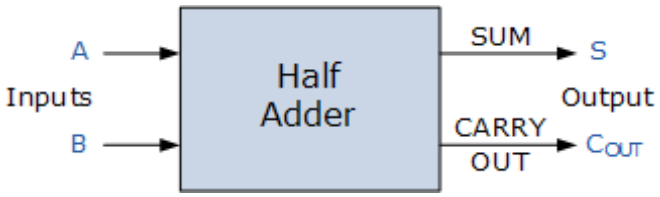
In digital circuit, combinational logic circuit can be constructed using just a few basic logic gates allowing it to add together two or more binary numbers is the **Binary Adder**.

Binary adders are arithmetic circuits in the form of half-adders and full-addersb used to add together two binary digits.

A basic Binary Adder circuit can be made from standard AND and Ex-OR gates allowing us to "add" together two single bit binary numbers, A and B.

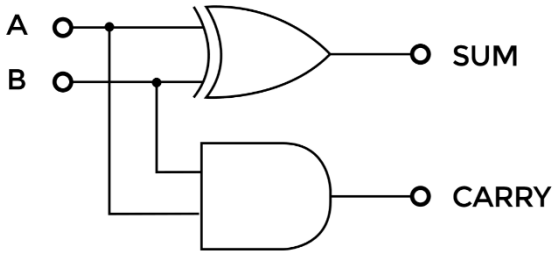
The addition of these two digits produces an output called the SUM of the addition and a second output called the CARRY or Carry-out, (C_{OUT}) bit according to the rules for binary addition.

Binary Adder Half Adder



For the simple 1-bit addition problem above, the resulting carry bit could be ignored but you may have noticed something else with regards to the addition of these two bits, the sum of their binary addition resembles that of an **Exclusive-OR Gate**. If we label the two bits as A and B then the resulting truth table is the sum of the two bits but without the final carry.

Truth Table of Half Adder			
Inputs		**Outputs**	
B	**A**	**SUM**	**CARRY**
0	**0**	0	0
0	**1**	1	0
1	**0**	1	0
1	**1**	0	1



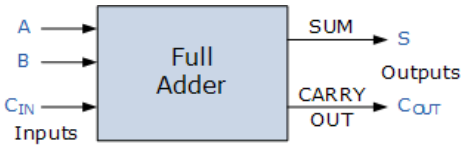
From the truth table of the half adder we can see that the SUM (S) output is the result of the Exclusive-OR gate and the Carry-out (Cout) is the result of the AND gate. Then the Boolean expression for a half adder is as follows.

One major disadvantage of the *Half Adder* circuit when used as a binary adder, is that there is no provision for a "Carry-in" from the previous circuit when adding together multiple data bits.

For example, suppose we want to add together two 8-bit bytes of data, any resulting carry bit would need to be able to "ripple" or move across the bit patterns starting from the least significant bit (LSB). The most complicated operation the half adder can do is "1 + 1" but as the half adder has no carry input the resultant added value would be incorrect. One simple way to overcome this problem is to use a **Full Adder** type binary adder circuit.

A Full Adder Circuit

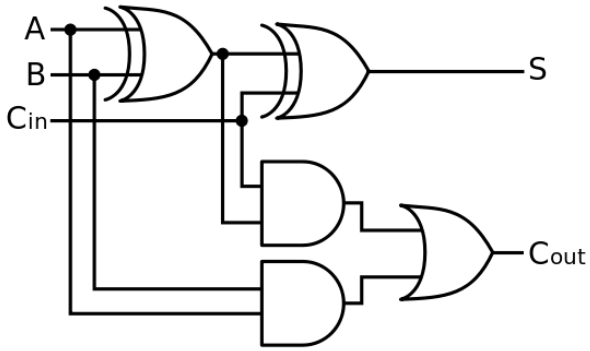
The main difference between the **Full Adder** and the previous **Half Adder** is that a full adder has three inputs. The same two single bit data inputs A and B as before plus an additional *Carry-in* (C-in) input to receive the carry from a previous stage as shown below.



Then the **full adder** is a logical circuit that performs an addition operation on three binary digits and just like the half adder, it also generates a carry out to the next addition column. Then a *Carry-in* is a possible carry from a less significant digit, while a *Carry-out* represents a carry to a more significant digit.

In many ways, the full adder can be thought of as two half adders connected together, with the first half adder passing its carry to the second half adder as shown.

Truth Table of Full Adder				
Inputs			**Output**	
C_{in}	**B**	**A**	**SUM**	**C_{out}**
0	**0**	**0**	0	0
0	**0**	**1**	1	0
0	**1**	**0**	1	0
0	**1**	**1**	0	1
1	**0**	**0**	1	0
1	**0**	**1**	0	1
1	**1**	**0**	0	1
1	**1**	**1**	1	1



An n-bit Binary Adder, ripple carry adder

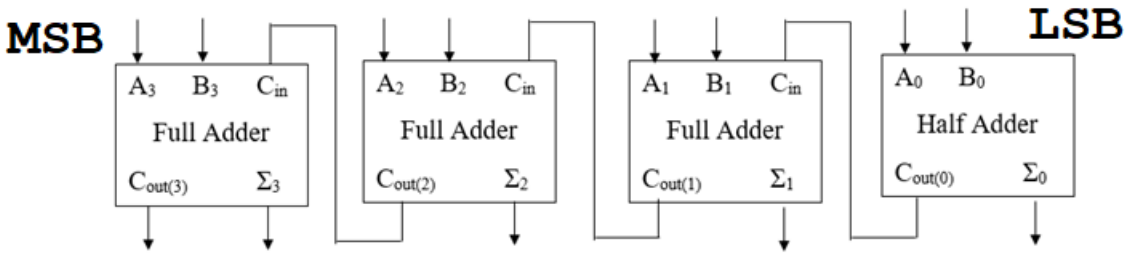
We have seen above that single 1-bit binary adders can be constructed from basic logic gates. But what if we wanted to add together two n-bit numbers, then n number of 1-bit full adders need to be connected or "cascaded" together to produce what is known as a **Ripple Carry Adder**.

A "ripple carry adder" is simply "n", 1-bit full adders cascaded together with each full adder representing a single weighted column in a long binary addition. It is called a ripple carry adder

because the carry signals produce a “ripple” effect through the binary adder from right to left, (LSB to MSB).

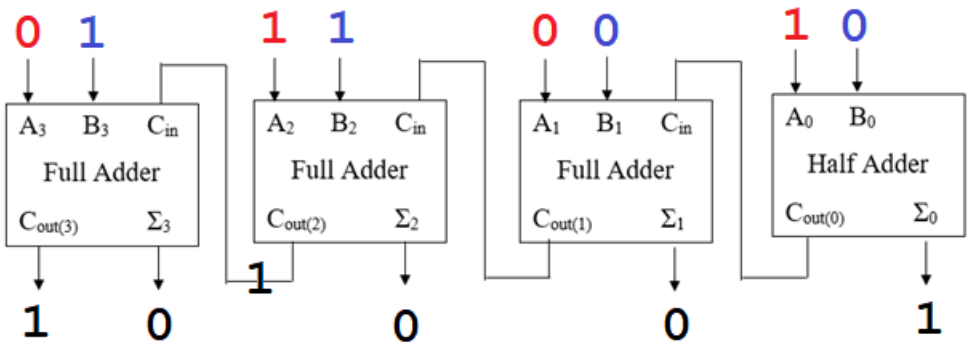
For example, suppose we want to “add” together two 4-bit code, the two outputs of the first full adder will provide the first place digit sum (S) of the addition plus a carry-out bit that acts as the carry-in digit of the next binary adder.

The second binary adder in the chain also produces a summed output (the 2nd bit) plus another carry-out bit and we can keep adding more full adders to the combination to add larger numbers, linking the carry bit output from the first full binary adder to the next full adder, and so forth. An example of a 4-bit adder is given below.



Example 7.2) Find the sum at each stage $\sum_3$, $\sum_2$, $\sum_1$, $\sum_0$, and the final $C_{out(3)}$ giving the following input:

A = 0101_2
B = 1100_2



One main disadvantage of “cascading” together 1-bit **binary adders** to add large binary numbers is that if inputs A and B change, the sum at its output will not be valid until any carry-input has “rippled” through every full adder in the chain because the MSB (most significant bit) of the sum has to wait for any changes from the carry input of the LSB (less significant bit). Consequently, there will be a finite delay before the output of the adder responds to any change in its inputs resulting in an accumulated delay.

When the size of the bits being added is not too large for example, 4 or 8 bits, or the summing speed of the adder is not important, this delay may not be important. However, when the size of the bits is larger for example 32 or 64 bits used in multi-bit adders, or summation is required at a very high clock speed, this delay may become prohibitively large with the addition processes not being completed correctly within one clock cycle.

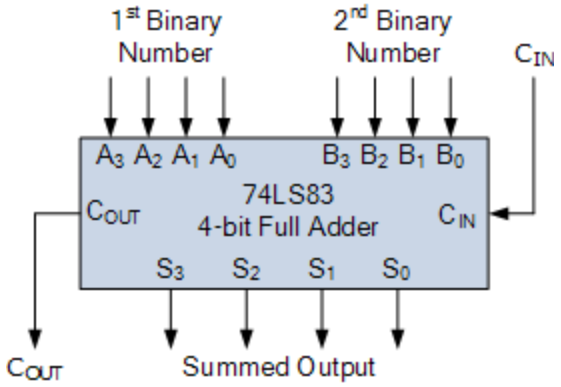
This unwanted delay time is called **Propagation delay**. Also another problem called "overflow" occurs when an n-bit adder adds two parallel numbers together whose sum is greater than or equal to 2^n

One solution is to generate the carry-input signals directly from the A and B inputs rather than using the ripple arrangement above. This then produces another type of binary adder circuit called a **Carry Look Ahead Binary Adder** where the speed of the parallel adder can be greatly improved using carry-look ahead logic.

The advantage of carry look ahead adders is that the length of time a carry look ahead adder needs in order to produce the correct SUM is independent of the number of data bits used in the operation, unlike the cycle time a parallel ripple adder needs to complete the SUM which is a function of the total number of bits in the addend.

4-bit full adder circuits with carry look ahead features are available as standard IC packages in the form of the TTL 4-bit binary adder 74LS83 or the 74LS283 and the CMOS 4008 which can add together two 4-bit binary numbers and generate a SUM and a CARRY output as shown.

74LS83 Logic Symbol



Digital Comparator

The Digital Comparator is another very useful combinational logic circuit used to compare the value of two binary digits.

Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of *Boolean Algebra*. There are two main types of **Digital Comparator** available and these are.

1. Identity Comparator – an *Identity Comparator* is a digital comparator with only one output terminal for when A = B, either A = B = 1 (HIGH) or A = B = 0 (LOW)
2. Magnitude Comparator – a *Magnitude Comparator* is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

The purpose of a **Digital Comparator** is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, …. A_n, etc) against that of a constant or unknown value such as B (B1, B2, B3, …. B_n, etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

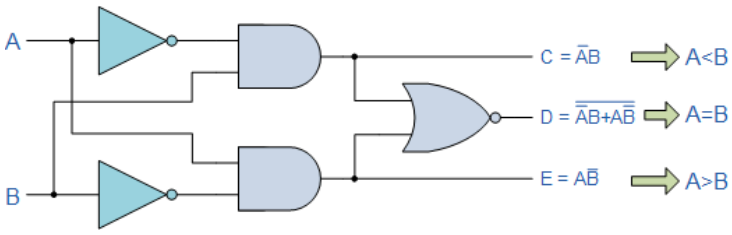
$$\mathbf{A > B,\ A = B,\ A < B}$$

Which means: A is greater than B, A is equal to B, or A is less than B

This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

1-bit Digital Comparator Circuit

Then the operation of a 1-bit digital comparator is given in the following logic circuit and Truth Table.



Magnitude Truth Table				
Inputs		**Output**		
B	**A**	**A>B**	**A=B**	**A<B**
0	**0**	0	1	0
0	**1**	1	0	0
1	**0**	0	0	1
1	**1**	0	1	0

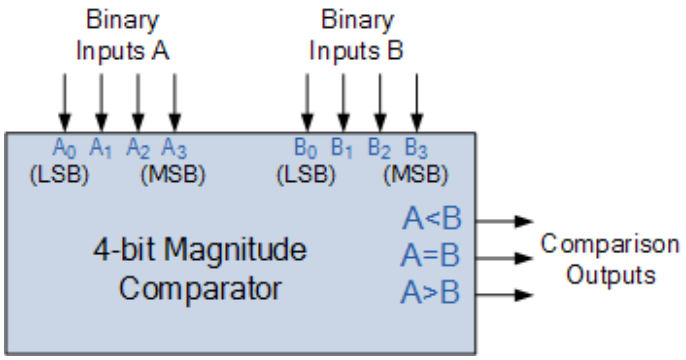
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = B resembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR function (equivalence) on each of the n-bits giving: $Q = A \oplus B$

Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the "magnitude" of these values, a logic "0" against a logic "1" which is where the term **Magnitude Comparator** comes from.

As well as comparing individual bits, we can design larger bit comparators by cascading together ***n*** of these and produce an n-bit comparator. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other.

A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words ("nibbles") are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

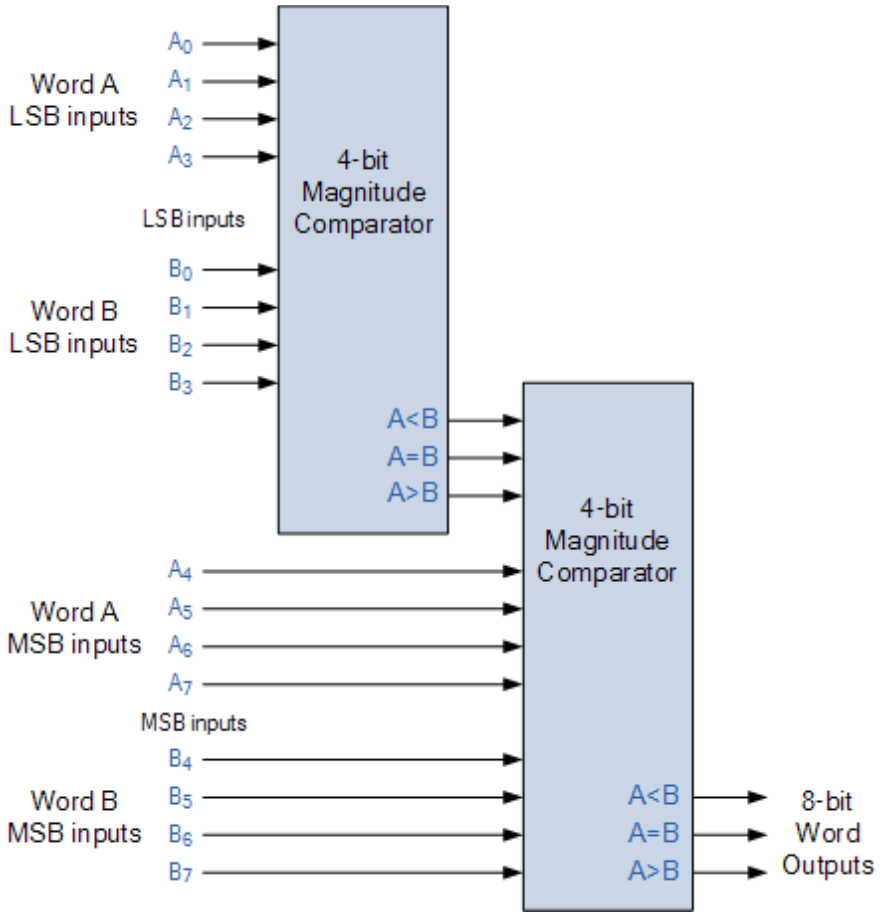
4-bit Magnitude Comparator



Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

8-bit Word Comparator

When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists, A = B then it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal.



If inequality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits stops. **Digital Comparator** are used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations. (Comparators, 2019)

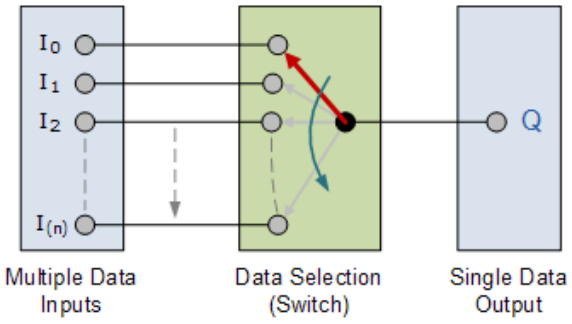
The Multiplexer

The multiplexer is a combinational logic circuit designed to switch one of several input lines to a single common output line.

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a **Multiplexer**.

The *multiplexer*, shortened to "MUX" or "MPX", is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output.

The most basic type of multiplexer device is that of a one-way rotary switch.



The rotary switch, also called a wafer switch as each layer of the switch is known as a wafer, is a mechanical device whose input is selected by rotating a shaft. In other words, the rotary switch is a manual switch that you can use to select individual data or signal lines simply by turning its inputs "ON" or "OFF". So how can we select each data input automatically using a digital device.

In digital electronics, multiplexers are also known as **data selectors** because they can "select" each input line, are constructed from individual Analogue Switches encased in a single IC package as opposed to the "mechanical" type selectors such as normal conventional switches and relays.

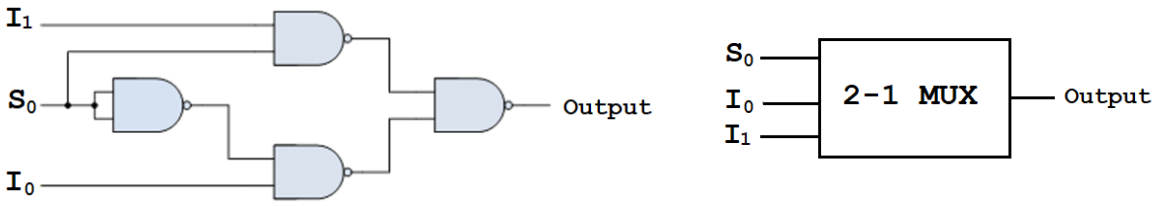
They are used as one method of reducing the number of logic gates required in a circuit design or when a single data line or data bus is required to carry two or more different digital signals. For example, a single 8-channel multiplexer.

Generally, the selection of each input line in a multiplexer is controlled by an additional set of inputs called *control lines* and according to the binary condition of these control inputs, either "HIGH" or "LOW" the appropriate data input is connected directly to the output. Normally, a multiplexer has an even number of 2^n data input lines and a number of "control" inputs that correspond with the number of data inputs.

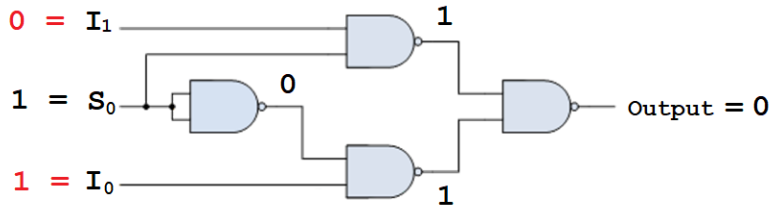
Note that multiplexers are different in operation to *Encoders*. Encoders are able to switch an n-bit input pattern to multiple output lines that represent the binary coded (BCD) output equivalent of the active input.

We can build a simple 2-line to 1-line (2-to-1) multiplexer from basic logic NAND gates as shown.

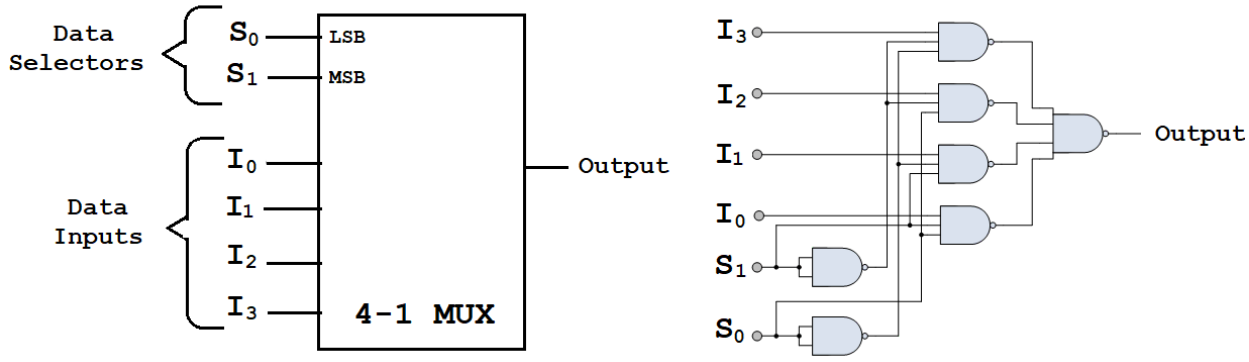
2-to-1 Multiplexer Design



If we select $S_0 = 1$ and set $I_0 = 1$ and $I_1 = 0$, the **Output** will be 0. It is because from $S_0 = 1$ we are selecting input data 1, I_1, and input data 1 is set to 0, $I_1 = 0$.



4-to-1 Multiplexer Design



From the circuit below, if you set the data selector to

$S_0 = 0$

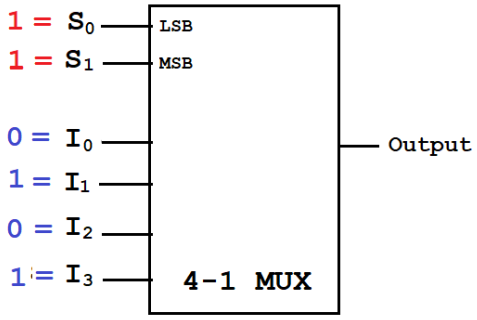
$S_1 = 1$
and the data inputs to:

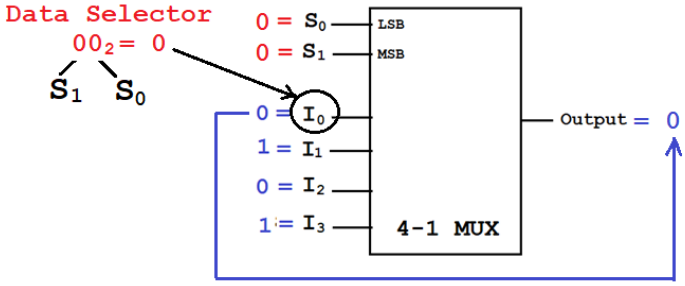
$I_0 = 1$
$I_1 = 1$
$I_2 = 0$
$I_3 = 1$

In this case, the data selector is $10_2 = 2$ which is selected data input 2. As data input 2 is set to 0, then the output of this circuit will be 0.

Example 7.3) Find the output of the following 4-1 MUX inputs

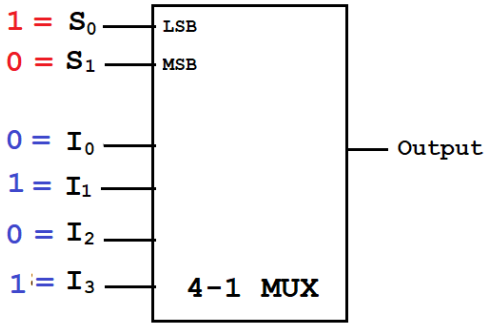
a)

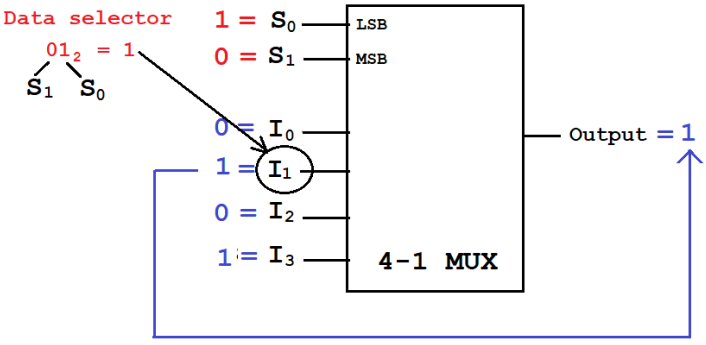




Output = 0

b)





Output = 1

Binary Decoder

Binary Decoder is another combinational logic circuit constructed from individual logic gates.

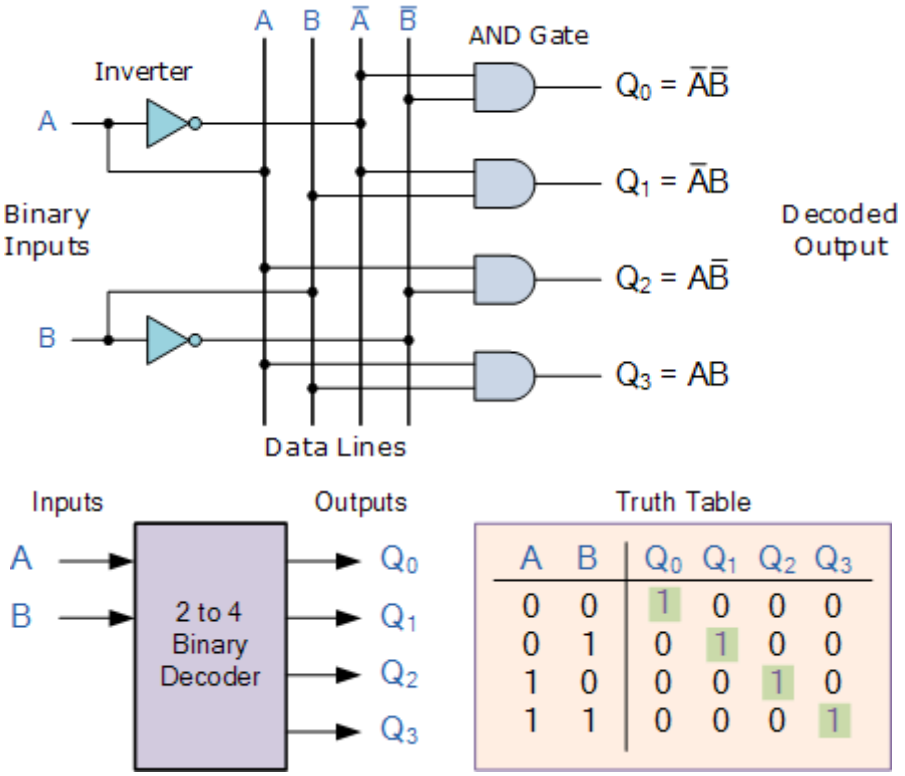
The name "Decoder" means to translate or decode coded information from one format into another, so a binary decoder transforms "n" binary input signals into an equivalent code using 2^n outputs.

Binary Decoders are another type of digital logic device that has inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, so a decoder that has a set of two or more bits will be defined as having an *n*-bit code, and therefore it will be possible to represent 2^n possible values. Thus, a decoder generally decodes a binary value into a non-binary one by setting exactly one of its *n* outputs to logic "1".

A *Binary Decoder* converts coded inputs into coded outputs, where the input and output codes are different and decoders are available to "decode" either a Binary or BCD (8421 code) input pattern to typically a Decimal output code. Commonly available BCD-to-Decimal decoders include the TTL 7442 or the CMOS 4028. Generally a decoder output code normally has more bits than its input code and practical "binary decoder" circuits include, 2-to-4, 3-to-8 and 4-to-16 line configurations.

An example of a 2-to-4 line decoder along with its truth table is given as:

A 2-to-4 Binary Decoders



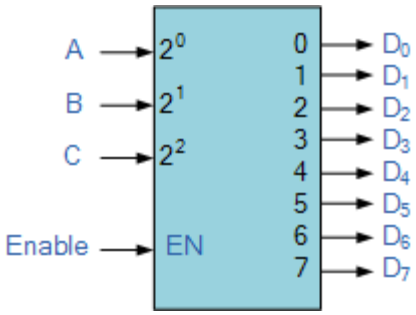
Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

This simple example above of a 2-to-4 line binary decoder consists of an array of four AND gates. The 2 binary inputs labelled A and B are decoded into one of 4 outputs, hence the description of 2-to-4 binary decoder. Each output represents one of the miniterms of the 2 input variables, (each output = a miniterm).

The binary inputs A and B determine which output line from Q0 to Q3 is “HIGH” at logic level “1” while the remaining outputs are held “LOW” at logic “0” so only one output can be active (HIGH) at any one time. Therefore, whichever output line is “HIGH” identifies the binary code present at the input, in other words it “de-codes” the binary input.

Some binary decoders have an additional input pin labelled “Enable” that controls the outputs from the device. This extra input allows the decoders outputs to be turned “ON” or “OFF” as required. These types of binary decoders are commonly used as “memory address decoders” in microprocessor memory applications.



74LS138 Binary Decoder

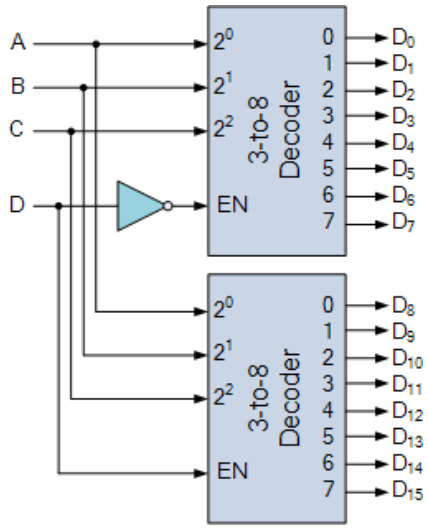
We can say that a binary decoder is a demultiplexer with an additional data line that is used to enable the decoder. An alternative way of looking at the decoder circuit is to regard inputs A, B and C as address signals. Each combination of A, B or C defines a unique memory address.

We have seen that a 2-to-4 line binary decoder (TTL 74155) can be used for decoding any 2-bit binary code to provide four outputs, one for each possible input combination. However, sometimes it is required to have a **Binary Decoder** with a number of outputs greater than is available, so by adding more inputs, the decoder can potentially provide 2^n more outputs.

So for example, a decoder with 3 binary inputs (n = 3), would produce a 3-to-8 line decoder (TTL 74138) and 4 inputs (n = 4) would produce a 4-to-16 line decoder (TTL 74154) and so on. But a decoder can also have less than 2^n outputs such as the BCD to seven-segment decoder (TTL 7447) which has 4 inputs and only 7 active outputs to drive a display rather than the full 16 (2^4) outputs as you would expect.

Here a much larger 4 (3 data plus 1 enable) to 16 line binary decoder has been implemented using two smaller 3-to-8 decoders.

A 4-to-16 Binary Decoder Configuration

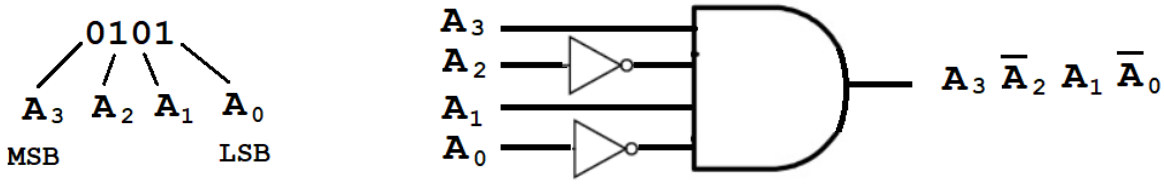


4-to-16 line decoder implemented with two 3-to-8 decoders

Inputs A, B, C are used to select which output on either decoder will be at logic "1" (HIGH) and input D is used with the enable input to select which encoder either the first or second will output the "1".

However, there is a limit to the number of inputs that can be used for one particular decoder, because as *n* increases, the number of AND gates required to produce an output also becomes larger resulting in the fan-out of the gates used to drive them becoming large.

For example, a 4-input AND gate to decode number 5, which is 0101_2, is designed as the following:

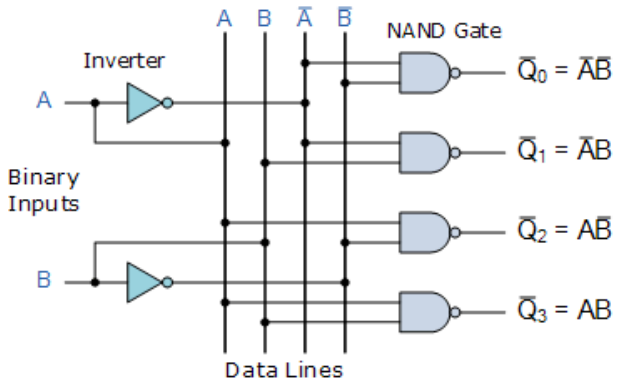


This type of active-"HIGH" decoder can be implemented using just Inverters, (NOT Gates) and AND gates. It is convenient to use an AND gate as the basic decoding element for the output because it produces a "HIGH" or logic "1" output only when all of its inputs are logic "1".

But some binary decoders are constructed using NAND gates instead of AND gates for their decoded output, since NAND gates are cheaper to produce than AND's as they require fewer transistors to implement within their design.

The use of NAND gates as the decoding element, results in an active-"LOW" output while the rest will be "HIGH". As a NAND gate produces the AND operation with an inverted output, the NAND decoder looks like this with its inverted truth table.

2-to-4 Line NAND Binary Decoder



Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Then for the NAND decoder, only one output can be LOW and equal to logic "0" at any given time, with all the other outputs being HIGH at logic "1".

Decoders are also available with an additional "Enable" input pin which allows the decoded output to be turned "ON" or "OFF" by applying a logic "1" or logic "0" respectively to it. So for example, when the enable input is at logic level "0", (EN = 0) all outputs are "OFF" at logic "0" (for AND gates) regardless of the state of the inputs A and B.

Generally to implement this enabling function the 2-input AND or NAND gates are replaced with 3-input AND or NAND gates. The additional input pin represents the enable function.

Memory Address Decoder

Binary Decoders are most often used in more complex digital systems to access a particular memory location based on an "address" produced by a computing device. In modern microprocessor systems the amount of memory required can be quite high and is generally more than one single memory chip alone.

One method of overcoming this problem is to connect lots of individual memory chips together and to read the data on a common "Data Bus". In order to prevent the data being "read" from each memory chip at the same time, each memory chip is selected individually one at time and this process is known as **Address Decoding**.

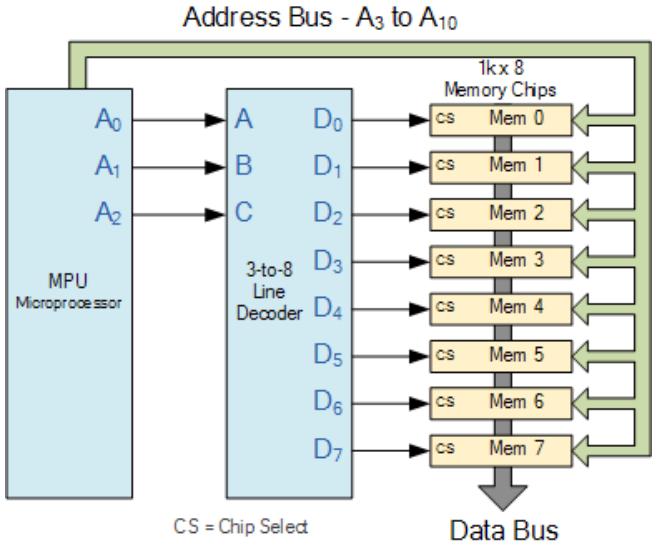
In this type of application, the address represents the coded data input, and the outputs are the particular memory element select signals. Each memory chip has an input called **Chip Select** or **CS** which is used by the MPU (micro-processor unit) to select the appropriate memory

chip when required. Generally a logic “1” on the chip select (CS) input selects the memory device while a logic “0” on the input de-selects it.

So by selecting or de-selecting each chip one at a time, allows us to select the correct memory address device for a particular address location. The advantage of address decoding is that when we specify a particular memory address, the corresponding memory location exists ONLY in one of the chips.

For example, Lets assume we have a very simple microprocessor system with only 1Kb (one thousand bytes) of RAM memory and 10 memory address lines available. The memory consists of 128×8-bit (128×8 = 1024 bytes) devices and for 1Kb we would need 8 individual memory chips but in order to select the correct memory chip we would also require a 3-to-8 line binary decoder as shown below.

Memory Address Decoding



The binary decoder requires only 3 address lines, (A_0 to A_2) to select each one of the 8 chips (the lower part of the address), while the remaining 8 address lines (A_3 to A_{10}) select the correct memory location on that chip (the upper part of the address).

Having selected a memory location using the address bus, the information at the particular internal memory location is sent to a common “Data Bus” for use by the microprocessor. This is of course a simple example but the principals remain the same for all types of memory chips or modules.

Binary Decoders are very useful devices for converting one digital format to another, such as binary or BCD type data into decimal or octal etc and commonly available decoder IC’s are the TTL 74LS138 3-to-8 line binary decoder or the 74ALS154 4-to-16 line decoder. They are also very useful for interfacing to 7-segment displays such as the TTL 74LS47 which we will look at in the next tutorial.

Display Decoder: 7-segment display

A Display Decoder is a combinational circuit which decodes and n-bit input value into a number of output lines to drive a display. A **Digital Decoder** IC, is a device which converts one digital format into another and one of the most commonly used devices for doing this is called the Binary Coded Decimal (BCD) to 7-Segment Display Decoder.

7-segment **LED** (Light Emitting Diode) or **LCD** (Liquid Crystal Display) type displays, provide a very convenient way of displaying information or digital data in the form of numbers, letters or even alpha-numerical characters.

Typically **7-segment displays** consist of seven individual coloured LED's (called the segments), within one single display package. In order to produce the required numbers or HEX characters from 0 to 9 and A to F respectively, on the display the correct combination of LED segments need to be illuminated and **BCD to 7-segment Display Decoders** such as the 74LS47 do just that.

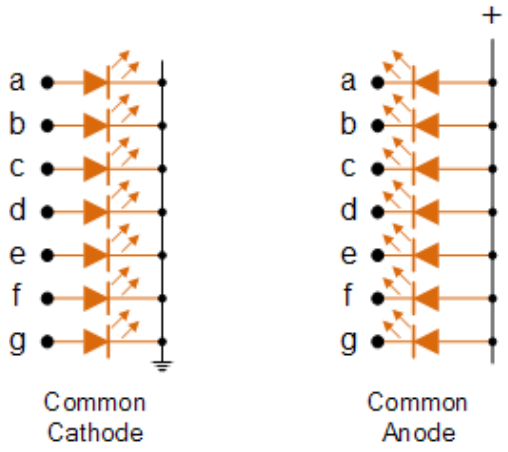
A standard 7-segment LED display generally has eight (8) input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal display segments. Some single displays have also have an additional input pin to display a decimal point in their lower right or left hand corner.

In electronics there are two important types of 7-segment LED digital display.

1. The Common Cathode Display (CCD) – In the common cathode display, all the cathode connections of the LED's are joined together to logic "0" or ground. The individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual Anode terminals.

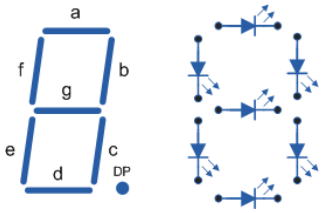
2. The Common Anode Display (CAD) – In the common anode display, all the anode connections of the LED's are joined together to logic "1" and the individual segments are illuminated by connecting the individual Cathode terminals to a "LOW", logic "0" signal.

Common Cathode and Common Anode Format

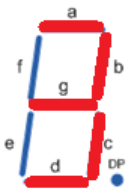


Electrical connection of the individual diodes for a common cathode display and a common anode display and by illuminating each light emitting diode individually, they can be made to display a variety of numbers or characters.

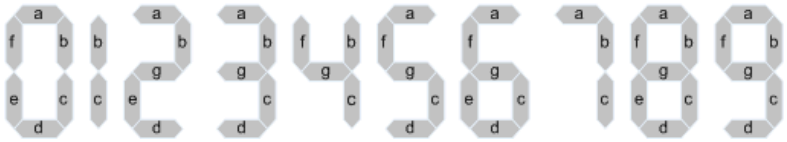
7-Segment Display Format



So in order to display the number "3" for example, segments a, b, c, d and g would need to be illuminated.



If we wanted to display a different number or letter then a different set of segments would need to be illuminated.



7-Segment Display Elements for all Numbers.

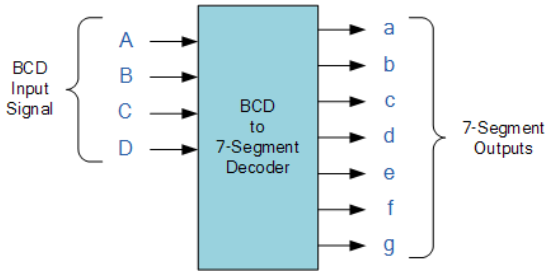
It can be seen that to display any single digit number from 0 to 9 in binary or letters from A to F in hexadecimal, we would require seven separate segment connections plus one additional connection for the LED's "common" connection. Also as the segments are basically a standard light emitting diode, the driving circuit would need to produce up to 20mA of current to illuminate each individual segment and to display the number "8", all seven segments would need to be lit resulting a total current of nearly 140mA, (8 x 20mA).

Obviously, the use of so many connections and power consumption is impractical for some electronic or microprocessor based circuits and so in order to reduce the number of signal lines required to drive just one single display, display decoders such as the BCD to 7-Segment Display Decoder and Driver IC's are used instead.

BCD to 7-Segment Display Decoders

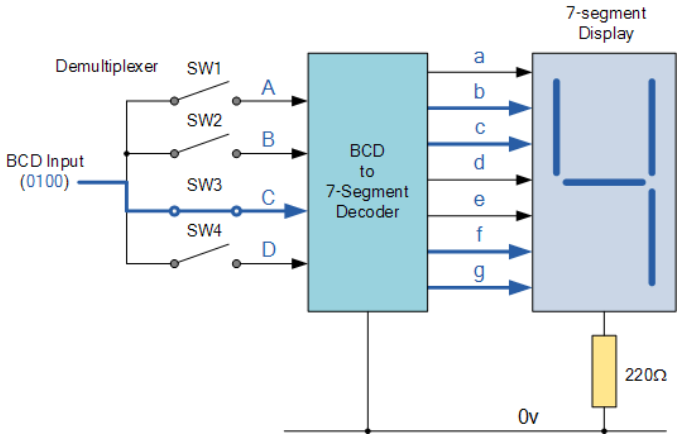
A binary coded decimal (BCD) to 7-segment display decoder such as the TTL 74LS47 or 74LS48, have 4 BCD inputs and 7 output lines, one for each LED segment. This allows a smaller 4-bit binary number (half a byte) to be used to display all the denary numbers from 0 to 9 and by adding two displays together, a full range of numbers from 00 to 99 can be displayed with just a single byte of eight data bits.

BCD to 7-Segment Decoder



The use of **packed** BCD allows two BCD digits to be stored within a single byte (8-bits) of data, allowing a single data byte to hold a BCD number in the range of 00 to 99.

An example of the 4-bit BCD input (0100) representing the number “4” is given below.



In practice current limiting resistors of about 150Ω to 220Ω would be connected in series between the decoder/driver chip and each LED display segment to limit the maximum current flow. There are different display decoders and drivers available for the different types of available displays, either LED or LCD. For example, the 74LS48 for common-cathode LED types, the 74LS47 for common-anode LED types, or the CMOS CD4543 for liquid crystal display (LCD) types.

Liquid crystal displays (LCD´s) have one major advantage over similar LED types in that they consume much less power and nowadays, both LCD and LED displays are combined together to form larger Dot-Matrix Alphanumeric type displays which can show letters and characters as well as numbers in standard Red or Tri-color outputs.

References

Binary Adder. (2019). Retrieved from Electronics Tutorials: https://www.electronics-tutorials.ws/combination/comb_7.html

Comparators. (2019). Retrieved from Electronics Tutorial: https://www.electronics-tutorials.ws/combination/comb_8.html

Electronics Tutotials. (2019). Retrieved from Combinational Logic Circuits: https://www.electronics-tutorials.ws/combination/comb_1.html