UNIT-3

Basic Digital Logical Circuits:

- Integrated circuits.
- Combinational Circuits-(Encoder, Decoder, Multiplexer, De-multiplexer, Compotator).
- Arithmetic Circuit (Half Adder, Full adder, Binary adder/ sub tractor)

PREPARING CIRCUIT FROM GIVEN BOOLEAN FUNCTION

In a logic circuit, the variables coming on the left hand side of Boolean expression are inputs to circuit and the variable function coming on the right hand side of expression is taken as output. A Boolean function can be implemented into a logic circuit using the basic gates:- AND , OR & NOT

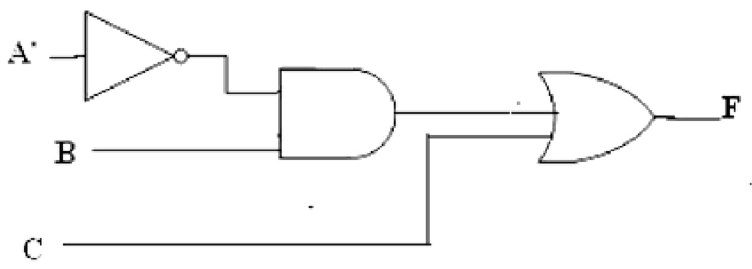
❖ **EXAMPLE NO 1 :** Prepare Truth table and circuit using given Boolean function

BOOLEAN FUNCTION: - F (A,B,C) = A'. B + C

The relationship between this function and its binary variables A, B, C can be represented in a truth table as shown below:

➢ **TRUTH TABLE**

	Inputs		Intermediates		Output
A	B	C	A'	A'B	F
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	0	0
1	1	1	0	0	1



➢ LOGIC DIAGRAM / LOGIC CIRCUIT

There are two methods for converting truth tables to boolean expressions.

1. The Sum of Products
2. Product of Sum

- A binary variable may appear either in its normal form (x) or in its complement form (x').Now consider two binary variables x and y combined with an AND operations. Since each variable may appear in either form, there are four possible combinations : x'y', x'y, xy' and xy. Each of these four AND terms called a **minterm** or a **standard product**.
- N variables can be combined to form 2^n minterms.
- Similarly n variables forming an OR terms, with each variable being primed or unprimed, provide 2^n possible combinations, called **maxterm** or **standard product**.

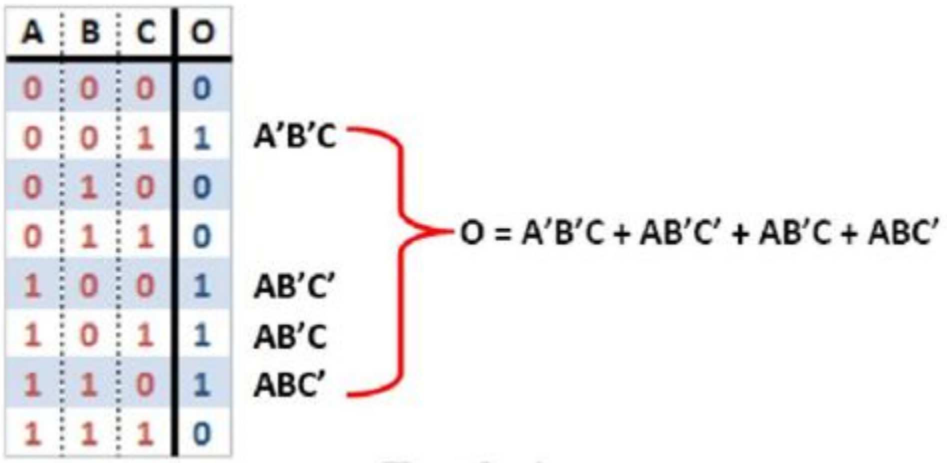
❖ SUM OF PRODUCT (SOP)

- A simple method for converting a truth table in a standard form of Boolean expression called the **Sum-of-Products (SOP)** form.
- SOP expressions can easily be implemented as a set of AND gates feeding into a single OR gate
- An SOP expression is literally a sum of Boolean terms called minterms. A minterm is a multiplicative combination of Boolean variables whose output equals 1.
- An example of an SOP expression is ABC + BC + DE, where ABC, BC, and DE are minterms. SOP expressions may be generated from truth tables using the following steps:

1. Determine which rows of the table have an output of 1.
2. Derive each row's minterm, such that the output is 1 given that row's input state.
3. Sum the minterms.

EXAMPLE NO 1 : A Truth Table conversion to an SOP expression. Prepare circuit and Boolean expression from given truth table.

➢ **TRUTH TABLE**



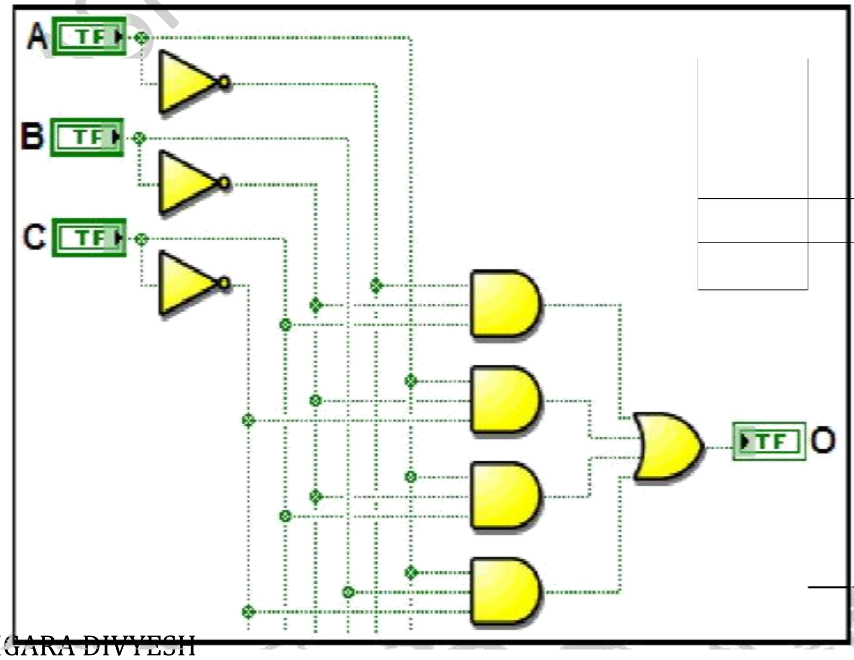
A	B	C	O	
0	0	0	0	
0	0	1	1	A'B'C
0	1	0	0	
0	1	1	0	
1	0	0	1	AB'C'
1	0	1	1	AB'C
1	1	0	1	ABC'
1	1	1	0	

O = A'B'C + AB'C' + AB'C + ABC'

BOOLEAN FUNCTION O = A'B'C + AB'C' + AB'C + ABC'

BOOLEAN FUNCTION O = A'B'C + AB'C' + AB'C + ABC'

➢ **LOGIC DIAGRAM / LOGIC CIRCUIT**



❖ **EXAMPLE NO 2: PREPARE BOOLEAN FUNCTION AND CIRCUIT FROM GIVEN BOOLEAN FUNCTION**

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

EXPLANATION

- The input variables are X, Y and Z, and the function output is F.
- Let's examine this function in some detail. The only non-zero entries are at:

$$X = 0, Y = 1, Z = 0$$

and

$$X = 1, Y = 0, Z = 1$$

The function is 1 for those two input conditions and zero for all other input conditions.

Now, lets' think about how we can implement this function. Here's a description of what we want to implement:

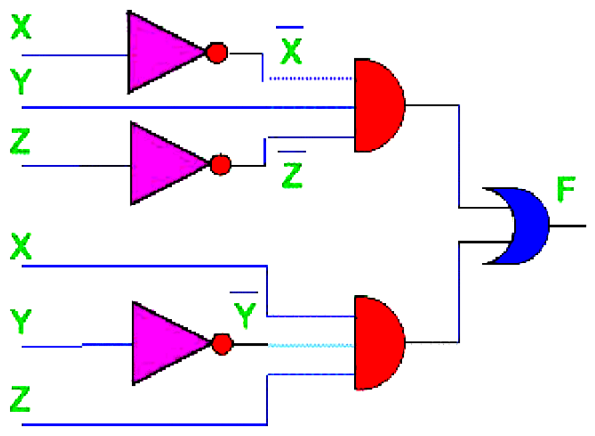
- We want the output to be 1 whenever we have either
  X=0 AND Y=1 AND Z=0
- OR when we have
    - X=1 AND Y=0 AND Z=1.

Here's the function:

$$F = \overline{X} \bullet Y \bullet \overline{Z} + X \bullet \overline{Y} \bullet Z$$

This function is read as (NOT-X AND Y AND NOT-Z) OR (X AND NOT-Y AND Z) and when we read NOT-X that means we have to have X=0 to make the three terms ANDed together work out to 1.

Now, let's look at a circuit that will implement this function. Here's the circuit. Notice how the inputs are grouped into groups of 3, ANDed together (after taking inverses where appropriate) and the results ORed at the end.



In producing our circuit we had to use the form:

$$F = \overline{X} \bullet Y \bullet \overline{Z} + X \bullet \overline{Y} \bullet Z$$

This form is composed of two groups of three. Each group of three is a minterm. What the expression minterm is intended to imply it that each of the groups of three in the expression takes

on a value of 1 only for one of the eight possible combinations of X, Y and Z and their inverses. Important points about minterms include the following.

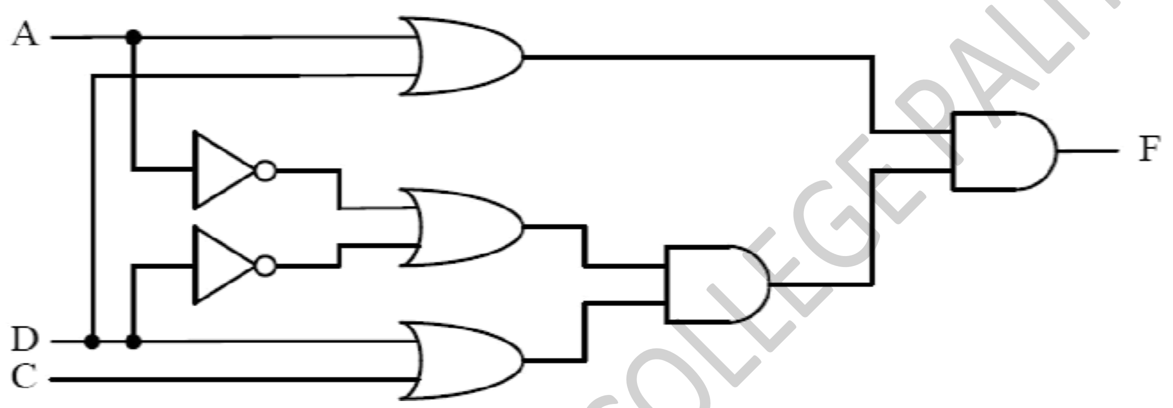
- In a minterm, each variable, X, Y or Z appears once, either as the variable itself or as the inverse.
- Each minterm corresponds to exactly one entry (row!) in the truth table.

A truth table gives a unique sum-of-products function that follows directly from expanding the ones in the truth table as minterms.

EXAMPLE NO 1 : Prepare Circuit from given boolean function

BOOLEAN FUCTION : F(A,B,C,D) = (A + D) . (A' + D') . (C + D)

➢ **LOGIC DIAGRAM / LOGIC CIRCUIT**



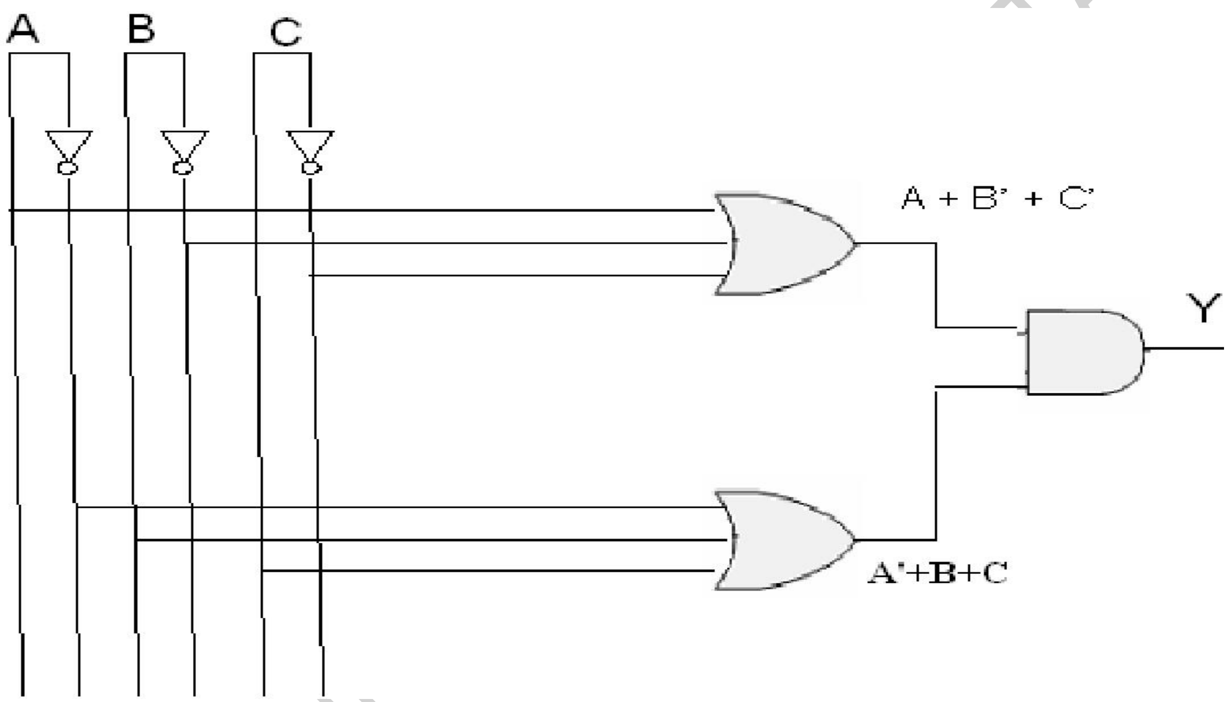
EXAMPLE NO 2 : A Truth Table conversion to an POS expression. Prepare circuit and Boolean function using given truth table.

TRUTH TABLE

Inputs			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

BOOLEAN FUNCTION : Y = (A +B’+C’) . (A’ + B +C)

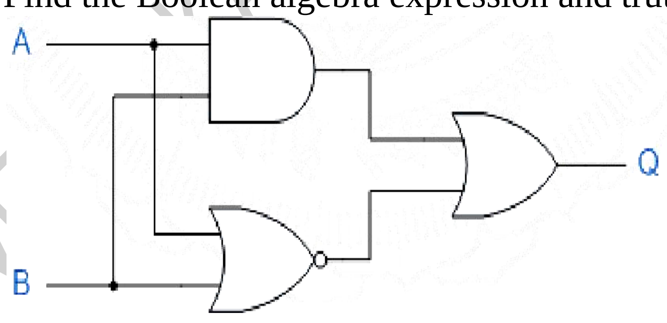
➢ **LOGIC DIAGRAM / LOGIC CIRCUIT**



PREPARING A TRUTH TABLE FROM CIRCUIT

EXAMPLE NO 1

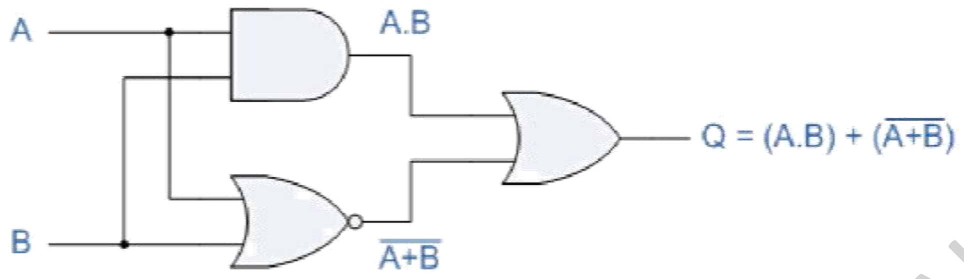
Find the Boolean algebra expression and truth table for the following circuit.



The circuit consists of an AND Gate, a NOR Gate and finally an OR Gate.

The expression for the AND gate is A.B, and the expression for the NOR gate is A+B.

Both these expressions are also separate inputs to the OR gate which is defined as A+B. Thus the final output expression is given as



The output of the system is given as

BOOLEAN EXPRESSION / BOOLEAN FUNCTION

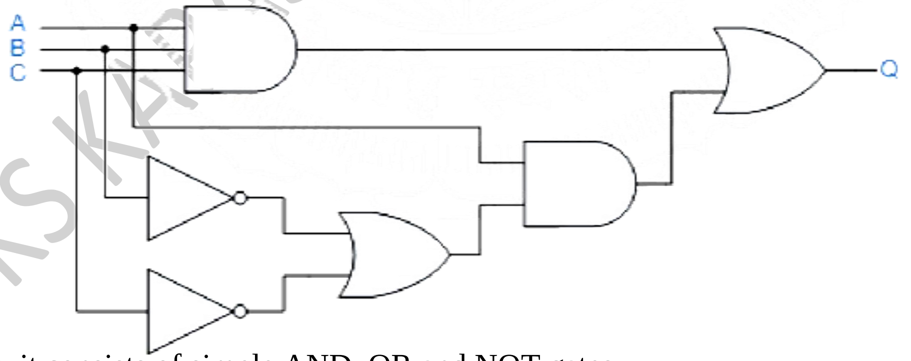
Q = (A.B) + (A+B)

TRUTH TABLE

Inputs		Intermediates		Output
B	A	A.B	A + B	Q
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

❖ **EXAMPLE NO 2**

Find the Boolean algebra expression and truth table for the following circuit.



The Circuit consists of simple AND, OR and NOT gates.

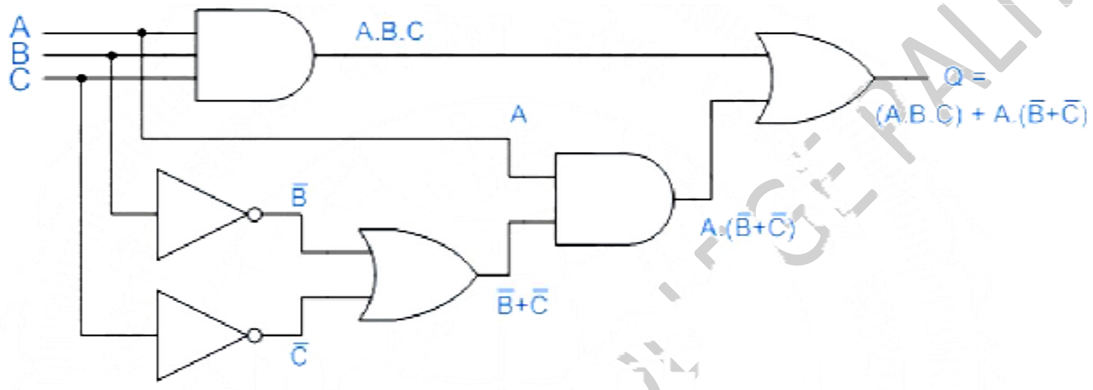
The output from the 3-input AND gate is only a "1" when **ALL** the inputs are at logic level "1" (A.B.C). The output from the lower OR gate is only a "1" when one or both inputs B or C are at logic level "0".

The output from the 2-input AND gate is a "1" when input A is a "1" and inputs B or C are at "0".

Then the output at Q is only a "1" when inputs A.B.C equal "1" or A is equal to "1" and both inputs B or C equal "0", A.(B+C).

Thus the final output expression is given as:

➢ **LOGIC DIAGRAM / LOGIC CIRCUIT**


A
B
C
A.B.C
A
$\overline{B}$
$\overline{C}$
$\overline{B}+\overline{C}$
$A.(\overline{B}+\overline{C})$
Q =
$(A.B.C) + A.(\overline{B}+\overline{C})$

BOOLEAN EXPRESSION / BOOLEAN FUNCTION
Q = (A.B.C) + A. (B’+C’)

➢ **TRUTH TABLE**

Inputs			Intermediates					Output
C	B	A	A.B.C	B	C	$\overline{B}$+C	A.(B+C)	Q
0	0	0	0	1	1	1	0	0
0	0	1	0	1	1	1	1	1
0	1	0	0	0	1	1	0	0
0	1	1	0	0	1	1	1	1
1	0	0	0	1	0	1	0	0
1	0	1	0	1	0	1	1	1
1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	1

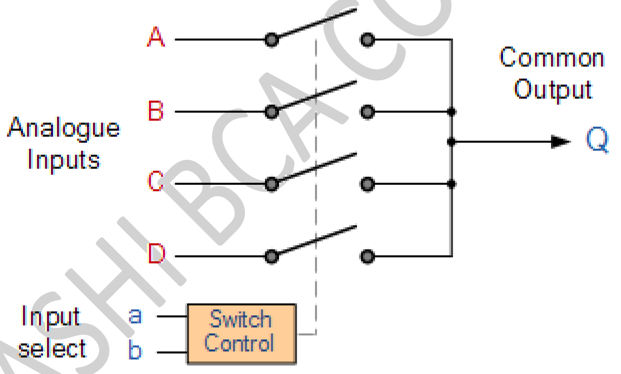
Multiplexer

A data selector, more commonly called a **Multiplexer**, shortened to "Mux" or "MPX", are combinational logic switching devices that operate like a very fast acting multiple position rotary switch. They connect or control, multiple input lines called "channels" consisting of either 2, 4, 8 or 16 individual inputs, one at a time to an output.

Then the job of a "multiplexer" is to allow multiple signals to *share* a single common output. For example, a single 8-channel multiplexer would connect one of its eight inputs to the single data output. Multiplexers are used as one method of reducing the number of logic gates required in a circuit or when a single data line is required to carry two or more different digital signals.

Digital **Multiplexers** 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. Generally, multiplexers have an even number of data inputs, usually an even power of two, n^2 , a number of "control" inputs that correspond with the number of data inputs and according to the binary condition of these control inputs, the appropriate data input is connected directly to the output. An example of a **Multiplexer** configuration is shown below.

4-to-1 Channel Multiplexer



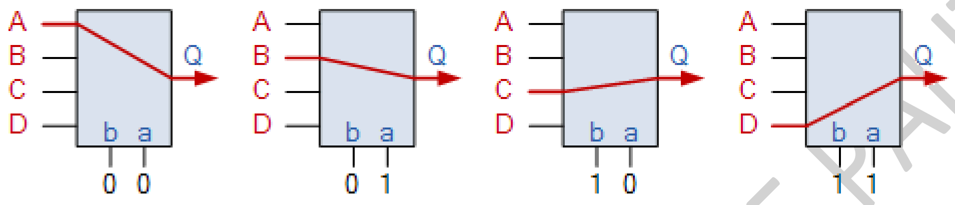
Addressing		Input Selected
b	A	
0	0	A
0	1	B
1	0	C
1	1	D

The Boolean expression for this 4-to-1 **Multiplexer** above with inputs A to D and data select lines a, b is given as:

$$Q = abA + abB + abC + abD$$

In this example at any one instant in time only ONE of the four analogue switches is closed, connecting only one of the input lines A to D to the single output at Q. As to which switch is closed depends upon the addressing input code on lines "a" and "b", so for this example to select input B to the output at Q, the binary input address would need to be "a" = logic "1" and "b" = logic "0".

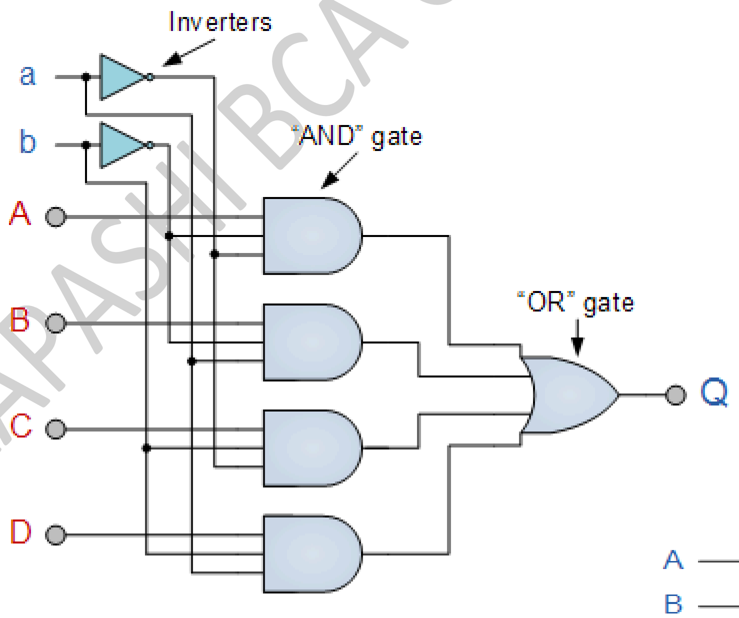
Then we can show the selection of the data through the multiplexer as a function of the data select bits as shown.



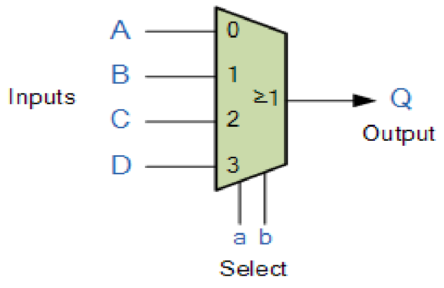
Adding more control address lines will allow the multiplexer to control more inputs but each control line configuration will connect only ONE input to the output.

Then the implementation of this Boolean expression above using individual logic gates would require the use of seven individual gates consisting of AND, OR and NOT gates as shown.

4 Channel Multiplexer using Logic Gates



The symbol used in logic diagrams to identify a multiplexer is as follows.



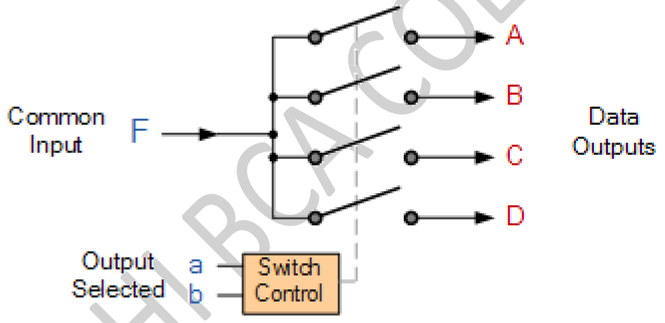
Multiplexer Symbol

Multiplexers are not limited to just switching a number of different input lines or channels to one common single output. There are also types that can switch their inputs to multiple outputs and have arrangements or 4 to 2, 8 to 3 or even 16 to 4 etc.

Demultiplexer

The data distributor, known more commonly as a **Demultiplexer** or "Demux", is the exact opposite of theMultiplexer we saw in the previous tutorial. The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The **demultiplexer**converts a serial data signal at the input to a parallel data at its output lines as shown below.

1-to-4 Channel De-multiplexer

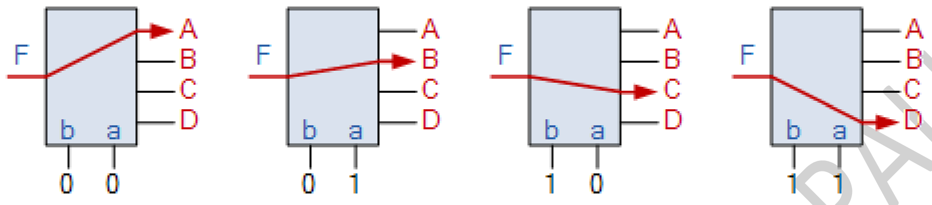


Addressing		Input Selected
b	a	
0	0	A
0	1	B
1	0	C
1	1	D

The Boolean expression for this 1-to-4 **Demultiplexer** above with outputs A to D and data select lines a, b is given as:

F = ab A + abB + abC + abD

The function of the **Demultiplexer** is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" as shown.

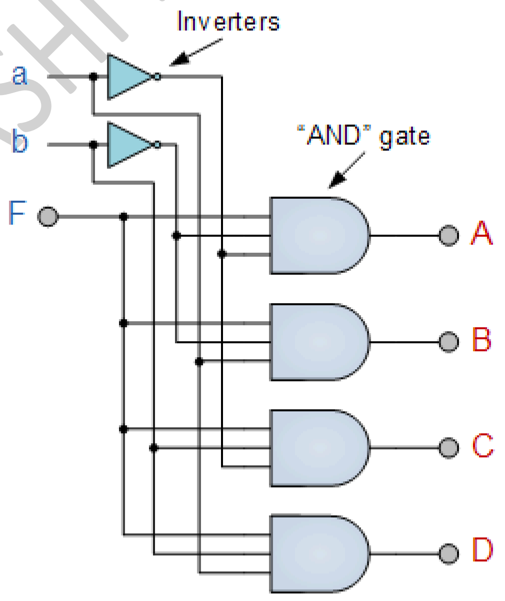


As with the previous **multiplexer circuit**, adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs.

Some standard demultiplexer IC´s also have an additional "enable output" pin which disables or prevents the input from being passed to the selected output. Also some have latches built into their outputs to maintain the output logic level after the address inputs have been changed. However, in standard decoder type circuits the address input will determine which single data output will have the same value as the data input with all other data outputs having the value of logic "0".

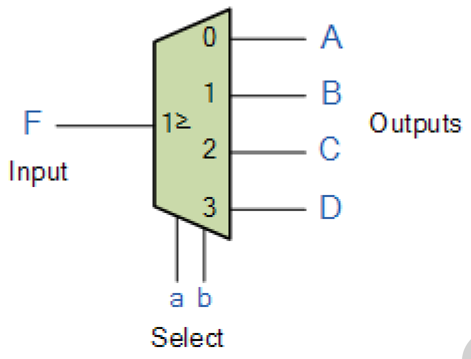
The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown.

4 Channel Demultiplexer using Logic Gates



The symbol used in logic diagrams to identify a demultiplexer is as follows.

The Demultiplexer Symbol



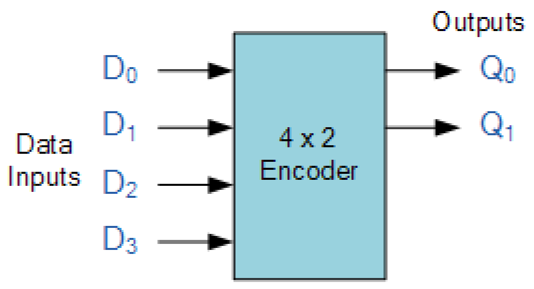
Standard **Demultiplexer** IC packages available are the TTL 74LS138 1 to 8-output demultiplexer, the TTL 74LS139 Dual 1-to-4 output demultiplexer or the CMOS CD4514 1-to-16 output demultiplexer. Another type of demultiplexer is the 24-pin, 74LS154 which is a 4-bit to 16-line demultiplexer/decoder. Here the individual output positions are selected using a 4-bit binary coded input. Like multiplexers, demultiplexers can also be cascaded together to form higher order demultiplexers.

The Digital Encoder

Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, a **Digital Encoder** more commonly called a **Binary Encoder** takes *ALL* its data inputs one at a time and then converts them into a single encoded output. So we can say that a binary encoder, is a multi-input combinational logic circuit that converts the logic level "1" data at its inputs into an equivalent binary code at its output.

Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An "n-bit" binary encoder has 2^n input lines and n-bit output lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations. The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to "1" and are available to encode either a decimal or hexadecimal input pattern to typically a binary or B.C.D. output code.

4-to-2 Bit Binary Encoder



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

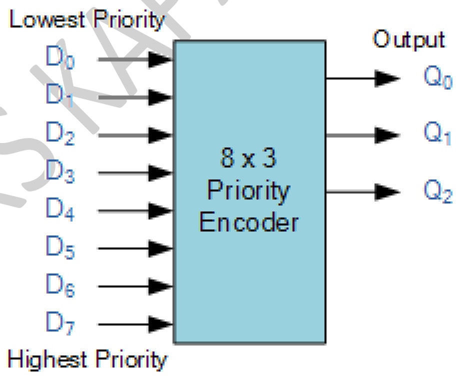
One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs $D_1$and D_2 HIGH at logic "1" both at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output binary number that is different to the actual input present. Also, an output code of all logic "0"s can be generated when all of its inputs are at "0" OR when input D_0 is equal to one.

One simple way to overcome this problem is to "Prioritise" the level of each input pin and if there was more than one input at logic level "1" the actual output code would only correspond to the input with the highest designated priority. Then this type of digital encoder is known commonly as a **Priority Encoder**or **P-encoder** for short.

Priority Encoder

The **Priority Encoder** solves the problems mentioned above by allocating a priority level to each input. The *priority encoders* output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored. The priority encoder comes in many different forms with an example of an 8-input priority encoder along with its truth table shown below.

8-to-3 Bit Priority Encoder

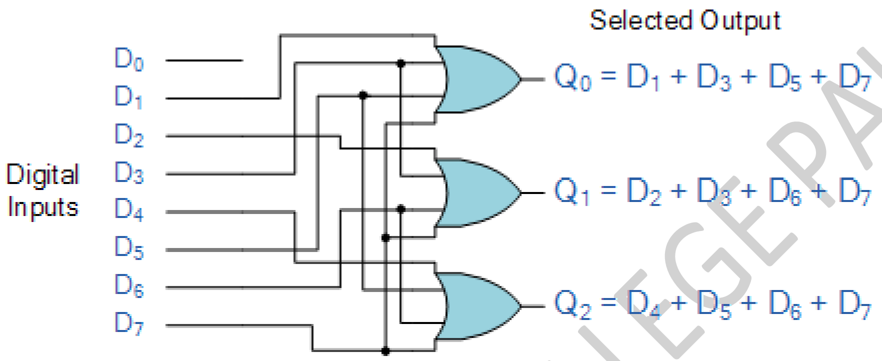


Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

Priority encoders are available in standard IC form and the TTL 74LS148 is an 8-to-3 bit priority encoder which has eight active LOW (logic "0") inputs and provides a 3-bit code of the highest ranked input at its output. Priority encoders output the highest order input first for example, if input lines "D2", "D3" and "D5" are applied simultaneously the output code would be for input "D5" ("101") as this has the highest order out of the 3 inputs. Once input "D5" had been removed the next highest output code would be for input "D3" ("011"), and so on.

Digital Encoder using Logic Gates


Selected Output
Digital Inputs
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$Q_0 = D_1 + D_3 + D_5 + D_7$
$Q_1 = D_2 + D_3 + D_6 + D_7$
$Q_2 = D_4 + D_5 + D_6 + D_7$

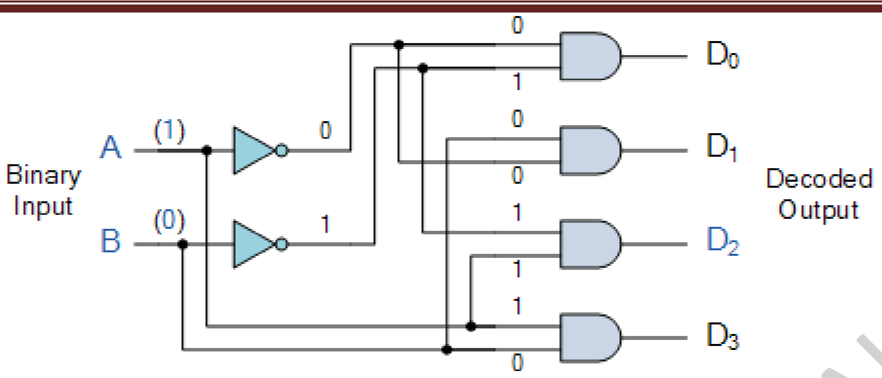
Binary Decoder

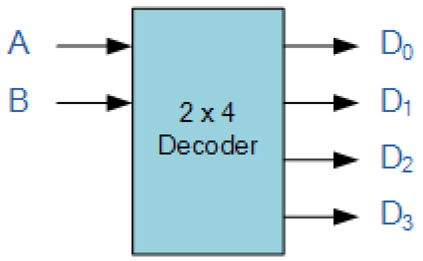
A **Decoder** is the exact opposite to that of an "Encoder" we looked at in the last tutorial. It is basically, a combinational type logic circuit that converts the binary code data at its input into one of a number of different output lines, one at a time producing an equivalent decimal code at its output. **Binary Decoders**have inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, and a n-bitdecoder has 2^n output lines.

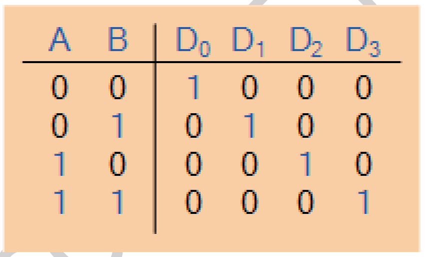
Therefore, if a binary decoder receives n inputs (usually grouped as a binary or Boolean number) it activates one and only one of its 2^n outputs based on that input with all other outputs deactivated. A decoders output code normally has more bits than its input code and practical "binary decoder" circuits include, 2-to-4, 3-to-8 and 4-to-16 line configurations.

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. An example of a 2-to-4 line decoder along with its truth table is given below. It consists of an array of four NAND gates, one of which is selected for each combination of the input signals A and B.

A 2-to-4 Binary Decoders.

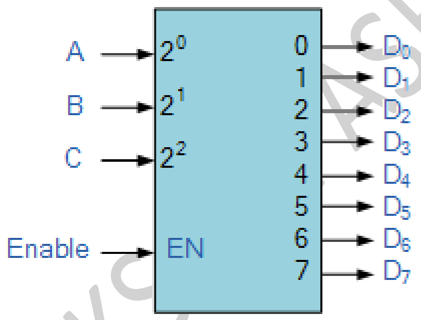






A	B	D$_0$	D$_1$	D$_2$	D$_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

In this simple example of a 2-to-4 line binary decoder, the binary inputs A and B determine which output line from D0 to D3 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 and these types of binary decoders are commonly used as **Address Decoders** in microprocessor memory applications.



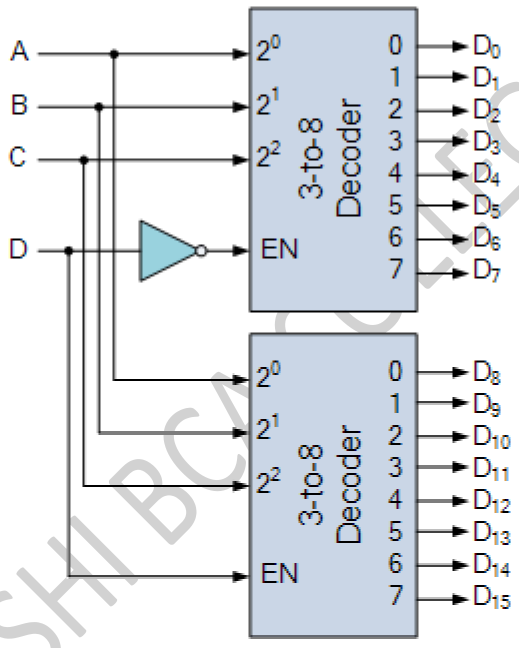
74LS138 Binary Decoder

Some binary decoders have an additional input labelled "Enable" that controls the outputs from the device. This allows the decoders outputs to be turned "ON" or "OFF" and we can see that the logic diagram of the basic decoder is identical to that of the basic demultiplexer.

Then, 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 address which can access a location having that address.

Sometimes it is required to have a **Binary Decoder** with a number of outputs greater than is available, or if we only have small devices available, we can combine multiple decoders together to form larger decoder networks as shown. Here a much larger 4-to-16 line binary decoder has been implemented using two smaller 3-to-8 decoders.

A 4-to-16 Binary Decoder Configuration.


A
B
C
D
2^0
2^1
2^2
EN
3-to-8 Decoder
0 1 2 3 4 5 6 7
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
D_8 D_9 D_{10} D_{11} D_{12} D_{13} D_{14} D_{15}

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 Dis used with the enable input to select which encoder either the first or second will output the "1".

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 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 to select the appropriate memory chip and a logic "1" on this input selects the device and a logic "0" on the input de-selects it. By selecting or de-selecting each chip, allows us to select the correct memory device for a particular address and when we specify a particular memory address, the corresponding memory location exists ONLY in one of the chips.

The Digital Comparator

Another common and very useful combinational logic circuit is that of the **Digital Comparator** circuit. 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 that has only one output terminal for when $A = B$ either "HIGH" $A = B = 1$ or "LOW" $A = B = 0$
-
- 2. Magnitude Comparator - a *Magnitude Comparator* is a type of digital comparator that has three output terminals, one each for equality, $A = B$ greater than, $A > B$ and less than $A < B$

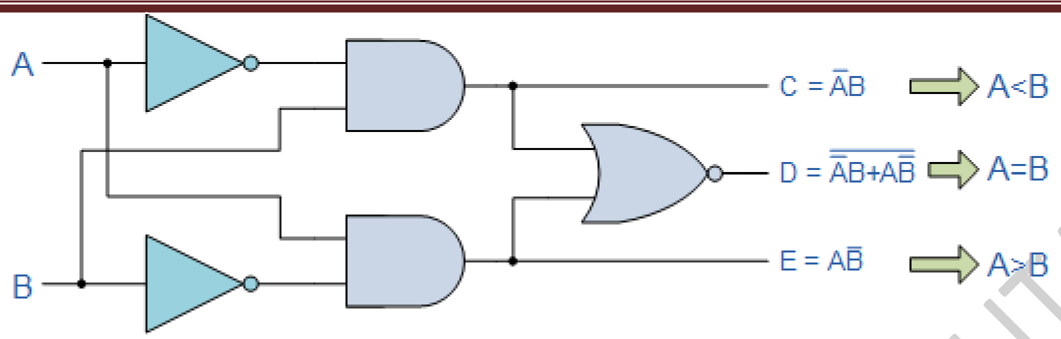
The purpose of a **Digital Comparator** is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, ... An, etc) against that of a constant or unknown value such as B (B1, B2, B3, ... Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For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

$$A > B, \quad A = B, \quad A < B$$

Which means: A is greater than B, A is equal to B, and 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



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

Truth Table

Inputs		Outputs		
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

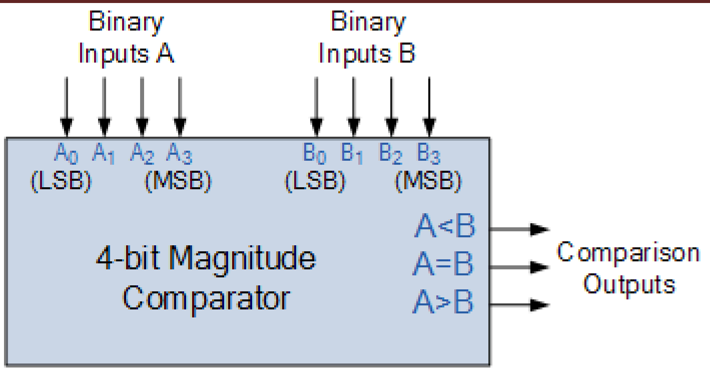
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = B resembles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

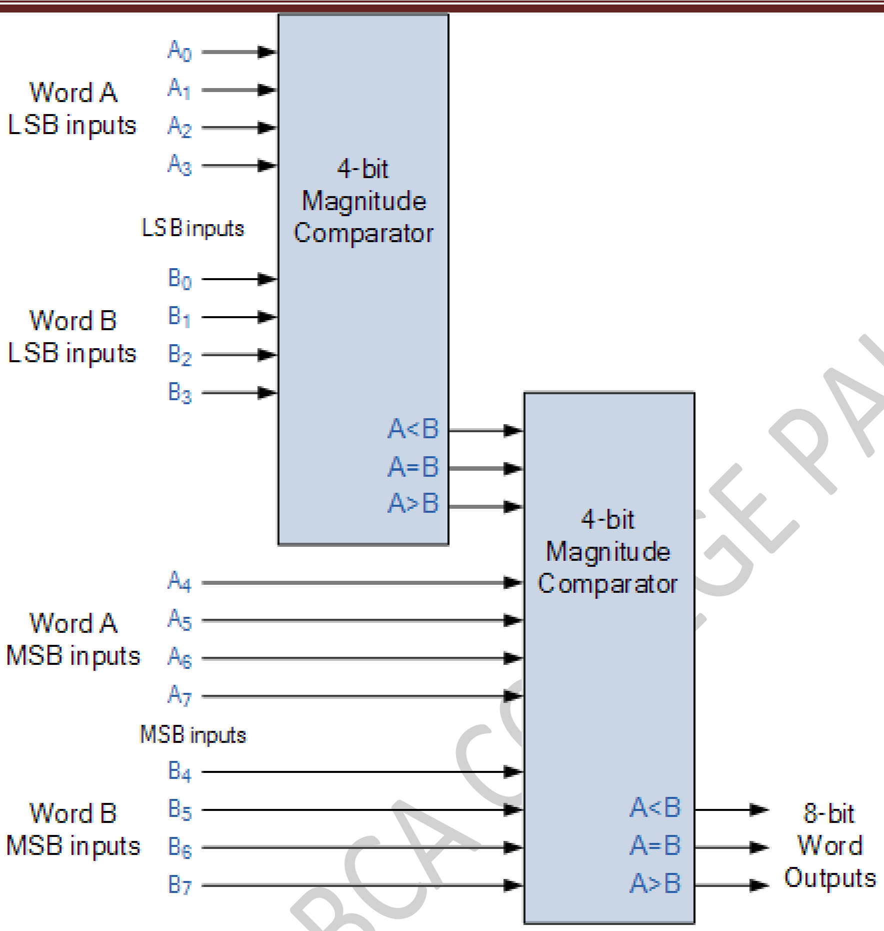
A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words ("nibbles") are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

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.