ElectronicsTutorials

COMBINATIONAL LOGIC

For Students, Professionals and Beyond

eBook 21

ASPENCORE

WWW.ELECTRONICS-TUTORIALS.WS

TABLE OF
CONTENTS

Our Terms of Use

This **Basic Electronics Tutorials eBook** is focused on combinational logic circuits with the information presented within this ebook provided “as-is” for general information purposes only.

This free e-book is presented as general information and study reference guide for the education of its readers who wish to learn Electronics. While every effort and reasonable care has been taken with respect to the accuracy of the information given herein, the author makes no representations or warranties of any kind, expressed or implied, about the completeness, accuracy, omission of errors, reliability, or suitability with respect to the information or related graphics contained within this e-book for any purpose.

As such it is provided for personal use only and is not intended to address your particular problem or requirement. Any reliance you place on such information is therefore strictly at your own risk. We can not and do not offer any specific technical advice, troubleshooting assistance or solutions to your individual needs.

We hope you find this guide useful and enlightening. For more information about any of the topics covered herein please visit our online website at:

www.electronics-tutorials.ws

1. Introduction

The operation of digital circuits is based around the various laws of *Boolean Algebra* using digital logic functions such as **AND**, **OR** and **NOT**. As such, these three logical functions are the basic building blocks of all digital circuits which themselves, can be combined or connected together to form much more complex switching circuits.

Each **AND**, **OR**, or **NOT** switching element can have one or more input variables but generally, only one single output. Thus the operation of each element or logic gate can be presented by a truth table listing all the possible combinations of input variables and its resulting output state as a direct result of its input conditions.

These three switching elements can therefore be classed as combinational devices, since their output state depends only on the current state of their inputs. Note that the input variables come from external sources, while the corresponding output state is produced internally by the combinational logic circuit itself.

Then a **Combinational Logic Circuits** output is dependant at all times on the combination of its inputs meaning that it is *memoryless*. That is, it has no knowledge of any previous events.

> *Combinational logic circuits output is dependent only on the state of its actual inputs since it has no feedback loops*

So if the condition of an input changes state, from 0-1 or 1-0, so too will the resulting output since by default, combinational circuits have no memory or feedback loops within their design.

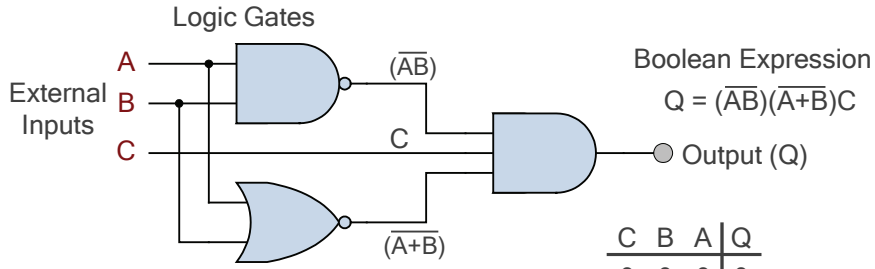
Thus combinational (or non-regenerative) logic circuits have the property that at any point in time, their output is directly related to their current input condition following some Boolean expression since they use digital logic gates.

By combining together different digital logic gates, we can create an endless number of possible combinations depending on the application. As a result, *combinational logic circuits* can be very simple or very complicated circuits with the three main ways of specifying the function of any combinational logic circuit given as follows:

1. **Boolean Algebra –** This forms the algebraic expression showing the operation of the combinational circuit for each input variable either true or false which will result in a logical-1 (HIGH) output condition.
2. **Truth Table –** A truth table defines the function of the circuit by providing a concise list that shows all the output states in tabular form for each possible combination of input variable which could be encountered.
3. **Logic Diagram –** This is a graphical representation of a logic circuit which shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.

All three of these logic circuit representations are shown in Figure 1.

Figure 1. Combinational Logic Circuit



Typical Truth Table

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

Since combinational logic circuits are made up from individual logic gates only, they can also be considered as being “decision making circuits”. Combinational logic is about combining logic gates together to process two or more input signals in order to produce at least one output state according to the logical function of each logic device.

Common combinational circuits made up from individual logic gates that carry out a desired application can include: multiplexers (selectors) and de-multiplexers (distributors), encoders and decoders, adders and subtractors, and/or comparators, etc.

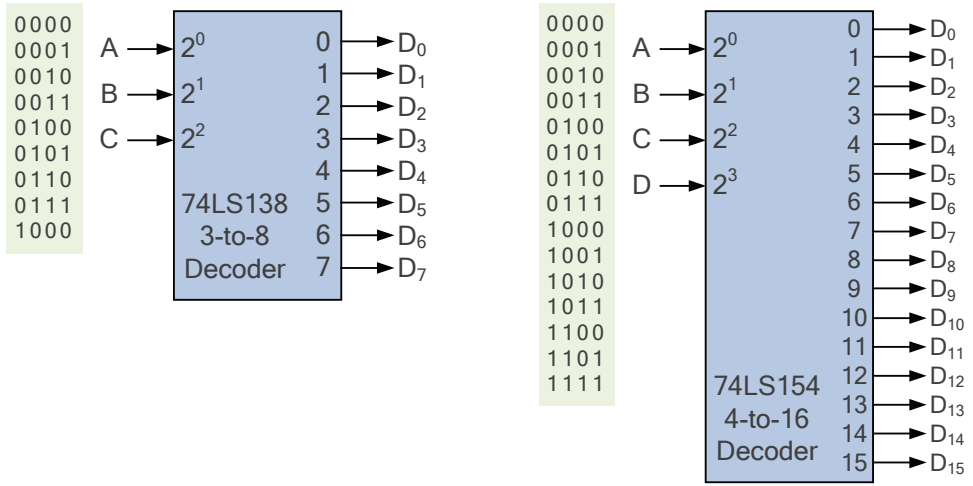
2. The Binary Decoder

Decoding is the process of converting some form of digital code, where the input and output codes are different, such as Binary, BCD, or Hexadecimal into a singular active output representing its numeric value. Then, decoders translate or decode digital information from one format into another based upon a given combination of actual inputs and as such, are a commonly used combinational logic circuit.

The basic **Binary Decoder** is nothing more than a simple combinational circuit made using standard **AND**, **OR** and **NOT** gates to convert binary information from *n-bit* input lines to a maximum of 2^n individual output lines, one for each element of information.

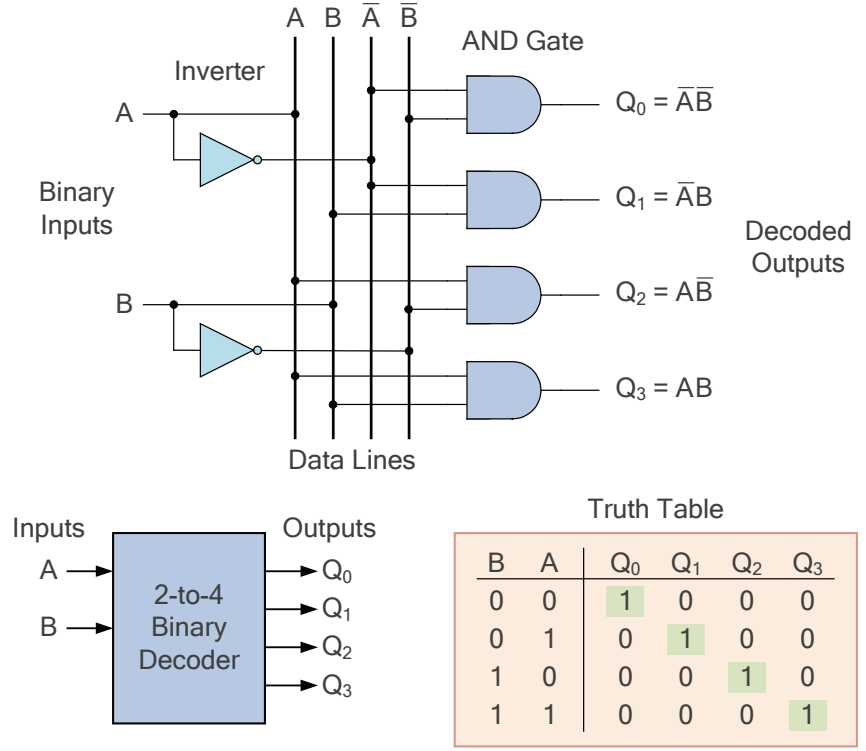
For example, a **74LS138**, 3-to-8-line (3-to-2^3) decoder will have three inputs and eight output lines. While the **74LS154**, 4-to-16-line (4-to-2^4) decoder will have four inputs and sixteen individual output lines, etc. as shown in Figure 2.

Figure 2. A 3-to-8 and 4-to-16 Decoder Block Diagram



So if a 2-to-4-line decoder has two inputs, it will choose or select one of its four (2^2) outputs depending upon which input combination is present. An example of a 2-to-4-line decoder along with its truth table is given in Figure 3.

Figure 3. A 2-to-4 Binary Decoder



Truth Table

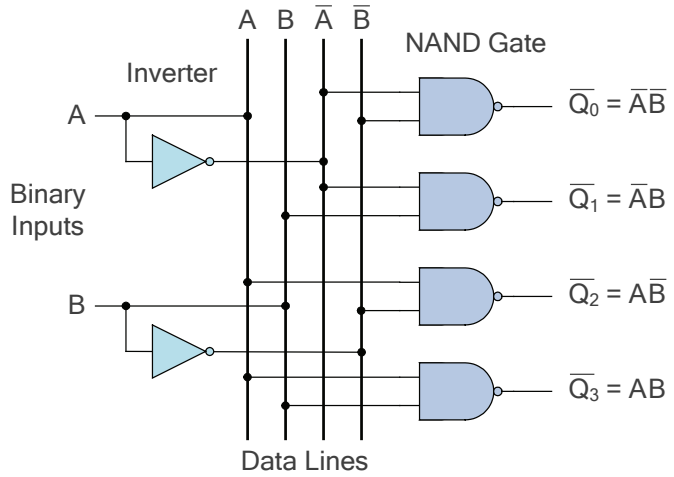
B	A	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Figure 3. shows that a basic binary decoder can be easily implemented using four 2-input **AND** gates and two **NOT** gates. If the input is **00** (A = 0 and B = 0), output Q_0 would be present. If the input is **01** (A = 0 and B = 1), output Q_1 would be present and so on.

Thus, for any combination of inputs, only one output will be HIGH (assuming positive logic) at any one time while all the other outputs will be LOW (logic level-0).

Since NAND gates are cheaper to make than AND gates (require fewer transistors to implement), NAND based decoders can also be constructed which produce a LOW output that corresponds to the actual input condition while all the other outputs will be HIGH as shown in Figure 4.

FIGURE 4. A NAND 2-TO-4 BINARY DECODER



Truth Table

B	A	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

The NAND gate decoder's outputs are classed as active-low and are commonly used for microprocessor address decoding since most memory and microprocessor peripheral chips use active-low enable signals.

Then as we have seen, a digital decoder selects only one output at a time and are used in applications such as data multiplexing, display and memory address decoding.

2.1 BCD to 7-segment Display Decoders

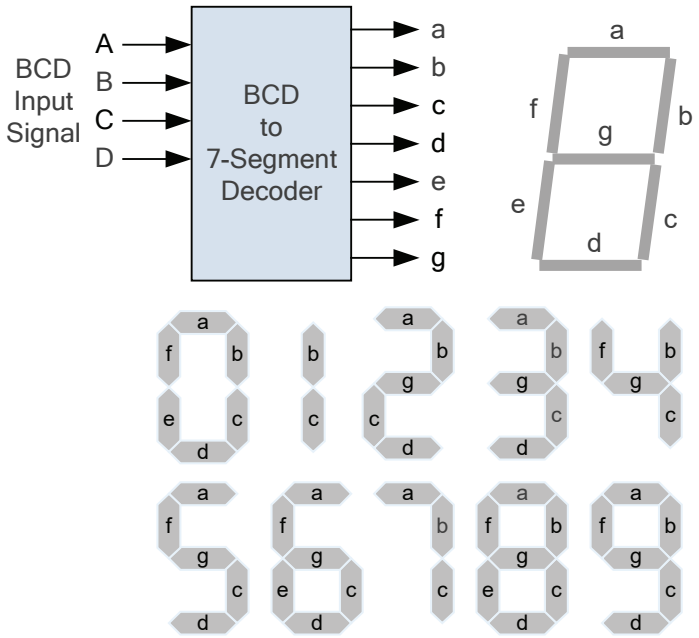
7-segment displays can be used for displaying a single binary digit, or connected together to display whole numbers. An LED display consists of seven *light emitting diode* (LED) segments producing each of the 10 decimal digits, 0 through 9. Depending on the type of display, common-cathode or common-anode, each segment (**a** through **g**) can be activated by either a HIGH or LOW voltage level.

If we want to display these numbers using a seven-segment display, we need a decoder which will light up the appropriate segments of the display. The input to the BCD to 7-segment decoder uses 4-bits in a *binary-coded-decimal* (BCD) format, where decimal 0 is represented by the four binary functions **ABCD = 0000**. Decimal 1 is represented by **ABCD = 0001**, 2 by **0010**, 3 by **0011**, and so on through to 9 = **1001**. Note that other decimal codes are also possible.

Thus BCD to 7-segment decoders such as the TTL **74LS47** or **74LS48**, have 4 BCD inputs and 7 output lines, one for each LED segment. The **74LS47** decoder/driver for driving common-anode displays. While its cousin, the **74LS48** is used for driving common-cathode displays.

Figure 5 shows a block diagram of a typical decoder, the display element, and the way the segments should be driven to represent the appropriate digit along with the truth table.

FIGURE 5. BCD TO 7-SEGMENT DISPLAY DECODER



Truth Table

D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

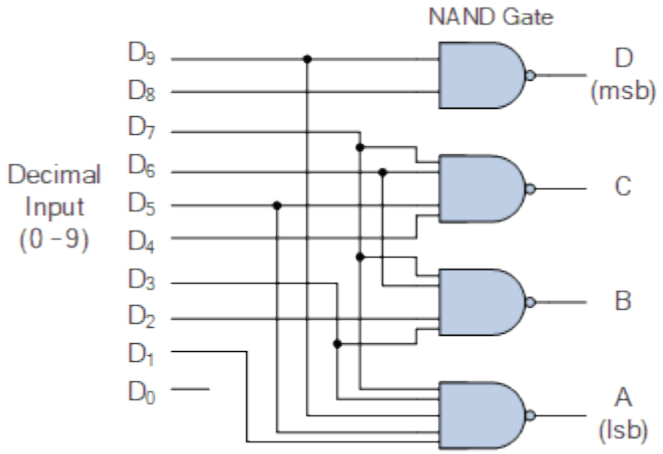
3. The Binary Encoder

Encoding is the opposite process of decoding. An *encoder* is a combinational circuit used to generate a coded output (such as BCD, binary or HEX) from a singular active numerical input line. That is, a **Binary Encoder** takes the logical state of ALL of its input lines and converts them into a single encoded representation of these inputs on its output lines.

Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes, etc. depending upon the number of data input lines to be converted. So, an "n-bit" binary encoder will have 2^n input lines resulting in *n-bit* output lines. Common types of digital encoders include 4-to-2, 8-to-3 and 16-to-4 line configurations.

For example, a 10-to-4 decimal to binary encoder of Figure 6. takes a 10-digit (0 to 9) input combination and outputs a 4-bit code representing the bit state of the input signal line which is HIGH (assuming positive logic).

Figure 6. 4-bit Decimal to Binary Encoder



Decimal Input	BCD Output D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

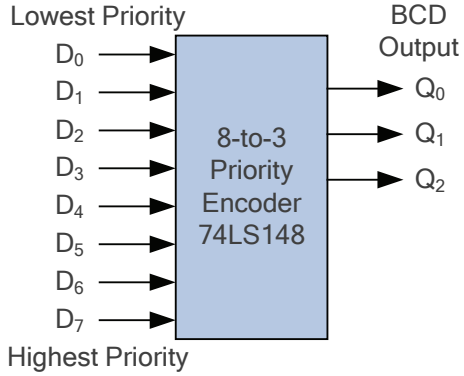
One of the main disadvantages of standard digital encoders is that only one input can be active at any given time, or else it will generate the wrong output code when there is two or more logic level-1 inputs present. For example, if we make inputs D_1 and D_2 HIGH (logic-1) simultaneously, the resulting output from the encoder is neither at **01** or at **10**, but instead will be at **11** which does not represent either binary 1 or binary 2. Also, an output code of all logic level **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. So if there is more than one input at logic level-1 at the same time, 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*.

3.1 Priority Encoders

The TTL **74LS148** is a 8-to-3 line decimal to binary **Priority Encoder** which means that if two or more inputs are present, the highest numeric input will have priority and be encoded to the output. In other words, it produces a BCD output according to highest-order decimal digit present on the input as shown in Figure 7.

Figure 7. 3-bit Decimal to Binary Priority Encoder



Decimal Inputs D7	D6	D5	D4	D3	D2	D1	D0	Outputs Q2	Q1	Q0
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

Thus, if input lines D_2, D_3 and D_5 are applied simultaneously the output code would be for input D_5 (**101**), since this has the highest numerical value from the 3 inputs. Once input D_5 had been removed the next highest output code would be for input D_3 (**011**), and so on until one of the inputs between D_4 to D_7 takes priority.

Priority encoders such as the **74HC923**, is a 20-key encoder for use in microprocessor, micro-controller and computer systems to handle keyboard and peripheral interrupt signals responding to the highest priority pending interrupt request (IRQ). *Positional Encoders* are another form of priority encoder used for the positional control of robotic arms or steering mechanisms, etc.

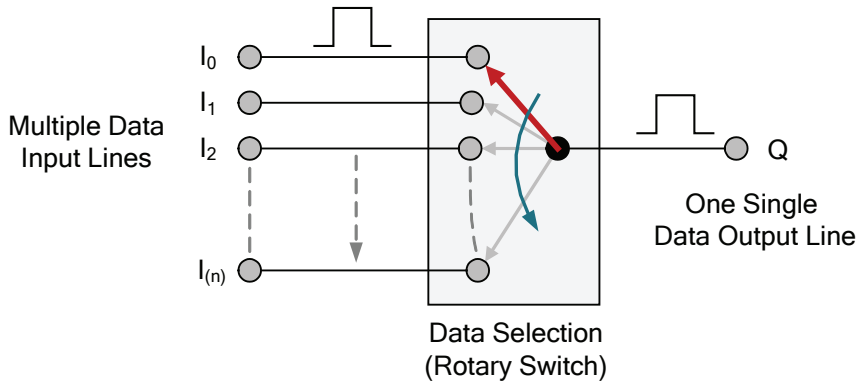
4. The Multiplexer

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. The electronic device used to do this is called a **Multiplexer**, or MUX.

The *multiplexer* is a combinational logic based fast acting multiple position rotary switch connecting or controlling multiple input lines called channels. It is designed to select one of several input lines, one at a time, and channel the data through to a single common output line. That is, a multiplexer may have 2^n data input lines, but only one output.

The most basic example of multiplexing is that of a one-way rotary switch as shown in Figure 8.

Figure 8. Basic Multiplexing Switch

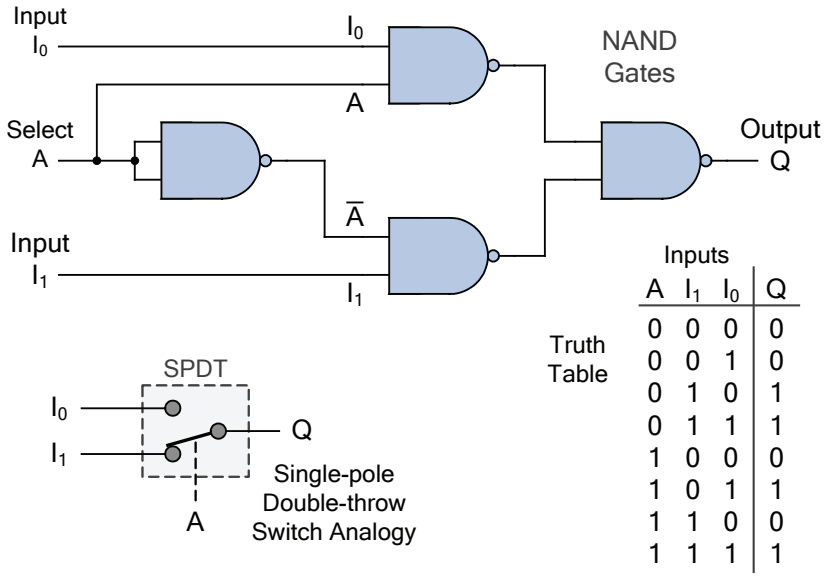


In the example of Figure 8. the rotary switch can select any one of the individual data or signal lines from I_0 to $I_{(n)}$ connecting the input directly to the output.

In digital electronics, multiplexers are also referred to as *Data Selectors* because they can "select" each input line. They are commonly constructed from individual MOSFET switches encased in a single IC package.

Generally, the selection of each input line in a multiplexer is controlled by an additional set of inputs called *control* or *select lines* to determine which input data line should be connected directly to the output. We can build a simple 2-line to 1-line (2-to-1) multiplexer using basic logic **NAND** gates as shown in Figure 9.

Figure 9. Basic 2-input Multiplexer Using NAND Gates



Inputs			
A	I_1	I_0	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Input **A** of this simple 2-1 line multiplexer circuit is used to control which input (I_0 or I_1) gets passed to the output at **Q**.

The truth table of Figure 9, shows that when the data select input, **A** is LOW, input I_1 is connected to the output, while input I_0 is blocked. When the data select **A** is HIGH, the reverse happens and now input I_0 is connected to the output while input I_1 is blocked.

So by the application of either a logic-0 or a logic-1 at **A**, we can select the appropriate input, I_0 or I_1 with the circuit acting a bit like a single-pole double-throw (SPDT) switch.

Since we only have only one control line (A), we can only switch 2^1 inputs to the common output. Thus producing a 2-to-1-line multiplexer and we can confirm this using the following Boolean expression.

$$Q = \overline{A}.I_1 + A.I_0$$

If we increase the number of data inputs to be selected by simply following the same procedure, larger multiplexer circuits can be implemented using a 2-to-1 multiplexer as their basic building blocks. So for example, a 4-input multiplexer we would require two data select lines as 4-inputs represents 2^2 data control lines give a circuit with four inputs, I_0, I_1, I_2, I_3 and two data select lines A and B as shown in Figure 10.

FIGURE 10. 4-CHANNEL MULTIPLEXER CIRCUIT

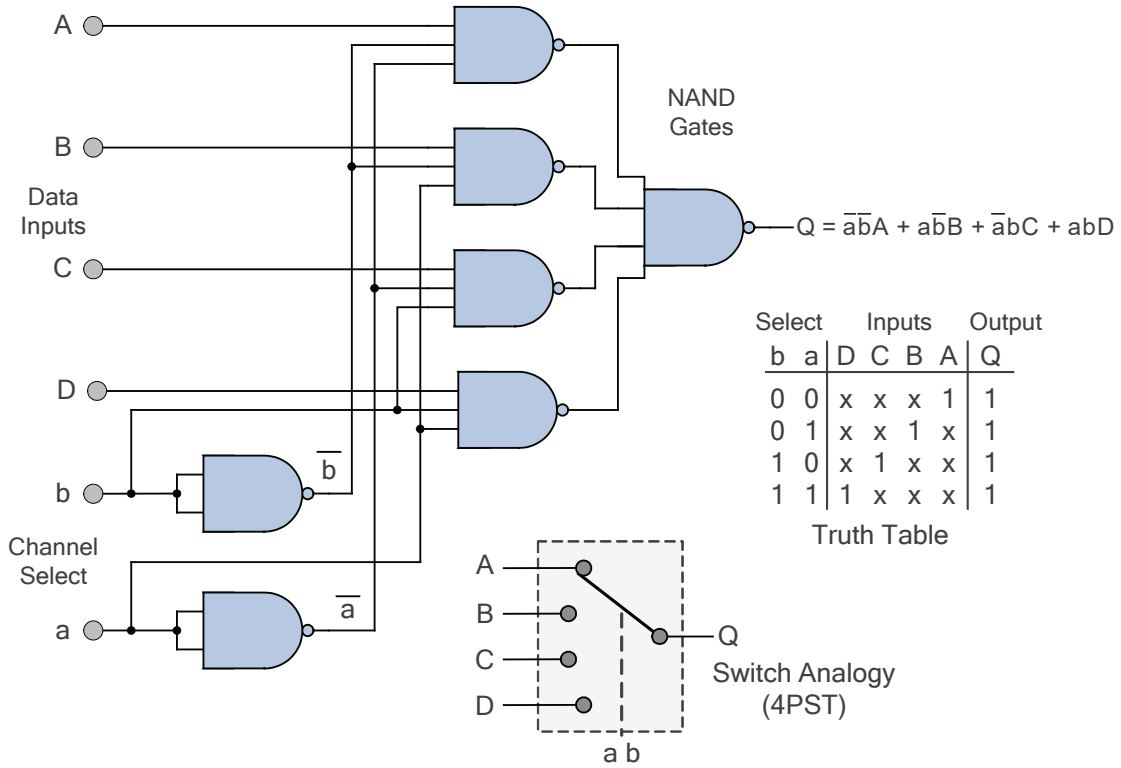
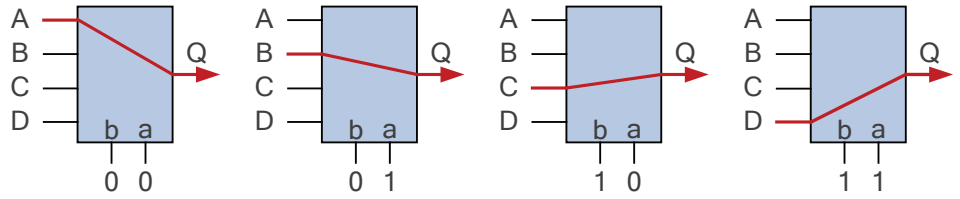


Figure 10. shows that at any one instant in time only ONE of the four input lines A to D is connected to the single output at Q. As to which switch is closed depends upon the addressing input code on lines "a" and "b". So for this example to select input B to the output at Q, the binary input address would need to be: select **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 given in Figure 11.

FIGURE 11. MULTIPLEXER INPUT LINE SELECTION



Clearly, adding more control address lines, (**n**) will allow the multiplexer to control more inputs such as the TTL **74LS151** 8-to-1 or the TTL **74LS153** Dual 4-to-1 line multiplexer. But remember, each control line configuration will only connect ONE input to the output.

Multiplexers are not just limited to switching 2^n 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 line configurations.

5. THE DEMULTIPLEXER

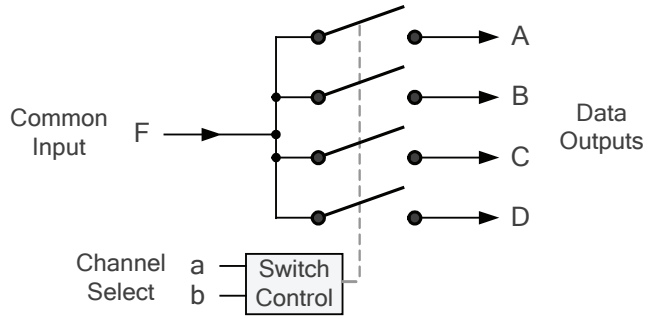
The **Demultiplexer**, or DeMUX is another combinational logic based multiple positional switch used to route analogue or digital signals, and is by design, the exact opposite of the previous multiplexer circuit.

The de*multiplexer*, also known as a *Data Distributor* takes one single input data line and switches it to one of a number of individual output lines one at any one time, in accordance with the binary number applied to the channel select lines.

That is, a demultiplexer connects a single input line to one of 2^n output lines , depending on the values of **n** select lines.

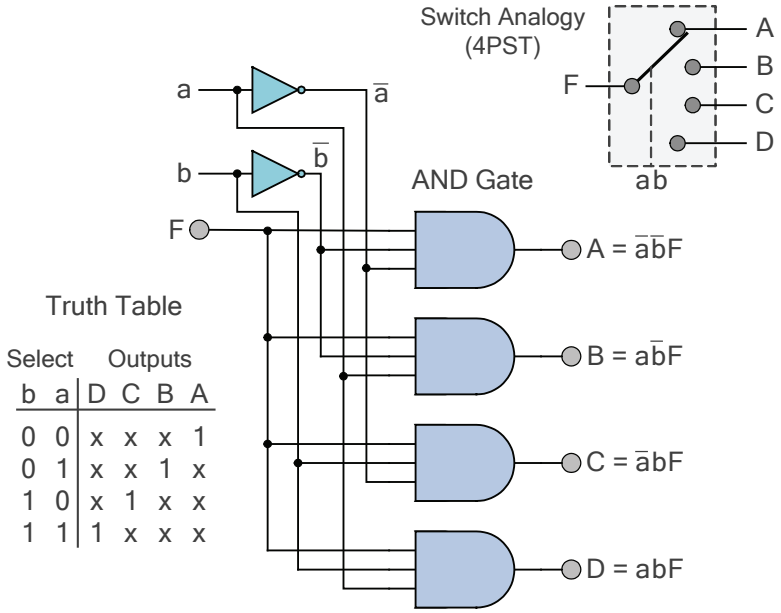
The most basic example of demultiplexing is that of the switching circuit in Figure 12.

FIGURE 12. BASIC DEMULTIPLEXER SWITCH



The implementation of the of the above circuit can be achieved using individual **AND** and **NOT** logic gates as shown in Figure 13.

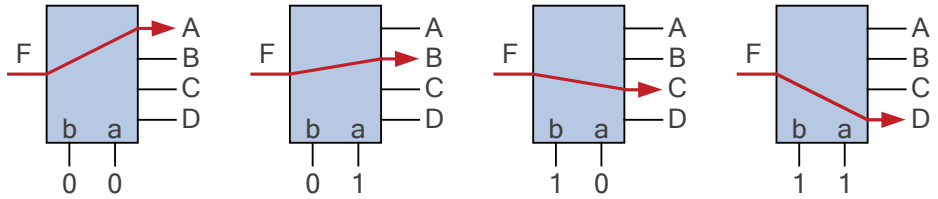
FIGURE 13. 4-CHANNEL DEMULTIPLEXER CIRCUIT



Note, if input **F** is permanently connected to a logic level-1, can be used as a decoder.

As with the previous multiplexer, the individual solid state switches are selected by the binary input address code on the output select pins **a** and **b** as shown in Figure 14.

FIGURE 14. DEMULTIPLEXER OUTPUT LINE SELECTION



Standard Demultiplexer IC packages available are the TTL **74LS138** 1-to-8 demultiplexer, the TTL **74LS139** Dual 1-to-4 demultiplexer or the CMOS **CD4514** 1-to-16 demultiplexer.

Another type of demultiplexer is the 24-pin, **74LS154** which is a 4-bit to 16-line demultiplexer and decoder. Here the individual output positions are selected using a 4-bit binary coded input. Like multiplexers, they can also be cascaded together to form higher order devices and by adding more channel select inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs.

6. THE BINARY HALF-ADDER

Combinational logic circuits can also be configured to perform arithmetic and logical operations such as **Binary Addition** which can be implemented using basic logic gates.

In conventional mathematics, two or more decimal or denary (base-10) numbers can be added together to give their sum value. For example, consider the addition of the two numbers in Figure 15.

FIGURE 15. ADDITION OF TWO DENARY NUMBERS

123	A	(Augend)
+ 456	B	(Addend)
579	SUM	

Here, number **A** (the augend) value of 123_{10} is added to 456_{10} (the addend) value of number **B** to produce the **SUM** value of 579_{10}. We add together starting from the right hand side knowing that each digit has a weighted value depending upon its position within the columns.

A **Carry** or *carry-over* number is generated if the addition of a column results in a number equal to, or greater than 10 (the Base number). This carry number is automatically added to the numbers within next column. Then the addition of denary numbers is simply, add the numbers and produce a carry.

The adding of binary numbers is exactly the same idea as that for adding together two or more decimal (denary) numbers except in binary there are only two digits with the largest digit being **1**. When adding binary numbers, a carry or carry-over is generated when the **SUM** value of any column is equal to, or is greater than 2, (the base number of binary).

The addition of two base-2 digits is relatively simple since the addition of: **0 + 0 = 0**, **0 + 1 = 1**, **1 + 0 = 1**, resulting in either a **0** or a **1** until we get to the addition of **1 + 1** in which the sum is equal to **10** (1,0 NOT TEN). That is: **1 + 1 = 10**, which is a base-2 ($10_2 = 2_{10}$) value.

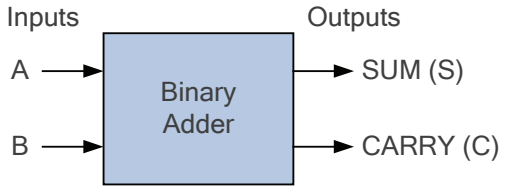
But the number two does not exist in binary only **0** and **1**. However, **2** in binary is equal to **10**, a zero for the sum plus an extra carry bit. In other words, **1 + 1** creates a "**CARRY**" as shown in Figure 16.

FIGURE 16. BINARY ADDITION OF TWO BITS

0	0	1	1
+0	+1	+0	+1
0	1	1	(carry) 1 ← 0

Then the operation of a simple adder requires two data inputs producing two individual outputs, the **Sum** (S) of the equation and a **Carry** (C) bit as shown in Figure 17.

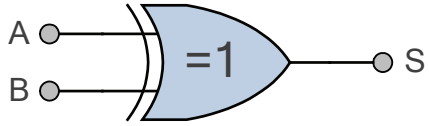
FIGURE 17. BINARY ADDER BLOCK DIAGRAM



Note that for the simple 1-bit addition problem above, the resulting carry bit could be ignored if not needed.

But you may have noticed something else with regards to the addition of these two bits, the resulting sum of their binary addition **0**, **1**, **1**, **0**, matches the output state of an *Exclusive-OR* gate as shown in Figure 18.

FIGURE 18. 2-INPUT EXCLUSIVE-OR GATE AND TRUTH TABLE



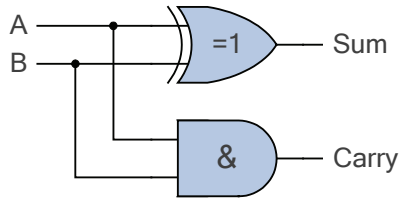
2-input Ex-OR Gate Symbol

Truth Table		
B	A	S
0	0	0
0	1	1
1	0	1
1	1	0

An **Exclusive-OR** gate only produces a logic-1 output when **EITHER** inputs are at logic-1. Then we need an additional output to produce the required CARRY bit when **BOTH** inputs **A** and **B** are at logic-1. One digital logic gate that fits the bill perfectly producing a logic-1 output when both of its inputs are HIGH (**1**) is the standard **AND** Gate.

By combining the **Exclusive-OR** gate with an **AND** gate results in a simple digital binary adder circuit known commonly as a **Half Adder** circuit as shown in Figure 19.

FIGURE 19. 2-INPUT HALF-ADDER AND TRUTH TABLE



Truth Table			
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 in Figure 19. we can see that the **SUM** (S) output is the result of the **Exclusive-OR** gate and that the **CARRY** (C) is the result of the **AND** gate. Then the Boolean expression for the Half Adder circuit of Figure 19. is given as:

For the **SUM** bit: $SUM\ (S) = A\ XOR\ B = A \oplus B$

For the **CARRY** bit: $CARRY\ (C) = A\ AND\ B = A.B$

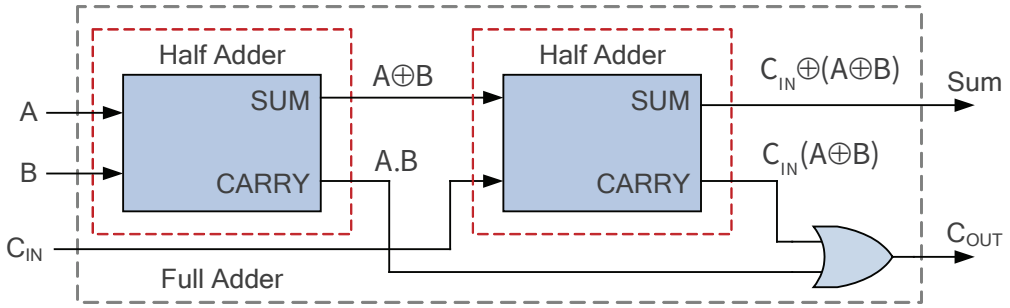
One major disadvantage of the half-adder circuit when used for binary addition, is that while it can produce a *Carry-out* (C-out) bit. There is no provision for a *Carry-in* from a previous circuit when adding together multiple data bits. One simple way to overcome this problem is to use what is called a *Full Adder* for binary addition.

7. The Binary Full-Adder

The **Full Adder** is a combinational logical circuit that performs binary addition on its inputs. 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 bit from a previous stage, and just like the half-adder, it produces a carry-out bit to the next addition column.

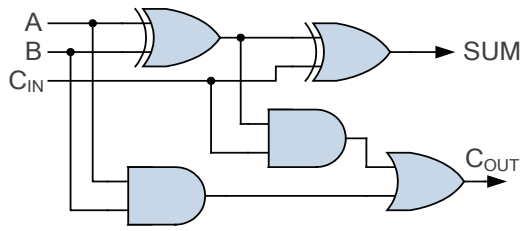
Then in many ways, the full-adder circuit can be thought of as two half-adders connected together, with the first half-adder passing its carry to the second half adder as shown in Figure 20.

Figure 20. Full-Adder Representation



Since the full-adder circuit of Figure 20. is basically two half-adders connected together, the truth table for the full-adder includes an additional column to take into account the Carry-in input as well as the SUM output, (S) and the Carry-out bit as shown in Figure 21.

Figure 21. Full-Adder and Truth Table



Truth Table				
Carry-In	B	A	SUM	Carry-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

Then the Boolean expression for a Full-adder of Figure 21 is given as follows:

For the **SUM** bit: $SUM = C_{IN}\ XOR\ (A\ XOR\ B) = C_{IN} \oplus A \oplus B$

For the **CARRY-OUT** bit: $C_{OUT} = C_{IN}(A\ XOR\ B)\ OR\ (A\ AND\ B) = C_{IN}(A \oplus B) + A.B$

Then, **Binary Adders** are combinational logic circuits which can be used to "add" together two binary numbers producing a **SUM** and **Carry-out** bits. 4-bit full-adder circuits with carry look ahead features such as the TTL 4-bit binary adder **74LS83** or the **74LS283** are available as standard IC packages to add together two 4-bit binary numbers and generate the required **SUM** and a **CARRY** bits.

8. The Binary Half-Subtractor

If we can use combinational logic circuits to "add" together two or more binary digits, we must also be able to "subtract" two or more binary digits, and we can. **Binary Subtractors** are another decision making combinational logic circuit which can subtract two (or more) binary numbers from each other. For example, A – B.

Whereas before the binary adder found the sum of two numbers, the binary subtractor will produce the difference between the two numbers. Similar to the previous adder circuit, the subtractor also produces two outputs. One called the **DIFFERENCE** and the other called the **BORROW** (carry in the case of the adder). There are commonly two types of binary subtractor. The **Half Subtractor** and the **Full Subtractor**.

Binary Subtraction can take many forms but the rules for subtraction are the same whichever process you use. As binary notation only has two digits, subtracting a "0" from a "0" or a "0" from a "1", leaves the result unchanged as 0 - 0 = 0 and 1 - 0 = 1. However, subtracting a 1 from a 1 would result in a 0, but subtracting a 1 from a 0 would require a **BORROW** bit as shown in Figure 22.

Figure 22. Binary Subtraction of Two Bits

0	1	1	(borrow) 1	→1 0
-0	-0	-1		-1
0	1	0		1

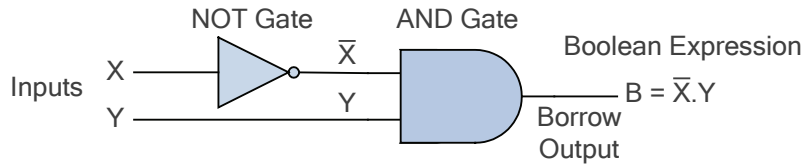
Again, the operation of a simple subtractor requires two data inputs and two individual outputs, the **Difference** (D) of the equation and a **Borrow** (B) bit. Note that to prevent any confusion between a binary subtractor input labelled, B and the resulting borrow bit output also being labelled, B. We shall now refer to the two data input bits as being X for the minuend, and Y for the subtrahend.

Since the difference between the two input bits is only a logic-1 when the two inputs are not equal, and a logic-0 when equal. This gives us again the expression for our familiar **Exclusive-OR Gate** which we can use for the **DIFFERENCE** output bit.

However, we need an additional output to produce the required borrow bit when the inputs are: X = 0 and Y = 1. Unfortunately there is no standard logic gate which could produce an output for this particular combination of X and Y inputs.

But we know that a logic **AND** gate produces a logical-1 output when both of its inputs are HIGH (1). So if we use an inverter or **NOT** gate to complement input X before it is fed to the **AND** gate. Then we could actually produce the required borrow output with the input conditions of X = 0 and Y = 1 as shown in Figure 23.

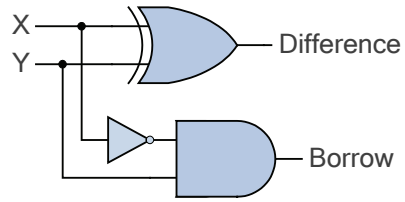
Figure 23. Borrow Bit Creation



So by combining an **Ex-OR** gate to generate the Difference, with a **NOT-AND** combination to generate the Borrow results in a simple digital binary subtractor circuit known commonly as the **Half Subtractor** (HS), as shown in Figure 24.

Figure 24. 2-input Half-subtractor and Truth Table

Symbol



Truth Table

Y	X	DIFFERENCE	BORROW
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	0

We can see from the truth table of the half subtractor circuit of Figure 24. that the **DIFFERENCE** (D) output is the result of the Exclusive-OR gate and the **BORROW** (B) is the result of the **NOT-AND** combination.

Then the Boolean expression for a half subtractor is given as follows.

For the **DIFFERENCE** bit: DIFFERENCE (D) = X XOR Y = $X \oplus Y$

For the **BORROW** bit: BORROW (B) = not-X AND Y = $\overline{X}.Y$

Note that if we compare the Boolean expressions of the half subtractor with that of the half adder, we can see that the two expressions for the **SUM** (adder) and **DIFFERENCE** (subtractor) are exactly the same because of the *Exclusive-OR* gate function.

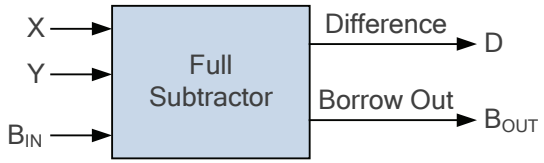
Also, the two Boolean expressions for the binary subtractors **BORROW** bit is very similar to that for the adders **CARRY**. Then all that is needed to convert a half adder to a half subtractor is the inversion of the minuend input **X**. That is, **X-Y** = **X+**(2's complement of **Y**).

As before, one major disadvantage of the Half Subtractor circuit when used as a binary subtractor, is that there is no provision for a "Borrow-in" from a previous circuit. Then we need to produce what is called a Full Binary Subtractor circuit.

9. The Binary Full-Subtractor

A **Full Subtractor** circuit has three inputs. The two single bit data inputs **X** (minuend) and **Y** (subtrahend) the same as before plus an additional **Borrow-in** (B-in) input to receive the borrow generated by the subtraction process from a previous stage as shown in Figure 25.

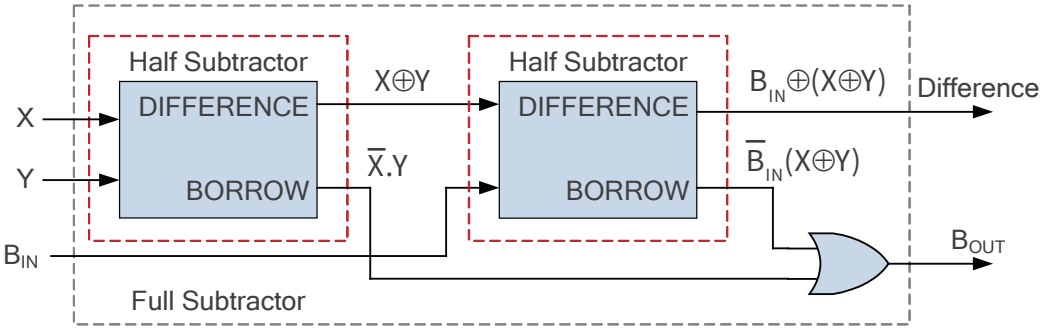
Figure 25. Binary Subtractor Block Diagram



Thus, the circuit of a full subtractor (FS) performs the operation of subtraction on three binary bits producing outputs for the difference D and borrow B-out.

Just like the binary adder circuit, the full subtractor can also be thought of as two half subtractors connected together, with the first half subtractor passing its borrow to the second half subtractor as shown in Figure 26.

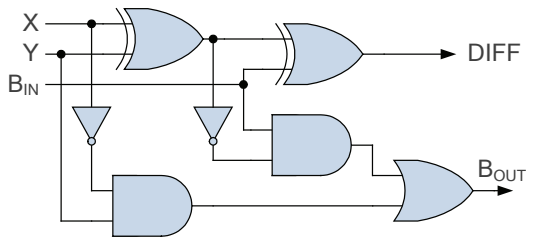
Figure 26. Full-Subtractor Representation



Since the full subtractor circuit represents two half-subtractors cascaded together, the truth table for a full subtractor will have eight different input combinations. As there are three input variables, two data bits and the Borrow-in bit as shown in Figure 27.

Figure 27. Full-subtractor and Truth Table

Symbol	Truth Table				
	Borrow-In	Y	X	DIFF.	Borrow-Out
	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	1
	0	1	1	0	0
	1	0	0	1	1
	1	0	1	0	0
	1	1	0	0	1
	1	1	1	1	1



Then we can see that the full subtractor circuit of Figure 27. subtracts the two input bits while taking into account the borrow bit.

Then the Boolean expression for a full subtractor is as follows.

For the **DIFFERENCE** bit: $\text{DIFFERENCE (D)} = B_{IN} \text{ XOR } (X \text{ XOR } Y) = B_{IN} \oplus (X \oplus Y)$

For the **BORROW** bit: $\text{BORROW (B)} = \text{not-X AND Y OR } B_{IN}.\text{not-}(X \text{ XOR } Y)$

$$= \overline{X}.Y + B_{IN}(\overline{X \oplus Y})$$

As with the binary adder, we can also have **n** number of 1-bit full binary subtractors cascaded together to subtract two parallel n-bit numbers from each other. For example two 4-bit binary numbers.

Since the only difference between a full adder and a full subtractor is the 1's complement inversion of one of its data inputs. We could use a 4-bit full-adder IC such as the **74LS283** to perform subtraction since **X – Y** is basically the same as saying, **X + (-Y)** which is effectively **X** plus the two's complement of **Y**.

10. The Digital Comparator

If we can both add and subtract two binary numbers, we must also be able to compare them to see if they are equal. For example, is the value of input **A** greater than, smaller than, or equal to the value at input **B**, and we can do this using a digital comparator.

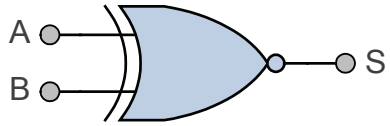
The **Digital Comparator** is another arithmetic logic circuit that uses standard **AND**, **NOR** and **NOT** gates to compare the digital signals present at its input terminals, producing an output result depending upon the condition of those inputs.

There are two main types of Digital Comparator available we can use 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 basic identity comparator evaluates two binary bits and outputs a 1 (HIGH) if they are exactly equal. The easiest way to compare the equality of two binary bits is to use an **Exclusive-NOR** (Ex-NOR) gate of Figure 28.

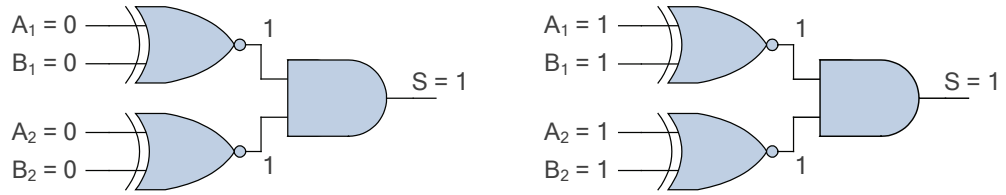
Figure 28. Exclusive-NOR Gate



Thus according to an Exclusive-NOR gates truth table, if both input bits are equal, either **0 - 0** or **1 - 1**, then it outputs a logic-1.

So if we wanted to compare two (or more) data bits, we would need additional Ex-NOR gates, with the output of all of them must be **1**. For example, to design a comparator to evaluate two data sets, A_1 B_1 with A_2 B_2, we can connect the two Ex-NOR outputs into a 2-input **AND** gate as shown in Figure 29.

Figure 29. Basic 2-bit Identity Comparator

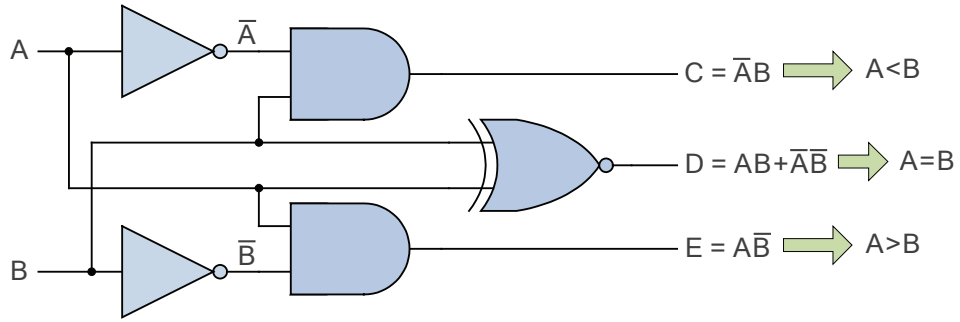


Thus if all four inputs (A_1, B_1, A_2, B_2) are either at logic **0** or logic **1**, then the Ex-NOR outputs are 1's, and the AND gate output is a **1** (HIGH), indicating equality. That is $A_1 = A_2$ and $B_1 = B_2$. Any other combination of inputs would result in the **AND** gate outputting a **0** (LOW).

The **Magnitude Comparator** not only produces an output if **A** equals **B** but can produce two more outputs if **A** is greater than **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 met.

Thus, a simple 2-bit comparator which has two inputs (**A**, **B**) will have three outputs corresponding to: **A<B**, **A=B**, or **A>B** as shown in Figure 30.

Figure 30. Basic 2-bit Magnitude Comparator



Then the logical operation of a 2-bit digital comparator is given in Table 1.

Table 1. Magnitude Comparator 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

Digital comparators commonly use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values 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 also design larger multi-bit comparators by cascading together n of these and produce a 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.

Commercially available digital comparators such as the TTL **74LS85** or CMOS **4063** 4-bit magnitude comparators have additional input terminals to allow more individual comparator circuits to be "cascaded" together in order to compare binary words larger than 4-bits with magnitude comparators of 8, 16 or even 32-bit words being produced.

End of this Combinational Logic eBook

Last revision: June 2023

With the completion of this Combinational Logic eBook you should have gained a good and basic understanding and knowledge of the various combinational logic circuits which can be constructed using AND, OR, NOT and Ex-OR gates. The information provided here should give you a firm foundation for continuing your study of electronics and electrical engineering as well as the study of digital logic circuits.

For more information about any of the topics covered here please visit our website at:

www.electronics-tutorials.ws

Main Headquarters
245 Main Street
Cambridge, MA 02142
www.aspencore.com

Central Europe/EMEA
Frankfurter Strasse 211
63263 Neu-Isenburg, Germany
info-europe@aspencore.com