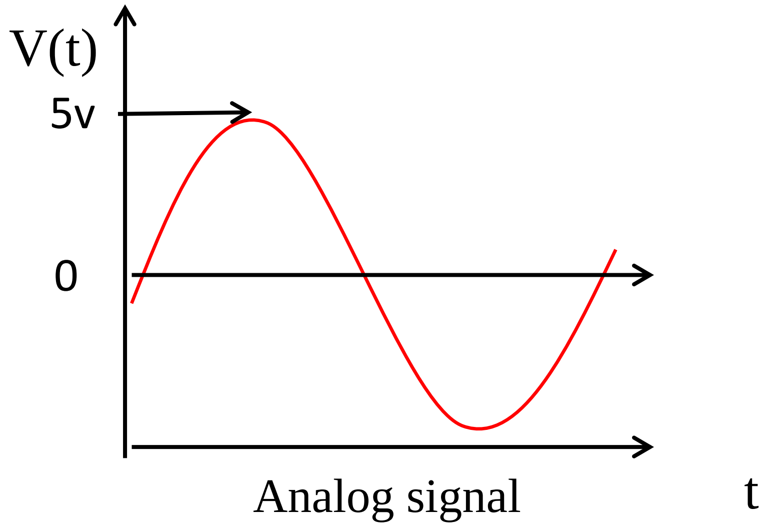
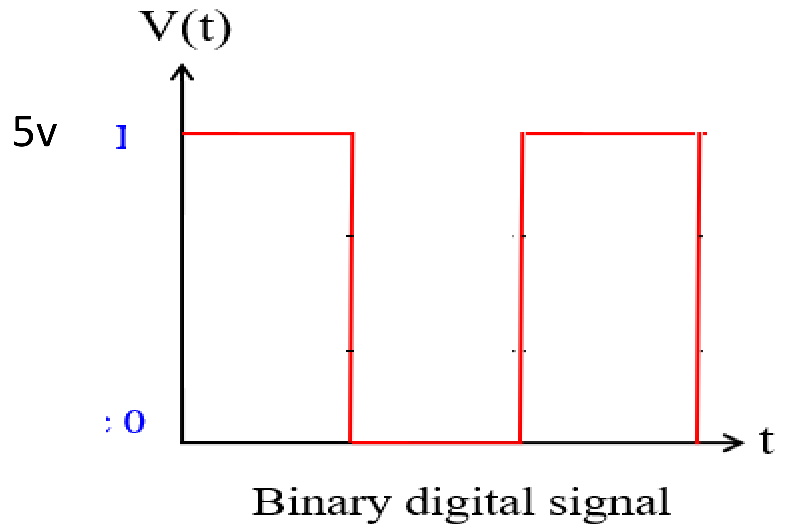
Analog and Digital Signal

- Analog system
  - The physical quantities or signals may vary continuously over a specified range.
- Digital system
  - The physical quantities or signals can assume only discrete values.
  - Greater accuracy



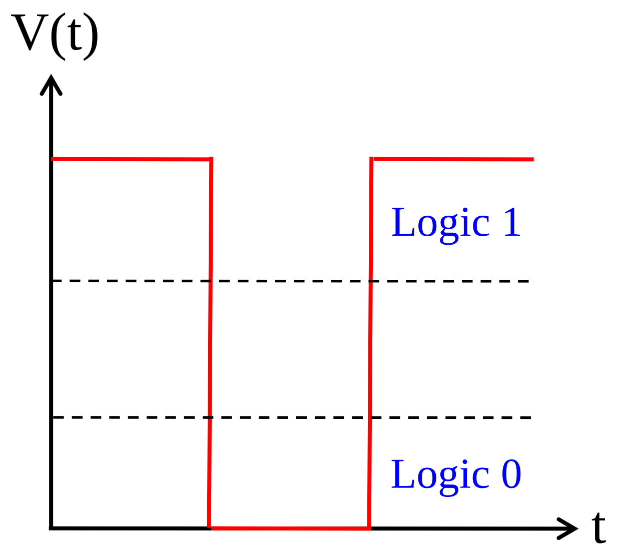
Analog signal



Binary digital signal

Binary Digital Signal

- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
  - Two level, or binary values are the most prevalent values.
- Binary values are represented abstractly by: (bit) bi – Binary &   it- digit
  - Number 0 is one bit and Number 1 is also one bit
  - Words (symbols) False (F) and True (T)
  - Words (symbols) Low (L) and High (H)
  - And words   On   and   Off
- Binary values are represented by values
- or ranges of values of physical quantities.



Binary digital signal

Decimal Number System

- Base (also called radix) = 10
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
  - Integer & fraction
- Digit Weight
  - Weight = (*Base*) $^{\textit{Position}}$
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

2	1	0	-1	-2
5	1	2	7	4
100	10	1	0.1	0.01
500	10	2	0.7	0.04

$$d_2{*}B^2+d_1{*}B^1+d_0{*}B^0+d_{-1}{*}B^{-1}+d_{-2}{*}B^{-2}$$

$$(512.74)_{10}$$

Octal Number System

- Base = 8
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

64	8	1		1/8	1/64
$\boxed{5}$	$\boxed{1}$	$\boxed{2}$		$\boxed{7}$	$\boxed{4}$
2	1	0		-1	-2

$$5 * 8^{2} + 1 * 8^{1} + 2 * 8^{0} + 7 * 8^{-1} + 4 * 8^{-2}$$

$$=(330.9375)_{10}$$

$$(512.74)_{8}$$

Decimal to Octal Conversion

Example: (175)$_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\, a_1\, a_0)_8 = (257)_8$

Example: (0.3125)$_{10}$

	Integer		Fraction	Coefficient
0.3125 * 8 =	2	.	5	$a_{-1} = 2$
0.5 * 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Hexadecimal Number System

- Base = 16
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
  - Weight = (*Base*) $^{\textit{Position}}$
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

256	16	1	1/16	1/256
1	E	5	7	A
2	1	0	-1	-2

$$1 * 16^{2} + 14 * 16^{1} + 5 * 16^{0} + 7 * 16^{-1} + 10 * 16^{-2}$$

$$= (485.4765625)_{10}$$

$$(1E5.7A)_{16}$$

Decimal to Octal

8	250	
8	31	2
	3	7

$(3\ 7\ 2)_8 = (250)_{10}$

Decimal to HexDeci

16	250	
	15	10
	F	A

$(FA)_{16} = (250)_{10}$

0	---	9	A	B	C	D	E	F
			10	11	12	13	14	15

Binary Number System

- Base = 2
  - 2 digits { 0, 1 }, called *b*inary dig*its* or “*bits*”
- Weights
  - Weight = (*Base*) $^{\textit{Position}}$
- Magnitude
  - Sum of “*Bit* x *Weight*”
- Formal Notation
- Groups of bits

4 bits = *Nibble*

8 bits = *Byte*

$$\begin{array}{ccccc} 4 & 2 & 1 & 1/2 & 1/4 \\ \boxed{1} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} \\ 2 & 1 & 0 & -1 & -2 \end{array}$$

$$1 *2^{2}+0 *2^{1}+1 *2^{0}+0 *2^{-1}+1 *2^{-2}$$

$$=(5.25)_{10}$$

$$(101.01)_{2}$$

Decimal (*Integer*) to Binary Conversion

- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
$13 / 2 =$	6	1	$a_0 = 1$
$6 / 2 =$	3	0	$a_1 = 0$
$3 / 2 =$	1	1	$a_2 = 1$
$1 / 2 =$	0	1	$a_3 = 1$

Answer: $(13)_{10} = (\underset{\text{MSB}}{a_3}\ a_2\ a_1\ \underset{\text{LSB}}{a_0})_2 = (1101)_2$

BINARY TO DECIMAL $(110111)_2 = (55)_{10}$

$$(110111)_2 = (1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10}$$

$$= (32 + 16 + 0 + 4 + 2 + 1)_{10}$$

$$= (55)_{10}$$

DECIMAL TO BINARY $(55.375)_{10}$

Whole Number Part

2	55	
2	27	1 LSB
2	13	1
2	6	1
2	3	0
	1 MSB	1

$(110111)_2 = (55)_{10}$

Fractional Part

	overflow
$0.375 \times 2 = 0.750$	0 MSB
$0.750 \times 2 = 1\ .50$	1
$0.50 \times 2 = 1\ .00$	1 LSB
$\boxed{(0.011)_2 = (0.375)_{10}}$	$\boxed{(0.011)_2}$

$$(0.011)_2 = (0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3})_{10}$$
$$= (\ 0\ +\ 1/4\ + 1/8\ )_{10}$$
$$= (0\ +\ 0.25 + 125)_{10}$$
$$= (0.375)_{10}$$

$\boxed{\textbf{Result} = (110111.011\ )_2}$

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the ‘Base’ (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.\underset{\underset{\text{MSB}}{\uparrow}}{a_{-1}}\ a_{-2}\ \underset{\underset{\text{LSB}}{\uparrow}}{a_{-3}})_2 = (0.101)_2$

The Power of 2

n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$

n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	**Kilo**
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	**Mega**
30	$2^{30}=1G$	**Giga**
40	$2^{40}=1T$	**Tera**

Addition

- Decimal Addition

$$\begin{array}{rccc} & 1 & 1 & \\ & & 5 & 5 \\ + & & 5 & 5 \\ \hline & 1 & 1 & 0 \end{array}$$

$\leftarrow$ **Carry**

= Ten $\geq$ Base

➔ Subtract a Base

BINARY ARITHMETIC

Addition	Sum	Carry
0 + 0 = 0	0	0
0 + 1 = 1	1	0
1 + 0 = 1	1	0
1 + 1 = 0	0	1

Subtraction	Diff.	Borrow
0 - 0 = 0	0	0
1 - 0 = 1	1	0
1 - 1 = 0	0	0
0 - 1 = 1	1	1

$10 - 9$

$1\ 0 - 9 = \boxed{1}\quad \boxed{1}$

Multiplication	Product
0 x 0 = 0	0
0 x 1 = 1 0	0
1 x 0 = 1 0	0
1 x 1 = 0 1	1

Binary Addition

- Column Addition

$$\begin{array}{lccccccccc}
 & & 1 & 1 & 1 & 1 & 1 & 1 & & \\
\textbf{Augend} & & & 1 & 1 & 1 & 1 & 0 & 1 & = 61_{10} \\
\textbf{Add end} & + & & & 1 & 0 & 1 & 1 & 1 & = 23_{10} \\
\hline
\textbf{Sum} & (& 1 & 0 & 1 & 0 & 1 & 0 & 0\ ) & = 84_{10}
\end{array}$$

Binary Subtraction

- Borrow a “Base” when needed

$$\begin{array}{lccccccccc}
 & & & 1 & & & & & & \\
 & & & \cancel{10} & 10 & & \cancel{10} & 10 & & \\
\text{Minu end} & & \cancel{1} & 0 & 0 & \cancel{1} & \cancel{1} & 0 & 1 & = 77 \\
\text{Subra end} & - & & & 1 & 0 & 1 & 1 & 1 & = 23 \\
\hline
\text{Difference} & & 0 & 1 & 1 & 0 & 1 & 1 & 0 & = 54
\end{array}$$

Binary Multiplication

- Bit by bit

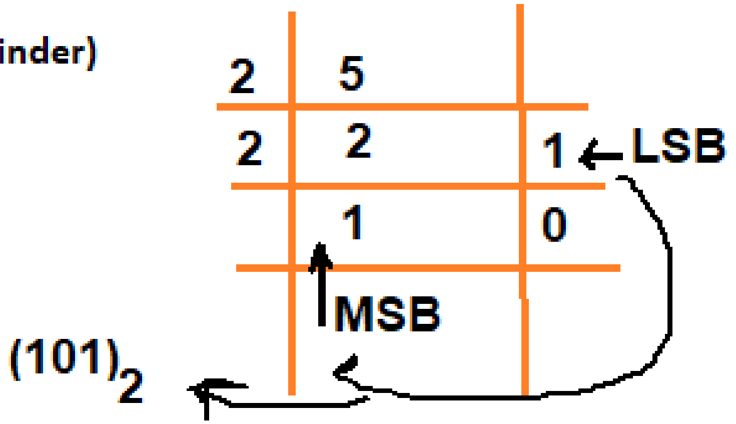
				1	0	1	1	1	Multiplicand	23_{10}
x					1	0	1	0	Multiplier	10_{10}
				0	0	0	0	0		
			1	0	1	1	1	x		
		0	0	0	0	0	x	x		
	1	0	1	1	1	x	x	x		
	1	1	1	0	0	1	1	0	Product	230_{10}

BINARY DIVISION

Division of $(1111101)_2 = (125)_{10}$ by $(11001)_2 = (25)_{10}$

```
                       1 0 1      (Quotient)
                   _____________________
(Divisor)  1 1 0 0 1 | 1 1 1 1 1 0 1
                     | 1 1 0 0 1
                     |___________________
                     |       1 1 0 0 1
                     |       1 1 0 0 1
                     |___________________
                     |          X         (Remainder)
```

$$\left(\frac{125}{25}\right)_{10} = (5)_{10}$$



Answer = $(1\ 0\ 1)_2 = (5)_{10}$

Home / Binary Numbers / Octal Number System

Octal Number System

The Octal Number System is another type of computer and digital numbering system which uses the Base-8 system

The **Octal Numbering System** is very similar in principle to the previous hexadecimal numbering system except that in Octal, a binary number is divided up into groups of only 3 bits, with each group or set of bits having a distinct value of between 000 (0) and 111 (4+2+1 = 7).

Octal numbers therefore have a range of just “8” digits, (0, 1, 2, 3, 4, 5, 6, 7) making them a Base-8 numbering system and therefore, q is equal to “8”.

Then the main characteristics of an **Octal Numbering System** is that there are only 8 distinct counting digits from 0 to 7 with each digit having a weight or value of just 8 starting from the least significant bit (LSB). In the earlier days of computing, octal numbers and the octal numbering system was very popular for counting inputs and outputs because as it works in counts of eight, inputs and outputs were in counts of eight, a byte at a time.

As the base of an **Octal Numbers** system is 8 (base-8), which also represents the number of individual numbers used in the system, the subscript 8 is used to identify a number expressed in octal. For example, an octal number is expressed as: 237_8

Just like the hexadecimal system, the “octal number system” provides a convenient way of converting large binary numbers into more compact and smaller groups. However, these days the octal numbering system is used less frequently than the more popular hexadecimal numbering system and has almost disappeared as a digital base number system.

Representation of an Octal Number

MSB	Octal Number							LSB
8^8	8^7	8^6	8^5	8^4	8^3	8^2	8^1	8^0
16M	2M	262k	32k	4k	512	64	8	1

As the octal number system uses only eight digits (0 through 7) there are no numbers or letters used above 8, but the conversion from decimal to octal and binary to octal follows the same pattern as we have seen previously for hexadecimal.

To count above 7 in octal we need to add another column and start over again in a similar way to hexadecimal.

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21….etc

Again do not get confused, 10 or 20 is **NOT** ten or twenty it is 1 + 0 and 2 + 0 in octal exactly the same as for hexadecimal. The relationship between binary and octal numbers is given below.

Octal Numbers

Decimal Number	3-bit Binary Number	Octal Number
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7
8	001 000	10 (1+0)
9	001 001	11 (1+1)
Continuing upwards in groups of three		

Then we can see that 1 octal number or digit is equivalent to 3 bits, and with two octal number, 77_8we can count up to 63 in decimal, with three octal numbers, 777_8 up to 511 in decimal and with four octal numbers, 7777_8 up to 4095 in decimal and so on.

Octal Numbers Example No1

Using our previous binary number of 1101010111001111_2 convert this binary number to its octal equivalent, (base-2 to base-8).

Binary Digit Value	0011010101110011111
Group the bits into three´s starting from the right hand side	001 101 010 111 001 111
Octal Number form	$1\,5\,2\,7\,1\,7_8$

Thus, 0011010101110011111_2 in its Binary form is equivalent to 152717_8 in Octal form or 54,735 in denary.

Octal Numbers Example No2

Convert the octal number 2322_8 to its decimal number equivalent, (base-8 to base-10).

Octal Digit Value	2322_8
In polynomial form	$= (2\times8^3) + (3\times8^2) + (2\times8^1) + (2\times8^0)$
Add the results	$= (1024) + (192) + (16) + (2)$
Decimal number form equals: 1234_{10}	

Then, converting octal to decimal shows that 2322_8 in its Octal form is equivalent to 1234_{10} in its Decimal form.

While **Octal** is another type of digital numbering system, it is little used these days instead the more commonly used Hexadecimal Numbering System is used as it is more flexible.

Hexadecimal Numbers

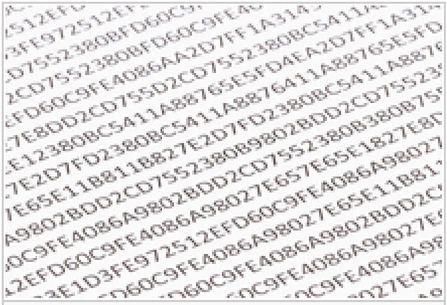
Hexadecimal Numbers group binary numbers into sets of four allowing for the conversion of 16 different binary digits

The one main disadvantage of binary numbers is that the binary string equivalent of a large decimal base-10 number can be quite long.

When working with large digital systems, such as computers, it is common to find binary numbers consisting of 8, 16 and even 32 digits which makes it difficult to both read or write without producing errors especially when working with lots of 16 or 32-bit binary numbers.

One common way of overcoming this problem is to arrange the binary numbers into groups or sets of four bits (4-bits). These groups of 4-bits uses another type of numbering system also commonly used in computer and digital systems called **Hexadecimal Numbers**.

The "Hexadecimal" or simply "Hex" numbering system uses the **Base of 16** system and are a popular choice for representing long binary values because their format is quite compact and much easier to understand compared to the long binary strings of 1's and 0's.



Hexadecimal Number String

Being a Base-16 system, the hexadecimal numbering system therefore uses 16 (sixteen) different digits with a combination of numbers from 0 through to 15. In other words, there are 16 possible digit symbols.

However, there is a potential problem with using this method of digit notation caused by the fact that the decimal numerals of 10, 11, 12, 13, 14 and 15 are normally written using two adjacent symbols. For example, if we write 10 in hexadecimal, do we mean the decimal number ten, or the binary number of two (1 + 0). To get around this tricky problem hexadecimal numbers that identify the values of ten, eleven, . . . , fifteen are replaced with capital letters of A, B, C, D, E and F respectively.

Then in the **Hexadecimal Numbering System** we use the numbers from 0 to 9 and the capital letters A to F to represent its Binary or Decimal number equivalent, starting with the least significant digit at the right hand side.

As we have just said, binary strings can be quite long and difficult to read, but we can make life easier by splitting these large binary numbers up into even groups to make them much easier to write down and understand. For example, the following group of binary digits $1101\ 0101\ 1100\ 1111_2$ are much easier to read and understand than 1101010111001111_2 when they are all bunched up together.

In the everyday use of the decimal numbering system we use groups of three digits or 000's from the right hand side to make a very large number such as a million or trillion, easier for us to understand and the same is also true in digital systems.

Hexadecimal Numbers is a more complex system than using just binary or decimal and is mainly used when dealing with computers and memory address locations. By dividing a binary number up into groups of 4 bits, each group or set of 4 digits can now have a possible value of between "0000" (0) and "1111" (8+4+2+1 = 15) giving a total of **16** different number combinations from 0 to 15. Don't forget that "0" is also a valid digit.

We remember from our first tutorial about **Binary Numbers** that a 4-bit group of digits is called a "nibble" and as 4-bits are also required to produce a hexadecimal number, a hex digit can also be thought of as a nibble, or half-a-byte. Then two hexadecimal numbers are required to produce one full byte ranging from 00 to FF.

Also, since 16 in the decimal system is the fourth power of 2 (or 2^4), there is a direct relationship between the numbers 2 and 16 so one hex digit has a value equal to four binary digits so now q is equal to "16".

Because of this relationship, four digits in a binary number can be represented with a single hexadecimal digit. This makes conversion between binary and hexadecimal numbers very easy, and hexadecimal can be used to write large binary numbers with much fewer digits.

The numbers 0 to 9 are still used as in the original decimal system, but the numbers from 10 to 15are now represented by capital letters of the alphabet from A to F inclusive and the relationship between decimal, binary and hexadecimal is given below.

Hexadecimal Numbers

Decimal Number	4-bit Binary Number	Hexadecimal Number
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4

5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10 (1+0)
17	0001 0001	11 (1+1)
Continuing upwards in groups of four		

Using the original binary number from above $1101\ 0101\ 1100\ 1111_2$ this can now be converted into an equivalent hexadecimal number of D5CF which is much easier to read and understand than a long row of 1's and 0's that we had before.

So by using hexadecimal notation, digital numbers can be written using fewer digits and with a much less likelihood of an error occurring. Similarly, converting hexadecimal based numbers back into binary is simply the reverse operation.

Then the main characteristics of a **Hexadecimal Numbering System** is that there are 16 distinct counting digits from 0 to F with each digit having a weight or value of 16 starting from the least significant bit (LSB). In order to distinguish Hexadecimal numbers from Denary numbers, a prefix of either a "#", (Hash) or a "$" (Dollar sign) is used before the actual **Hexadecimal Number** value, #D5CF or $D5CF.

As the base of a hexadecimal system is 16, which also represents the number of individual symbols used in the system, the subscript 16 is used to identify a number expressed in hexadecimal. For example, the previous hexadecimal number is expressed as: $D5CF_{16}$

Counting using Hexadecimal Numbers

So we now know how to convert 4 binary digits into a hexadecimal number. But what if we had more than 4 binary digits how would we count in hexadecimal beyond the final letter F. The simple answer is to start over again with another set of 4 bits as follows.

0…to…9, A,B,C,D,E,F, 10…to…19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21….etc

Do not get confused, 10 or 20 is **NOT** ten or twenty it is 1 + 0 and 2 + 0 in hexadecimal. In fact twenty does not even exist in hex. With two hexadecimal numbers we can count up to FF which is equal to decimal 255. Likewise, to count higher than FF we would add a third hexadecimal digit to the left so the first 3-bit hexadecimal number would be $100_{16,}$ (256_{10}) and the last would be $FFF_{16,}$(4095_{10}). The maximum 4-digit hexadecimal number is $FFFF_{16}$ which is equal to 65,535 in decimal and so on.

Representation of a Hexadecimal Number

MSB	Hexadecimal Number							LSB
16^8	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0
4.3G	2.6G	16M	1M	65k	4k	256	16	1

This adding of additional hexadecimal digits to convert both decimal and binary numbers into an **Hexadecimal Number** is very easy if there are 4, 8, 12 or 16 binary digits to convert. But we can also add zero's to the left of the most significant bit, the MSB if the number of binary bits is not a multiple of four.

For example, 11001011011001_2 is a fourteen bit binary number that is to large for just three hexadecimal digits only, yet too small for a four hexadecimal number. The answer is to ADD additional zero's to the left most bit until we have a complete set of four bit binary number or multiples thereof.

Adding of Additional 0's to a Binary Number

Binary Number	0011	0010	1101	1001
Hexadecimal Number	3	2	D	9

The main advantage of a **Hexadecimal Number** is that it is very compact and by using a base of 16 means that the number of digits used to represent a given number is usually less than in binary or decimal. Also, it is quick and easy to convert between hexadecimal numbers and binary.

Hexadecimal Numbers Example No1

Convert the following Binary number $1110\ 1010_2$ into its Hexadecimal number equivalent.

11101010_2			
Group the bits into four's starting from the right hand side			
=	1110	1010	
Find the Decimal equivalent of each individual group			
=	14	10	(in decimal)
Convert to Hexadecimal using the table above			
=	E	A	(in Hex)
Then, the hexadecimal equivalent of the binary number $1110\ 1010_2$ is $\#EA_{16}$			

Hexadecimal Numbers Example No2

Convert the following Hexadecimal number $\#3FA7_{16}$ into its Binary equivalent, and also into its Decimal or Denary equivalent using subscripts to identify each numbering system.

$\#3FA7_{16}$

$= 0011\ 1111\ 1010\ 0111_2$

= (8192 + 4096 + 2048 + 1024 + 512 + 256 + 128 + 32 + 4 + 2 + 1)

$= 16{,}295_{10}$

Then, the Decimal number of **16,295** can be represented as:-

$\#3FA7_{16}$ in Hexadecimal

or

$0011\ 1111\ 1010\ 0111_2$ in Binary.

Hexadecimal Numbers Summary

Then to summarise. The **Hexadecimal**, or **Hex**, numbering system is commonly used in computer and digital systems to reduce large strings of binary numbers into a sets of four digits for us to easily understand. The word “Hexadecimal” means sixteen because this type of digital numbering system uses 16 different digits from 0-to-9, and A-to-F.

Hexadecimal Numbers group binary numbers into sets of four digits. To convert a binary sequence into an equivalent *hexadecimal number*, we must first group the binary digits into a set of 4-bits. These binary sets can have any value from 0_{10} (0000_2) to 15_{10} (1111_2) representing the hexadecimal equivalent of 0 through to F.

In the next tutorial about Binary Logic we will look at converting whole strings of binary numbers into another digital numbering system called **Octal Numbers** and vice versa.

Binary Numbers Tutorial

There are different yet similar binary numbering systems used in digital electronic circuits and computers.

However, the numbering system used in one type of circuit may be different to that of another type of circuit, for example, the memory of a computer would use hexadecimal numbers while the keyboard uses decimal numbers.

Then the conversion from one number system to another is very important with the four main forms of arithmetic being.

- Decimal – The decimal numbering system has a base of 10 (MOD-10) and uses the digits from 0 through 9 to represent a decimal number value.
- Binary – The binary numbering system has a base of 2 (MOD-2) and uses only two digits a “0” and a “1” to represent a binary number value.
- Octal – The octal numbering system has a base of 8 (MOD-8) and uses 8 digits between 0 and 7 to represent an octal number value.
- Hexadecimal – The Hexadecimal numbering system has a base of 16 (MOD-16) and uses a total of 16 numeric and alphabetic characters to represent a number value. Hexadecimal numbers consist of digits 0 through 9 and letters A to F.

Long binary numbers are difficult to both read or write and are generally converted into a system more easily understood or user friendly. The two most common derivatives based on binary numbers are the **Octal** and the **Hexadecimal** numbering systems, with both of these limited in length to a byte (8-bits) or a word (16-bits).

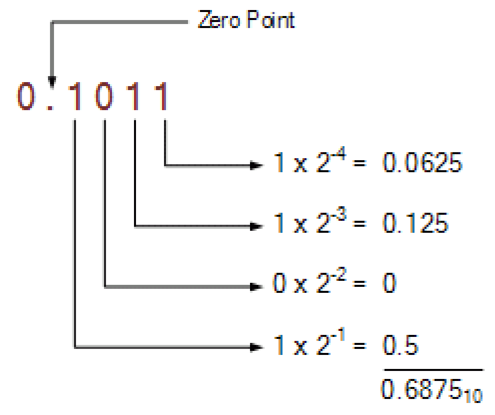
Octal numbers can be represented by groups of 3-bits and hexadecimal numbers by groups of 4-bits together, with this grouping of the bits being used in electronic or computer systems in displays or printouts. The grouping together of binary numbers can also be used to represent **Machine Code**used for programming instructions and control such as an **Assembly Language**.

Comparisons between the various **Decimal**, **Binary**, **Hexadecimal** and **Octal** numbers are given in the following table.

Digital Numbering System Comparison Table

Base, b	Byte (8-bits)	Word (16-bits)
Decimal	0 to 255_{10}	0 to $65,535_{10}$
Binary	0000 0000 to $1111\ 1111_2$	0000 0000 0000 0000 to $1111\ 1111\ 1111\ 1111_2$
Hexadecimal	00 to FF_{16}	0000 to $FFFF_{16}$
Octal	000 to 377_8	000 000 to $177\ 777_8$

We can see from the table above that the Hexadecimal numbering system uses only four digits to express a single 16-bit word length, and as a result it is the most commonly used **Base Numbering System** for digital, micro-electronic and computer systems.



Binary Fractions

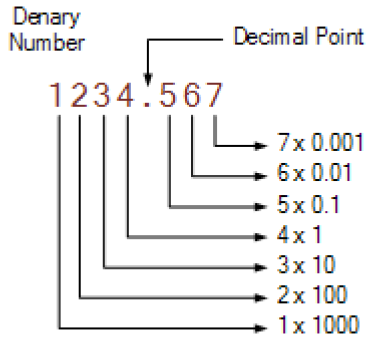
Binary Fractions use the same weighting principle as decimal numbers except that each binary digit uses the base-2 numbering system

We know that decimal (or *denary*) numbers use the base ten (base-10) numbering system where each digit in a decimal number is allowed to take one of ten possible values in the range of 0 to 9. So moving from right to left along a decimal number, each digit will have a value ten times greater than the digit to its immediate right.

But as well as each digit being ten times bigger than the previous number as we move from right-to-left, each digit can also be ten times smaller than its neighbouring number as we move along in the opposite direction from left-to-right.

However, once we reach zero (0) and the decimal point, we do not need to just stop, but can continue moving from left-to-right along the digits producing what are generally called **Fractional Numbers**.

A Typical Fractional Number



Here in this decimal (or denary) number example, the digit immediately to the right of the decimal point (number 5) is worth one tenth (1/10 or 0.1) of the digit immediately to the left of the decimal point (number 4) which as a multiplication value of one (1).

Thus as we move through the number from left-to-right, each subsequent digit will be one tenth the value of the digit immediately to its left position, and so on.

Then the decimal numbering system uses the concept of positional or relative weighting values producing a positional notation, where each digit represents a different weighted value depending on the position occupied either side of the decimal point.

Thus mathematically in the standard denary numbering system, these values are commonly written as: 4^0, 3^1, 2^2, 1^3 for each position to the left of the decimal point in our example above. Likewise, for the fractional numbers to right of the decimal point, the weight of the number becomes more negative giving: 5^{-1}, 6^{-2}, 7^{-3} etc.

So we can see that each digit in the standard decimal system indicates the magnitude or weight of that digit within the number. Then the value of any decimal number will be equal to the sum of its digits multiplied by their respective weights, so for our example above: $N = 1234.567_{10}$ in the weighted decimal format this will be equal too:

$$1000 + 200 + 30 + 4 + 0.5 + 0.06 + 0.007 = 1234.567_{10}$$

or it could be written to reflect the weighting of each denary digit:

$$(1\times1000) + (2\times100) + (3\times10) + (4\times1) + (5\times0.1) + (6\times0.01) + (7\times0.001) = 1234.567_{10}$$

or even in polynomial form as:

$$(1\times10^3) + (2\times10^2) + (3\times10^1) + (4\times10^0) + (5\times10^{-1}) + (6\times10^{-2}) + (7\times10^{-3}) = 1234.567_{10}$$

We can also use this idea of positional notation where each digit represents a different weighted value depending upon the position it occupies in the binary numbering system. The difference this time is that the binary number system (or simply binary numbers) is a positional system, where the different weighted positions of the digits are to the power of 2 (base-2) instead of 10.

Binary Fractions

The binary numbering system is a base-2 numbering system which contains only two digits, a “0” or a “1”. Thus each digit of a binary number can take the “0” or the “1” value with the position of the 0 or 1 indicating its value or weighting. But we can also have binary weighting for values of less than 1 producing what are called unsigned fractional binary numbers.

Similar to decimal fractions, binary numbers can also be represented as unsigned fractional numbers by placing the binary digits to the right of the decimal point or in this case, binary point. Thus all the fractional digits to the right of the binary point have respective weightings which are negative powers of two, creating a binary fraction. In other words, the powers of 2 are negative.

So for the fractional binary numbers to the right of the binary point, the weight of each digit becomes more negative giving: 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, and so on as shown.

Binary Fractions

$$2^{0} = 1$$

$$2^{-1} = \frac{1}{2^{1}} = \frac{1}{2} = 0.5$$

$$2^{-2} = \frac{1}{2^{2}} = \frac{1}{4} = 0.25$$

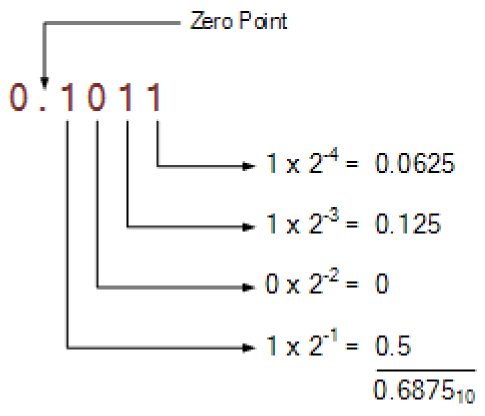
$$2^{-3} = \frac{1}{2^{3}} = \frac{1}{8} = 0.125$$

$$2^{-4} = \frac{1}{2^{4}} = \frac{1}{16} = 0.0625$$

$$2^{-5} = \frac{1}{2^{5}} = \frac{1}{32} = 0.03125$$

etc, etc.

Thus if we take the binary fraction of 0.1011_2 then the positional weights for each of the digits is taken into account giving its decimal equivalent of:



For this example, the decimal fraction conversion of the binary number 0.1011_2 is 0.6875_{10}.

Binary Fractions Example No1

Now lets suppose we have the following binary number of: 1101.0111_2, what will be its decimal number equivalent.

1101.0111 = $(1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) + (0\times2^{-1}) + (1\times2^{-2}) + (1\times2^{-3}) + (1\times2^{-4})$

= 8 + 4 + 0 + 1 + 0 + 1/4 + 1/8 + 1/16

= 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125 + 0.0625 = 13.4375_{10}

Hence the decimal equivalent number of 1101.0111_2 is given as: 13.4375_{10}

So we can see that fractional binary numbers, that is binary numbers that have a weighting of less than 1 (2^0), can be converted into their decimal number equivalent by successively dividing the binary weighting factor by the value of two for each decrease in the power of 2, remembering also that 2^0 is equal to 1, and not zero.

Other Binary Fraction Examples

0.11 = $(1\times2^{-1}) + (1\times2^{-2})$ = 0.5 + 0.25 = 0.75_{10}

11.001 = $(1\times2^1) + (1\times2^0) + (1\times2^{-3})$ = 2 + 1 + 0.125 = 3.125_{10}

1011.111 = $(1\times2^3) + (1\times2^1) + (1\times2^0)\ (1\times2^{-1}) + (1\times2^{-2}) + (1\times2^{-3})$

= 8 + 2 + 1 + 0.5 + 0.25 + 0.125 = 11.875_{10}

Converting Decimal to a Binary Fraction

The conversion of a decimal fraction to a fractional binary number is achieved using a method similar to that we used for integers. However, this time multiplication is used instead of division with the integers instead of remainders used with the carry digit being the binary equivalent of the fractional part of the decimal number.

When converting from decimal to binary, the integer (positive sequence right-to-left) part and the fractional (negative sequence from left-to-right) part of the decimal number are calculated separately.

For the integer part of the number, the binary equivalent is found by successively dividing (known as successive division) the integer part of the decimal number repeatedly by 2 (÷2), noting the remainders in reverse order from the least significant bit (LSB) to the most significant bit (MSB), until the value becomes “0” producing the binary equivalent.

So to find the binary equivalent of the decimal integer: 118_{10}

118 (divide by 2) = 59 plus remainder **0** (LSB)

59 (divide by 2) = 29 plus remainder **1** (↑)

29 (divide by 2) = 14 plus remainder **1** (↑)

14 (divide by 2) = 7 plus remainder **0** (↑)

7 (divide by 2) = 3 plus remainder **1** (↑)

3 (divide by 2) = 1 plus remainder **1** (↑)

1 (divide by 2) = 0 plus remainder **1** (MSB)

Then the binary equivalent of 118_{10} is therefore: 1110110_2 ← (LSB)

The fractional part of the number is found by successively multiplying (known as successive multiplication) the given fractional part of the decimal number repeatedly by 2 (×2), noting the carries in forward order, until the value becomes "0" producing the binary equivalent.

So if the multiplication process produces a product greater than 1, the carry is a "1" and if the multiplication process produces a product less than "1", the carry is a "0".

Note also that if the successive multiplication processes does not seem to be heading towards a final zero, the fractional number will have an infinite length or until the equivalent number of bits have been obtained, for example 8-bits. or 16-bits, etc. depending on the degree of accuracy required.

So to find the binary fraction equivalent of the decimal fraction: 0.8125_{10}

0.8125 (multiply by 2) = **1**.625 = 0.625 carry **1** (MSB)

0.625 (multiply by 2) = **1**.25 = 0.25 carry **1** (↓)

0.25 (multiply by 2) = **0**.50 = 0.5 carry **0** (↓)

0.5 (multiply by 2) = **1**.00 = 0.0 carry **1** (LSB)

Thus the binary equivalent of 0.8125_{10} is therefore: 0.1101_2 ← (LSB)

We can double check this answer using the procedure above to convert a binary fraction into a decimal number equivalent: $0.1101 = 0.5 + 0.25 + 0.0625 = 0.8125_{10}$

Binary Fraction Example No2

Find the binary fraction equivalent of the following decimal number: 54.6875

First we convert the integer 54 to a binary number in the normal way using successive division from above.

54 (divide by 2) = 27 remainder **0** (LSB)

27 (divide by 2) = 13 remainder **1** (↑)

13 (divide by 2) = 6 remainder **1** (↑)

6 (divide by 2) = 3 remainder **0** (↑)

3 (divide by 2) = 1 remainder **1** (↑)

1 (divide by 2) = 0 remainder **1** (MSB)

Thus the binary equivalent of 54_{10} is therefore: 110110_2

Next we convert the decimal fraction 0.6875 to a binary fraction using successive multiplication.

0.6875 (multiply by 2) = **1**.375 = 0.375 carry **1** (MSB)

0.375 (multiply by 2) = **0**.75 = 0.75 carry **0** (↓)

0.75 (multiply by 2) = **1**.50 = 0.5 carry **1** (↓)

0.5 (multiply by 2) = **1**.00 = 0.0 carry **1** (LSB)

Thus the binary equivalent of 0.6875_{10} is therefore: 0.1011_2 ← (LSB)

Hence the binary equivalent of the decimal number: 54.6875_{10} is 110110.1011_2

Binary Fractions Summary

We have seen here in this tutorial about **Binary Fractions** that to convert any decimal fraction into its equivalent binary fraction, we must multiply the decimal fractional part, and only the decimal fractional part by 2 and record the digit that appears to the left of the binary point. This binary digit which is the carry digit will ALWAYS be either a “0” or a “1”.

We must then multiply the remaining decimal fraction by 2 again repeating the above sequence using successive multiplication until the fraction is reduced to zero or the required amount of binary bits has been completed for a repeating binary fraction. Fractional numbers are represented by negative powers of 2.

For mixed decimal numbers we must perform two separate operations. Successive division for the integer part to the left of the decimal point and successive multiplication for the fractional part to the right of the decimal point.

Note that the integer part of a mixed decimal number will always have an exact binary number equivalent but the decimal fractional part may not, since we could get a repeating fraction resulting in an infinite number of binary digits if we wanted to represent the decimal fraction exactly.