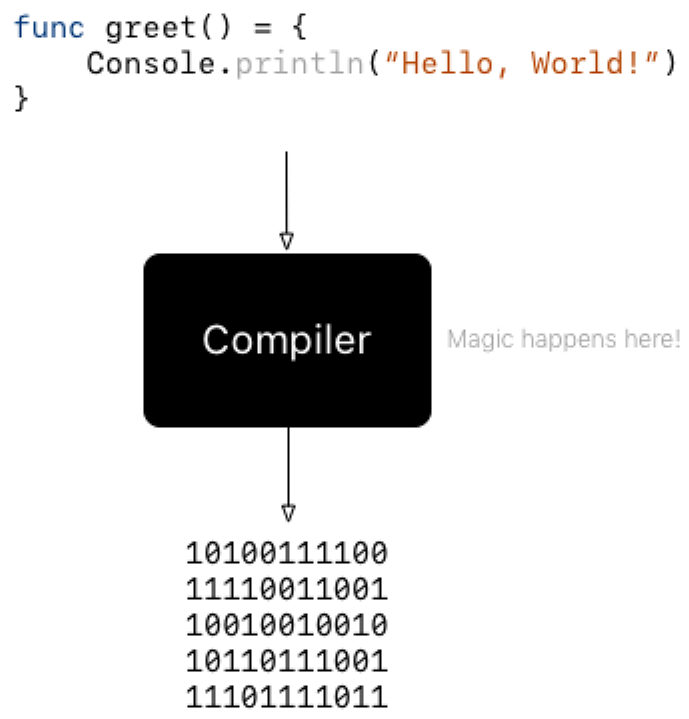
C++ Compilation :

The compilation of a C++ program involves three steps:

1. Preprocessing: the preprocessor takes a C++ source code file and deals with the `#include`s, `#define`s and other preprocessor directives. The output of this step is a "pure" C++ file without pre-processor directives.
2. Compilation: the compiler takes the pre-processor's output and produces an object file from it.
3. Linking: the linker takes the object files produced by the compiler and produces either a library or an executable file.



```
func greet() = {
    Console.println("Hello, World!")
}
```

```
10100111100
11110011001
10010010010
10110111001
11101111011
```

Object files are source file compiled into binary machine language, but they contain unresolved external references (such as `printf`,for instance). They may need to be linked against other object files, third party libraries and almost always against C/C++ runtime library.

Compiler vs Interpreter

The difference between an interpreter and a compiler is given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.

Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

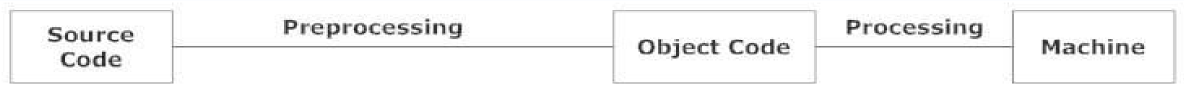


Figure: Compiler

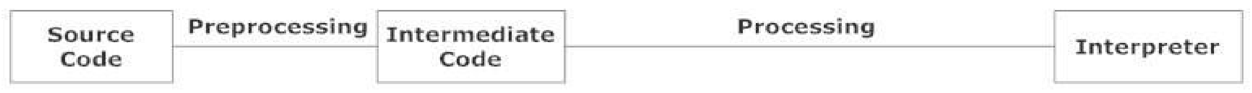


Figure: Interpreter

DataTypes in C++

Data types are used to tell the variables the type of data it can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires a different amount of memory.

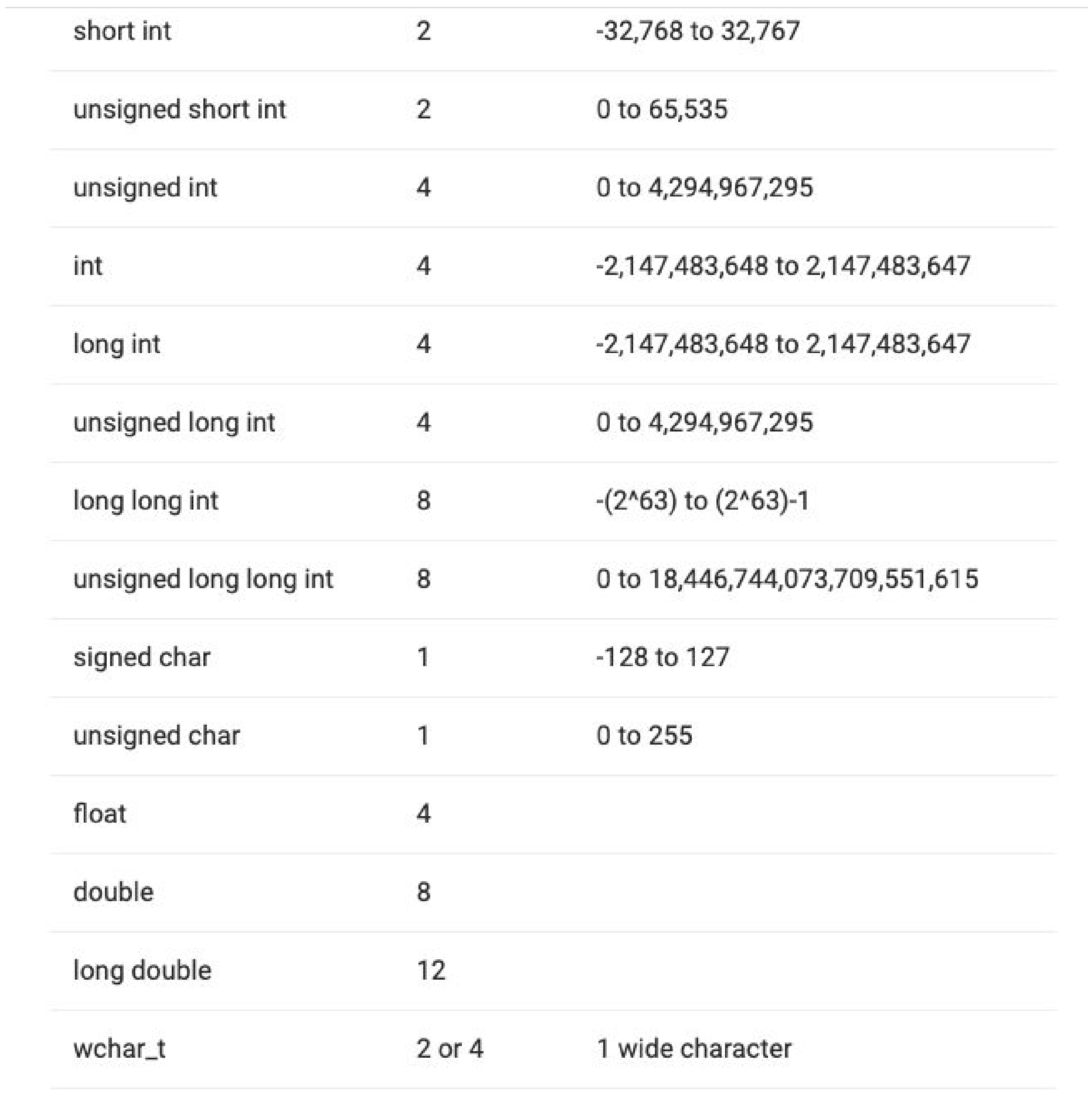
1. Primitive Data Types: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:
Integer
Character

Boolean
Floating Point
Double Floating Point
Valueless or Void

2.Derived Data Types: The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:
Function
Array
Pointer

3.Abstract or User-Defined Data Types: These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:
Class
Structure

DATA TYPE	SIZE (IN BYTES)	RANGE



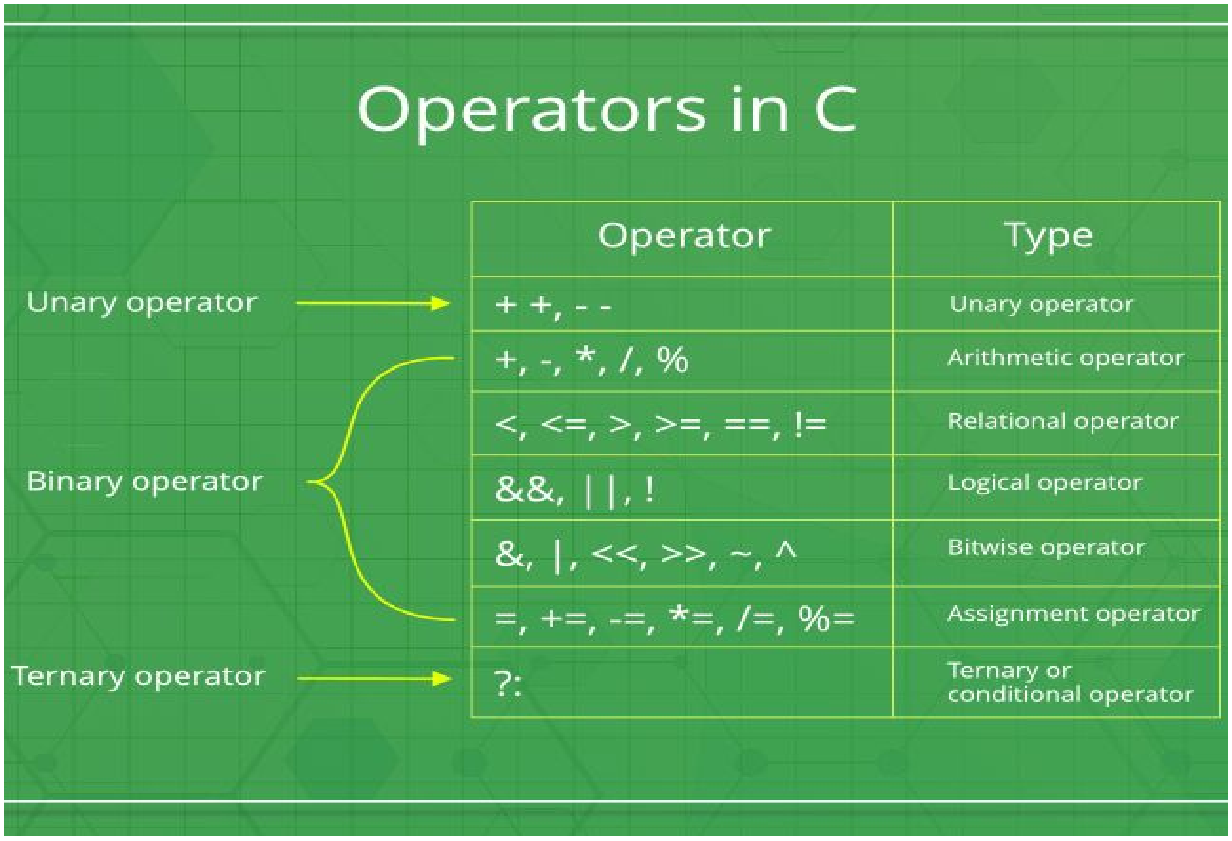
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 or 4	1 wide character

Operators in C++

Symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands.

For example, consider the below statement:

c = a + b;
Here, ‘+’ is the operator known as addition operator and ‘a’ and ‘b’ are operands. The addition operator tells the compiler to add both of the operands ‘a’ and ‘b’.



Unary operator
Binary operator
Ternary operator

Operators in C

Operator	Type
+ +, - -	Unary operator
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
&&, ||, !	Logical operator
&, |, <<, >>, ~, ^	Bitwise operator
=, +=, -=, *=, /=, %=	Assignment operator
?:	Ternary or conditional operator

<u>Arithmetic Operators:</u> These are the operators used to perform arithmetic/mathematical operations on operands. Examples: (+, -, *, /, %,++,–). Arithmetic operator are of two types:

Unary Operators: Operators that operates or works with a single operand are unary operators. For example: (++ , –)

Increment: The ‘++’ operator is used to increment the value of an integer. When placed before the variable name (also called pre-increment operator), its value is incremented instantly. For example, ++x.
And when it is placed after the variable name (also called post-increment operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement. For example, x++.

Decrement: The ‘ – – ‘ operator is used to decrement the value of an integer. When placed before the variable name (also called pre-decrement operator), its value is decremented instantly. For example, – – x.
And when it is placed after the variable name (also called post-decrement operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement. For example, x – –.

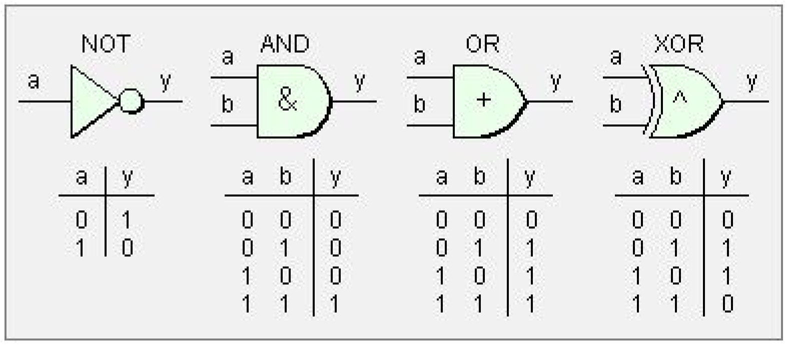
Binary Operators: Operators that operates or works with two operands are binary operators. For example: (+ , – , * , /)

Relational Operators: These are used for comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, an operand is greater than the other operand or not etc. Some of the relational operators are (==, >= , <=).

Logical Operators: Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a boolean value either true or false.
For example, the logical AND represented as ‘&&’ operator in C or C++ returns true when both the conditions under consideration are satisfied.

Otherwise it returns false. Therfore, a && b returns true when both a and b are true (i.e. non-zero).

FOUNDATION FOR BITWISE OPERATORS



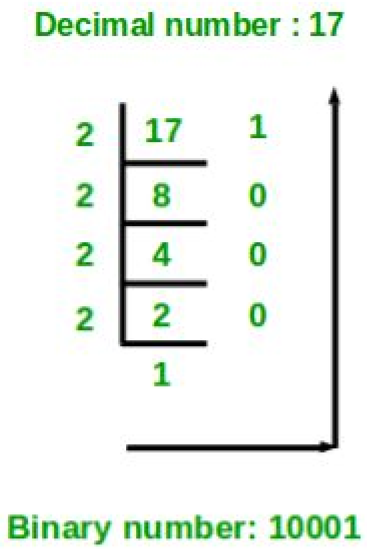
a	y
0	1
1	0

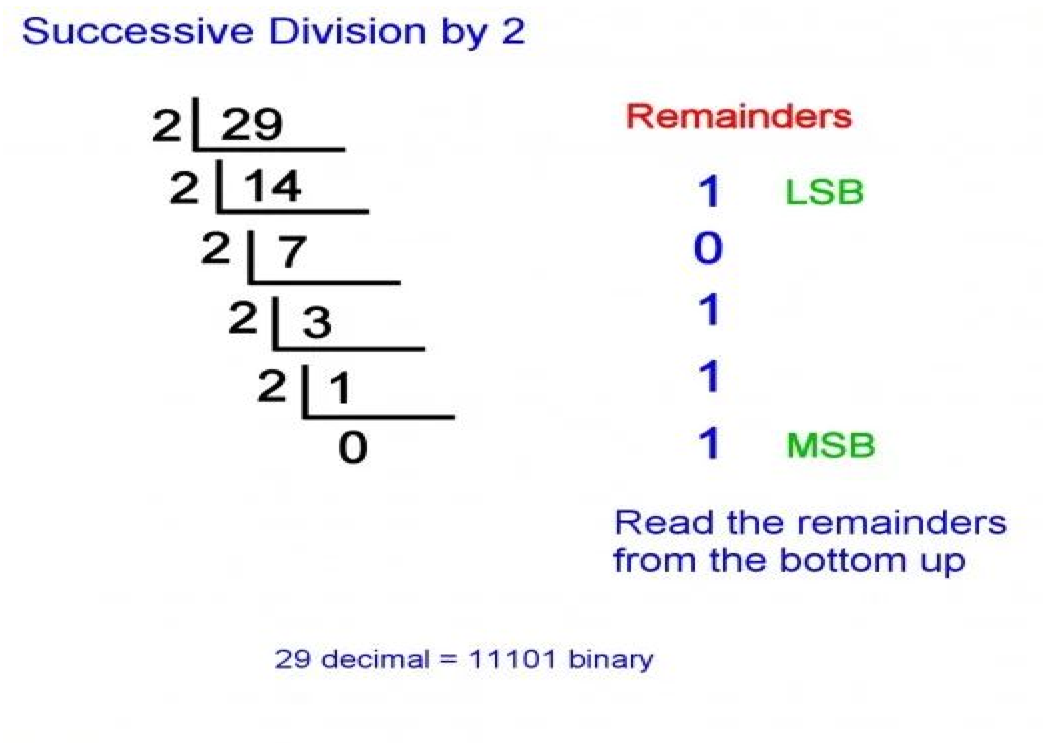
a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

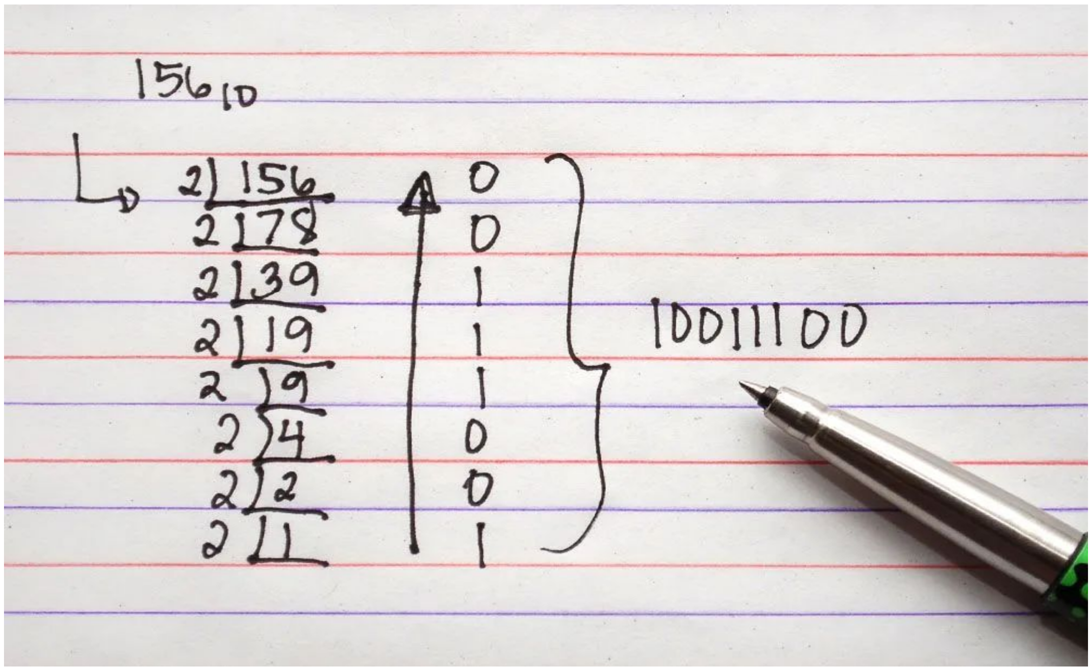
a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

CONVERSION OF DECIMAL TO BINARY :

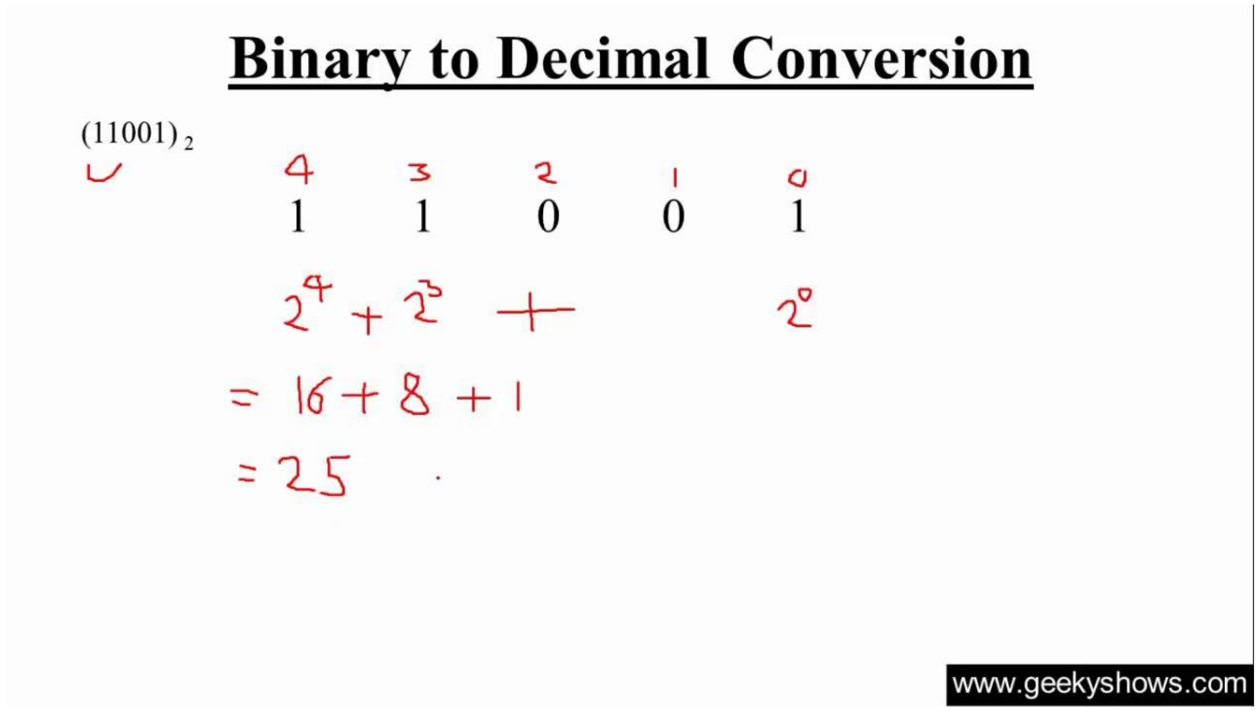






CONVERSION OF BINARY TO DECIMAL:

Convert Binary 11001 to Decimal ?



Binary to Decimal Conversion

$(11001)_2$

4	3	2	1	0
1	1	0	0	1

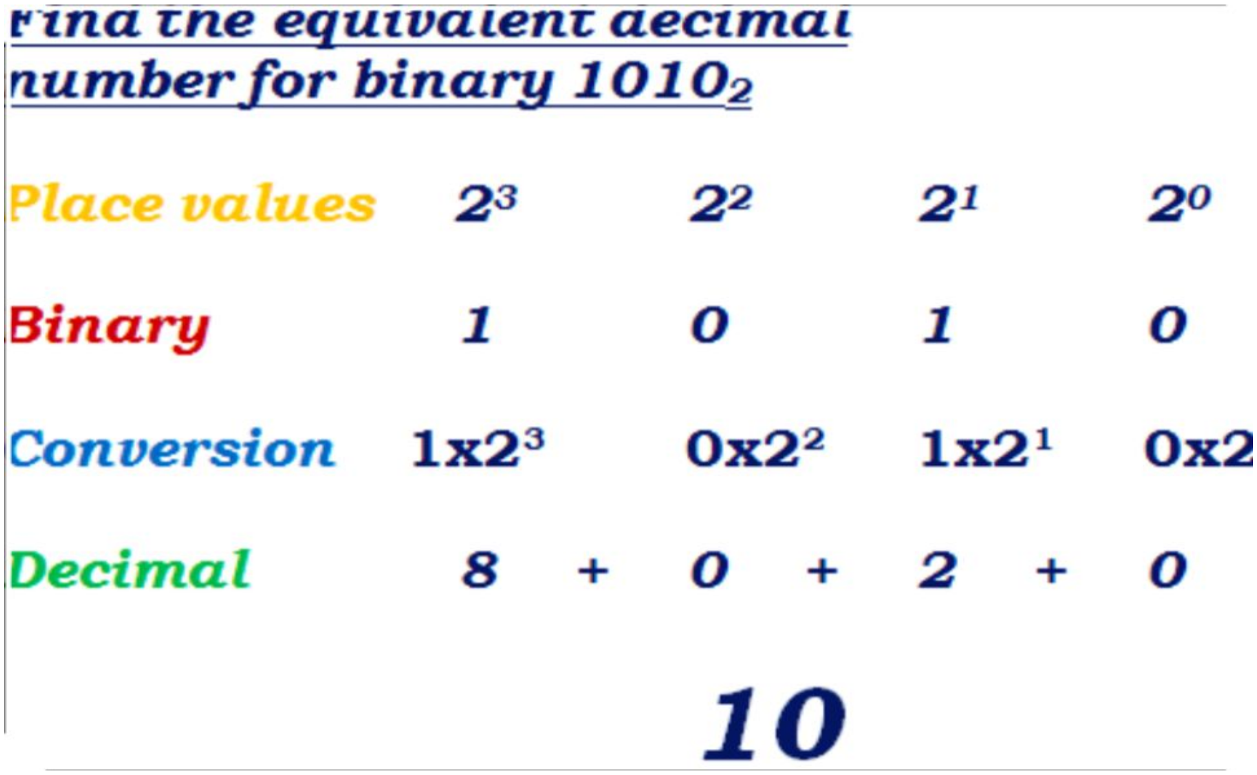
$2^4 + 2^3 + \quad 2^0$

$= 16 + 8 + 1$

$= 25$

www.geekyshows.com

Convert Binary 1010 to Decimal ?

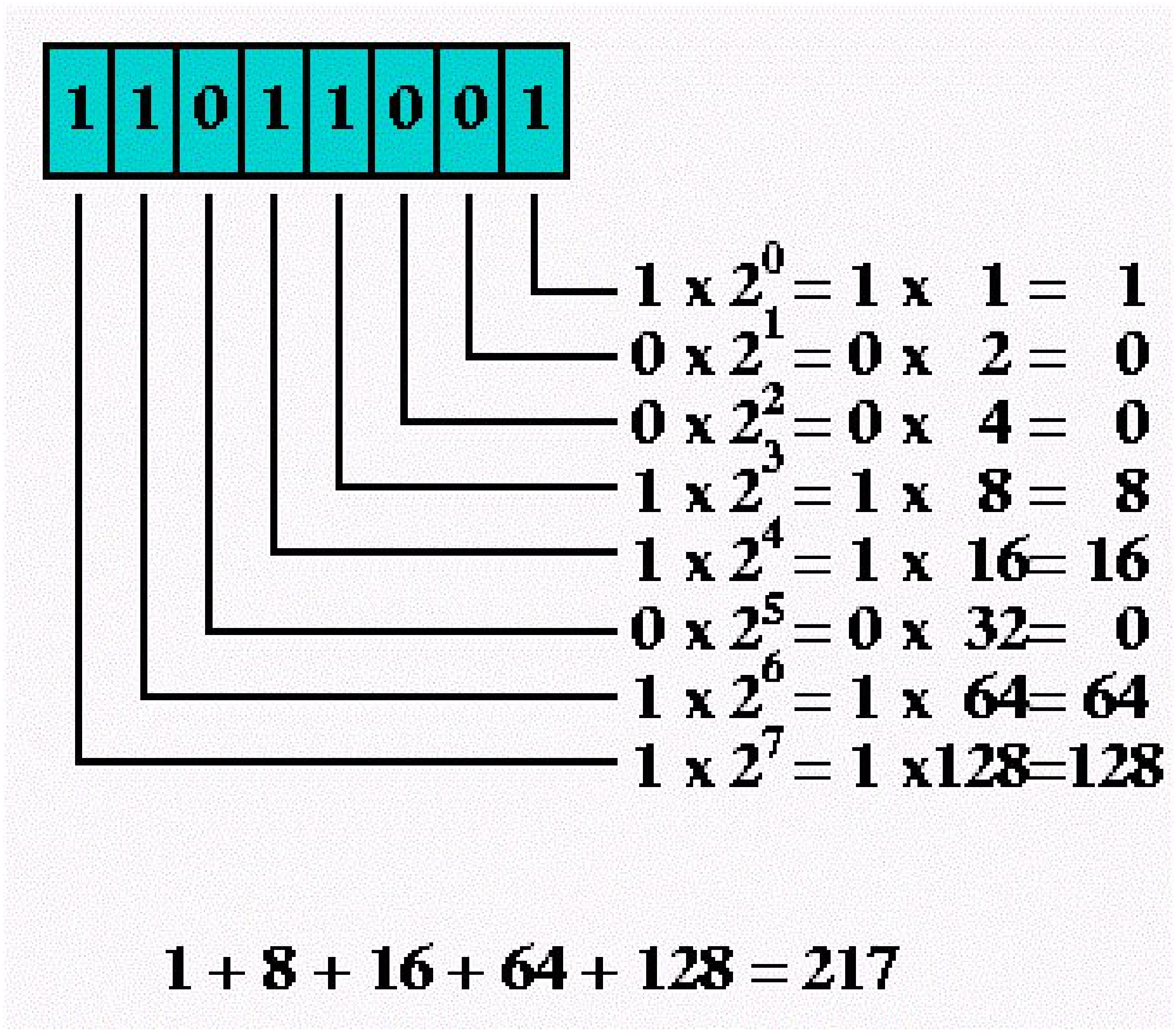


Find the equivalent decimal number for binary 1010_2

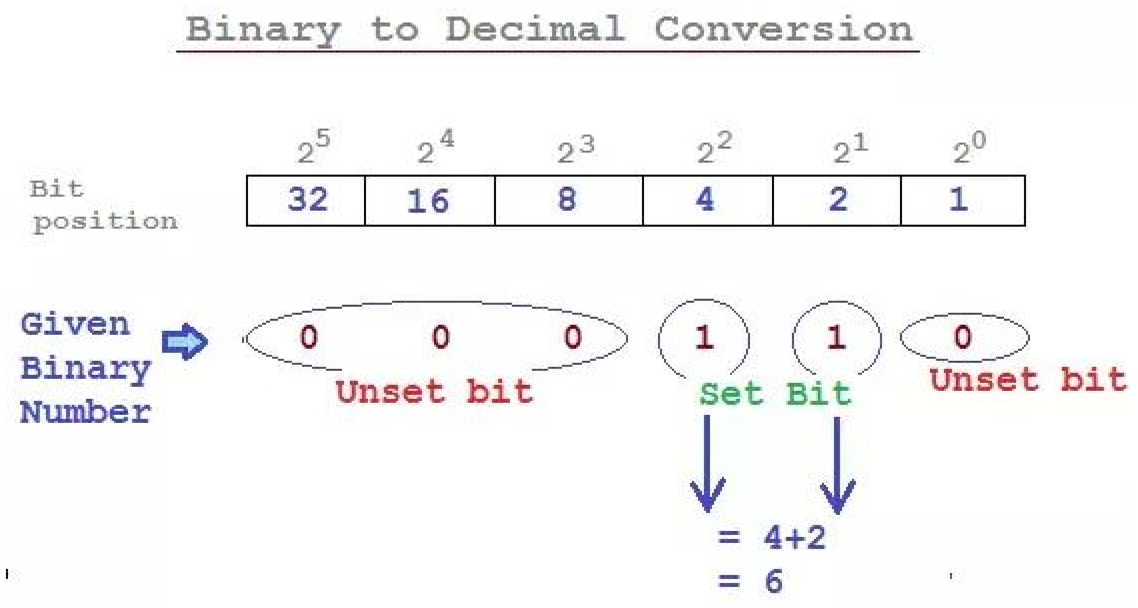
Place values	2^3		2^2		2^1		2^0
Binary	1		0		1		0
Conversion	$1\text{x}2^3$		$0\text{x}2^2$		$1\text{x}2^1$		0x2
Decimal	8	+	0	+	2	+	0

10

Convert Binary 11011001 to Decimal

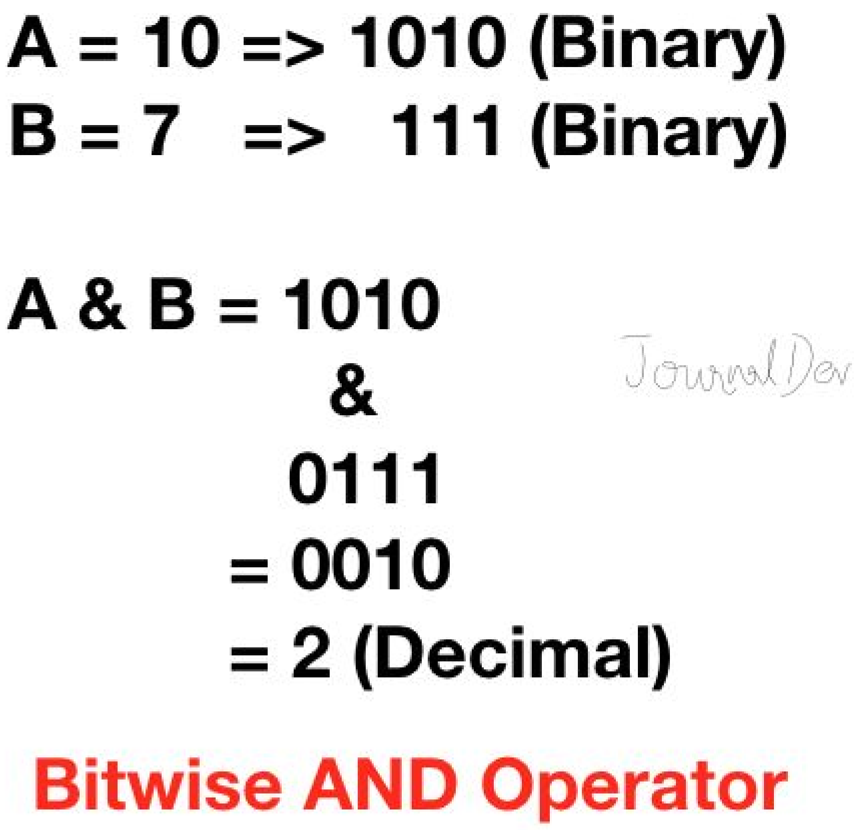

1 1 0 1 1 0 0 1
$1 \times 2^0 = 1 \times 1 = 1$
$0 \times 2^1 = 0 \times 2 = 0$
$0 \times 2^2 = 0 \times 4 = 0$
$1 \times 2^3 = 1 \times 8 = 8$
$1 \times 2^4 = 1 \times 16 = 16$
$0 \times 2^5 = 0 \times 32 = 0$
$1 \times 2^6 = 1 \times 64 = 64$
$1 \times 2^7 = 1 \times 128 = 128$
$1 + 8 + 16 + 64 + 128 = 217$

Shortcut :



Bitwise Operators: The Bitwise operators is used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. The mathematical operations such as addition, subtraction, multiplication etc. can be performed at bit-level for faster processing.
For example, the bitwise AND represented as & operator in C or C++ takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

The & (bitwise AND) in C or C++ takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
The | (bitwise OR) in C or C++ takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
The ^ (bitwise XOR) in C or C++ takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.


A = 10 => 1010 (Binary)

B = 7 => 111 (Binary)

A & B = 1010

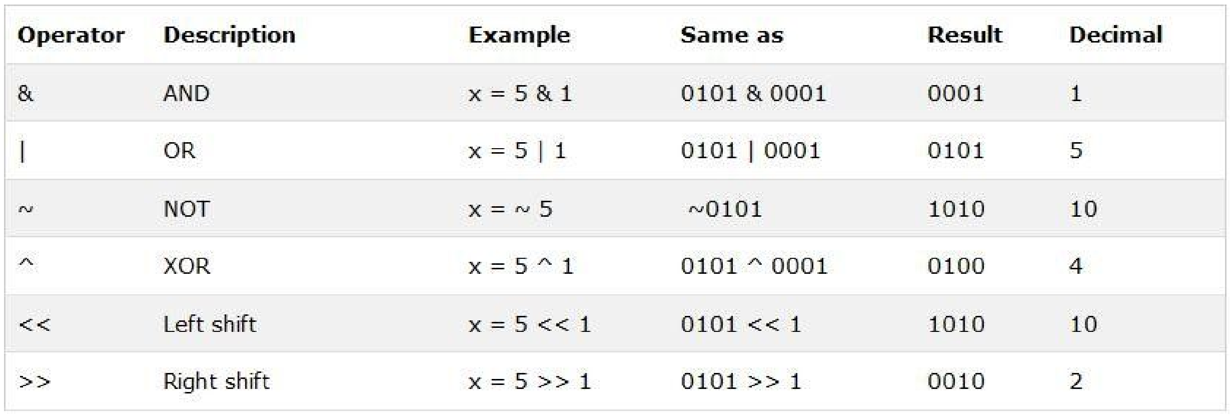
&

0111

= 0010

= 2 (Decimal)

Bitwise AND Operator



Operator	Description	Example	Same as	Result	Decimal
&	AND	x = 5 & 1	0101 & 0001	0001	1
|	OR	x = 5 | 1	0101 | 0001	0101	5
~	NOT	x = ~ 5	~0101	1010	10
^	XOR	x = 5 ^ 1	0101 ^ 0001	0100	4
<<	Left shift	x = 5 << 1	0101 << 1	1010	10
>>	Right shift	x = 5 >> 1	0101 >> 1	0010	2

The bitwise operators should not be used in place of logical operators. The result of logical operators (&&, || and !) is either 0 or 1, but bitwise operators return an integer value. Also, the logical operators consider any non-zero operand as 1.

Assignment Operators: Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value. The value on the right side must be of the same data-type of variable on the left side otherwise the compiler will raise an error.
Different types of assignment operators are shown below:
“=”: This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.

For example:

a = 10;

ch = 'y';

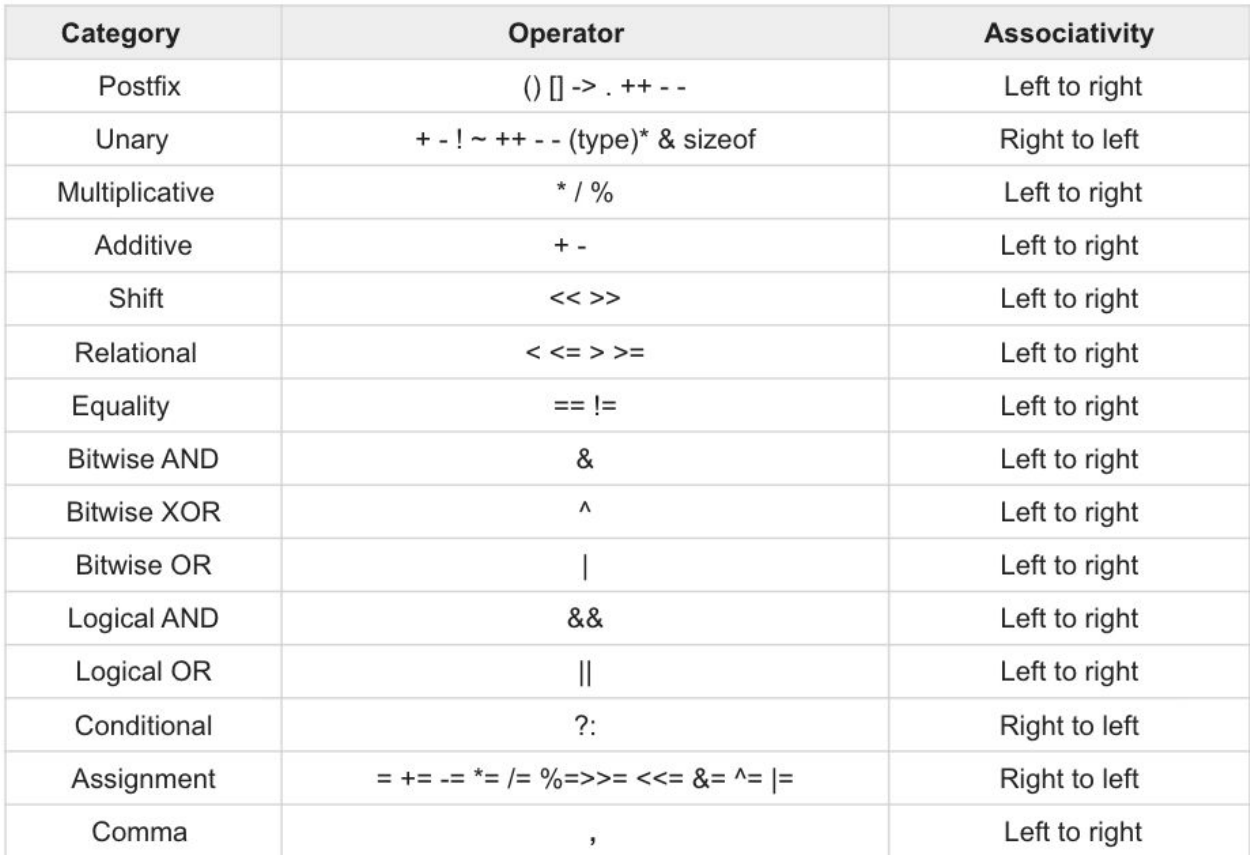
(a += b) can be written as (a = a + b)
If initially value stored in a is 5. Then (a += 6) = 11.

(a -= b) can be written as (a = a - b)
If initially value stored in a is 8. Then (a -= 6) = 2.

(a *= b) can be written as (a = a * b)
If initially value stored in a is 5. Then (a *= 6) = 30.

(a /= b) can be written as (a = a / b)
If initially value stored in a is 6. Then (a /= 2) = 3.

OPERATOR PRECEDENCE CHART



Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Let's solve some practical examples now . Lets Code <>

NExt LIve :

Arrays in C++

TIN RATS
BYE BYE