Operators

Dr. Shweta Sharma
Assistant Professor
Dept. of Computer Engineering
NIT Kurukshetra

Operators

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.

Types of C Operators

Types of Operators	Description
Arithmetic Operators	Perform mathematical calculations like addition, subtraction, multiplication, division and modulus
Assignment Operators	Assign the values for the variables in C programs.
Relational operators	Compare the value of two variables.
Logical operators	Perform logical operations on the given two variables.
Bit wise operators	Perform bit operations on given two variables.
Conditional (ternary) operators	Conditional operators return one value if condition is true and returns another value is condition is false.
Increment/decrement operators	Either increase or decrease the value of the variable by one.
Special operators	&, *, sizeof( ) and ternary operators.

ARITHMETIC OPERATORS IN C

Arithmetic Operators/ Operation	Example
+ (Addition)	A+B
- (Subtraction)	A-B
* (multiplication)	A*B
/ (Division)	A/B
% (Modulus)	A%B

Example showing use of Arithmetic Operator

```
#include <stdio.h>
int main()
{
   int a=40,b=20, add,sub,mul,div,mod;
   add = a+b;
   sub = a-b;
   mul = a*b;
   div = a/b;
   mod = a%b;
   printf("Addition of a, b is : %d\n", add);
   printf("Subtraction of a, b is : %d\n", sub);
   printf("Multiplication of a, b is : %d\n", mul);
   printf("Division of a, b is : %d\n", div);
   printf("Modulus of a, b is : %d\n", mod);
   return 0;
}
```

ASSIGNMENT OPERATORS IN C

Operators	Example/Description
=	test = 10; 10 is assigned to variable test
+=	test += 10; This is same as test = test + 10
-=	test -= 10; This is same as test = test – 10
*=	test *= 10; This is same as test = test * 10
/=	test /= 10; This is same as test = test / 10
%=	test %= 10; This is same as test = test % 10
&=	test&=10; This is same as test = test & 10
^=	test ^= 10; This is same as test = test ^ 10

Example showing use of Assignment Operator

```
# include <stdio.h>

int main()
{
int Total=0,i;
for(i=0;i<10;i++)
{
Total+=i; // This is same as Total = Total+i
}
printf("Total = %d", Total);
return 0;
}
```

RELATIONAL OPERATORS IN C

Relational operators are used to find the relation between two variables. i.e. to compare the values of two variables in a C program

Operators	Example/Description
>	x > y (x is greater than y)
<	x < y (x is less than y)
>=	x >= y (x is greater than or equal to y)
<=	x <= y (x is less than or equal to y)
==	x == y (x is equal to y)
!=	x != y (x is not equal to y)

Example showing use of Relational Operator

```
#include <stdio.h>

int main()
{
   int m=40,n=20;
   if (m == n)
   {
      printf("m and n are equal");
   }
   else
   {
      printf("m and n are not equal");
   }
return 0;
}
```

LOGICAL OPERATORS IN C

Operators	Example/Description
&& (logical AND)	(a>5)&&(b<5) It returns true when both conditions are true
|| (logical OR)	(a>=10)||(b>=10) It returns true when at-least one of the condition is true
! (logical NOT)	!((a>5)&&(b<5)) It reverses the state of the operand "((a>5) && (b<5))" If "((a>5) && (b<5))" is true, logical NOT operator makes it false

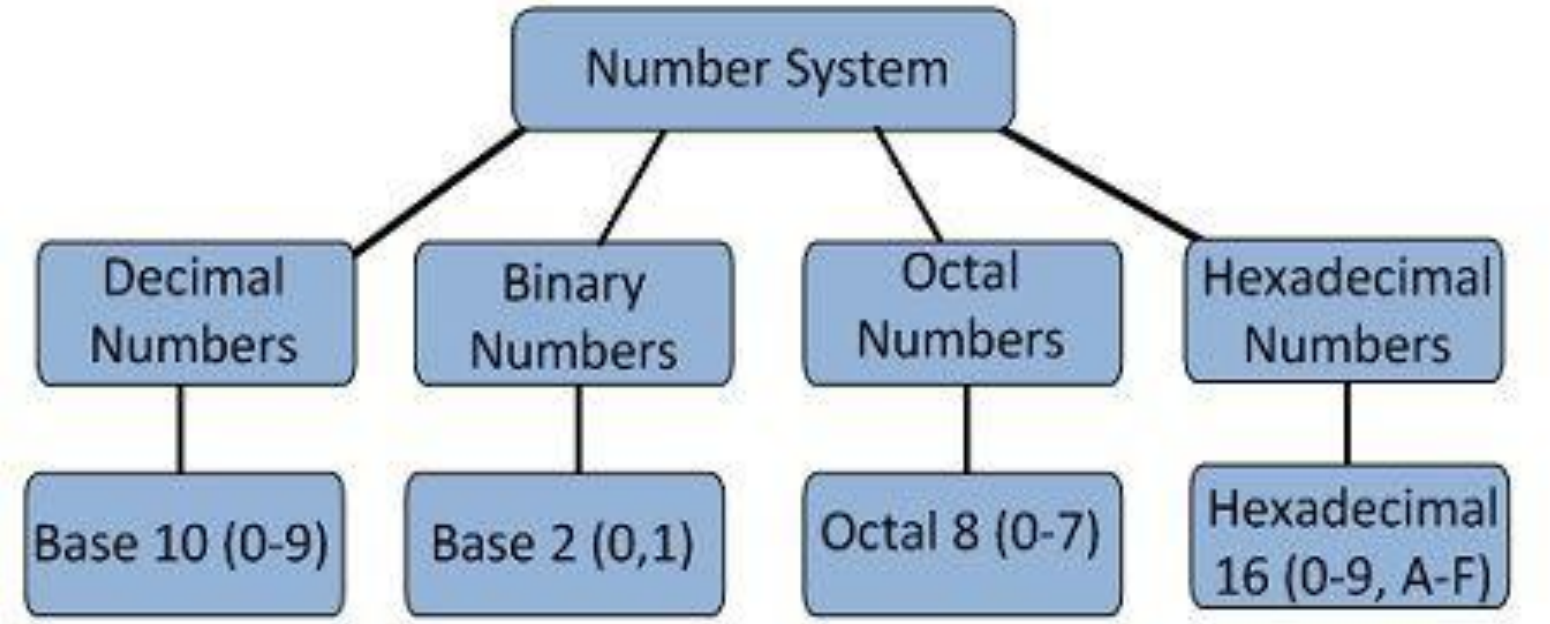
Example showing use of Logical Operator

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  int a=20,p=30;
  if (m>n && m !=0)
  {     printf("&& Operator : Both conditions are true\n");    }
  if (a>p | |  p!=20)
  {     printf("| |  Operator : Only one condition is true\n");     }
  if (!(m>n && m !=0))
  {    printf("! Operator : Both conditions are true\n");   }
  else
  {    printf("! Operator : Both conditions are true. " \
     "But, status is inverted as false\n");
  }
return 0;
}
```

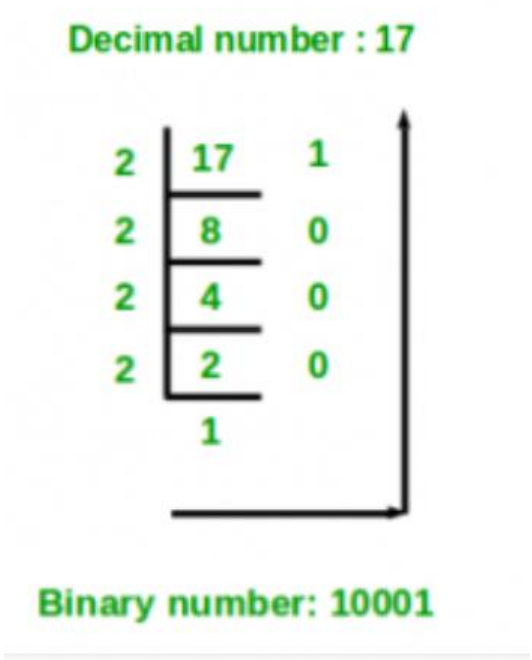
BIT WISE OPERATORS IN C

- These operators are used to perform bit operations.

- Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits


Number System
Decimal Numbers
Binary Numbers
Octal Numbers
Hexadecimal Numbers
Base 10 (0-9)
Base 2 (0,1)
Octal 8 (0-7)
Hexadecimal 16 (0-9, A-F)

Decimal to binary



Fractional Decimal to Binary

Example: number is $(98.46)_{10}$ in decimal

$(1100010.0111)_2$

Binary equivalent of 98 is 1100010_2

Fractional part:

$0.46 \times 2 = 0.92 = 0$

$0.92 \times 2 = 1.84 = 1$

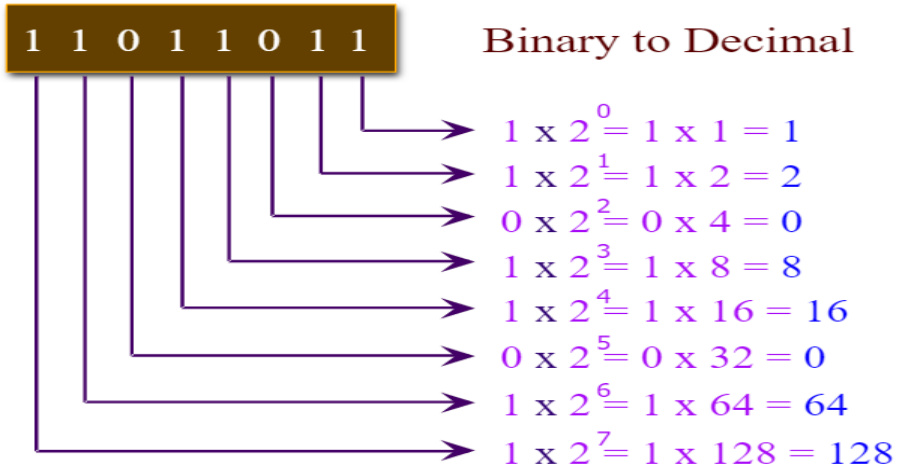
$0.84 \times 2 = 1.68 = 1$

$0.68 \times 2 = 1.36 = 1$

$0.36 \times 2 = 0.72 = 0$

2	98	
2	49	– 0
2	24	– 1
2	12	– 0
2	6	– 0
2	3	– 0
	1	– 1

Binary to Decimal



$1 + 2 + 8 + 16 + 64 + 128 = 219$

$(11011011)_2 = (219)_{10}$

Fractional binary to Decimal

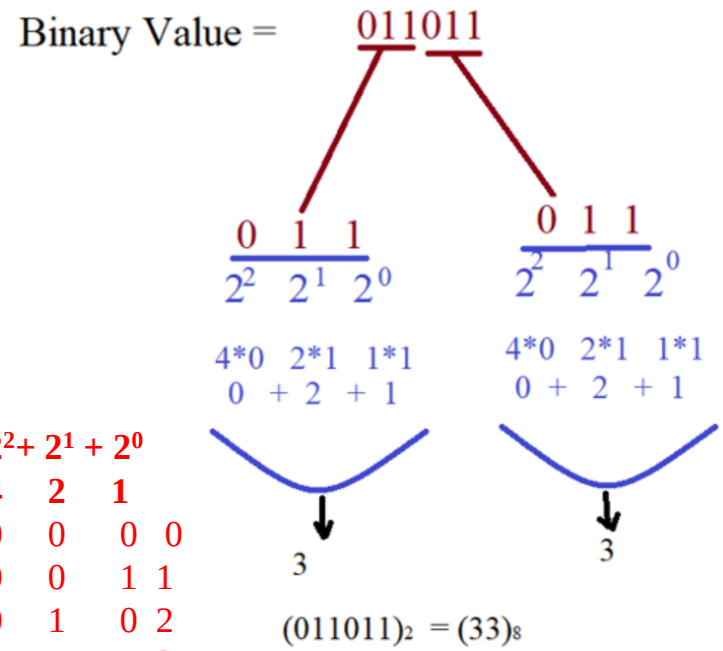
$$(0.11011)_2 = 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5}$$

$$= \frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32}$$

$$= 0.5 + 0.25 + 0 + 0.0625 + 0.03125$$

$$= (0.84375)_{10}$$

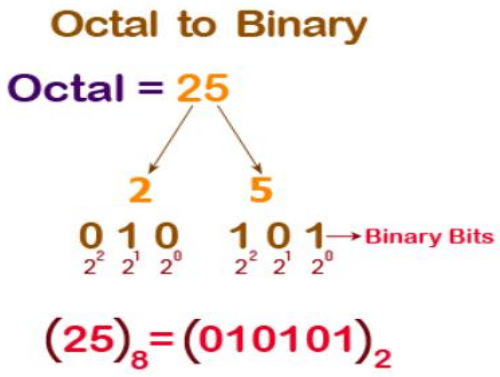
Binary to Octal



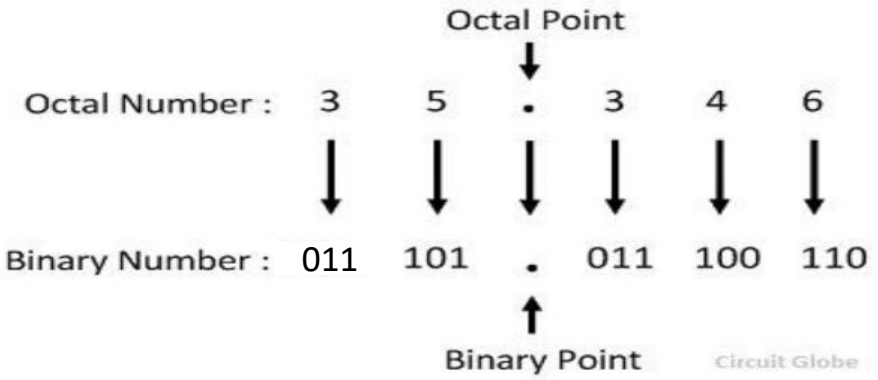
$2^2 + 2^1 + 2^0$

4	2	1	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Octal to Binary

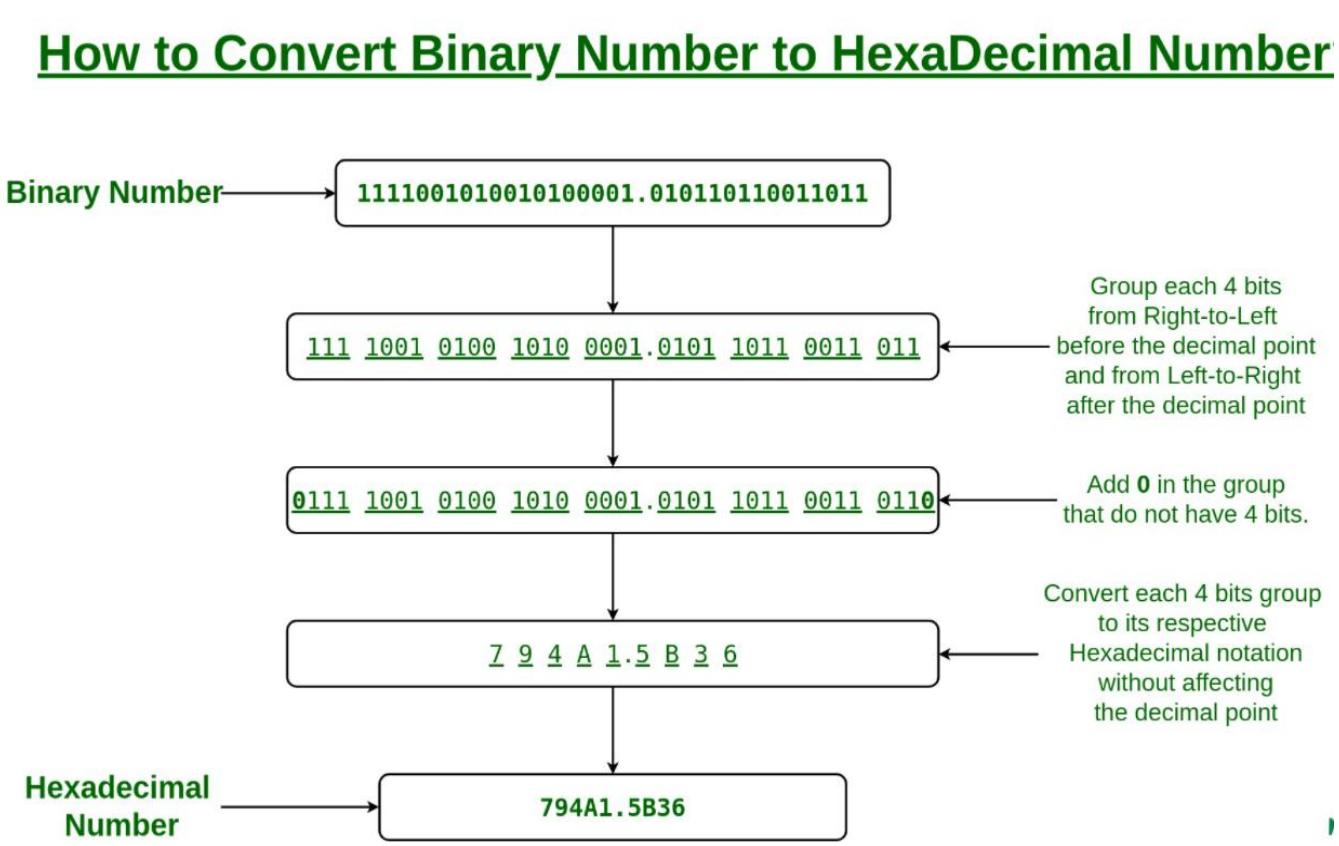


Fractional Octal to Binary



$2^3 + 2^2 + 2^1 + 2^0$

8	4	2	1	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10 A
1	0	1	1	11 B
1	1	0	0	12 C
1	1	0	1	13D
1	1	1	0	14 E
1	1	1	1	15 F



Bitwise operators (&, |, ^, ~, <<, >>)

Bitwise operators modify variables considering the bit patterns that represent the values they store.

operator	asm equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift bits left
>>	SHR	Shift bits right

Program showing Bitwise Operator in C

```
#include <stdio.h>
int main()
{
  int m = 40,n = 80,AND_opr,OR_opr,XOR_opr,NOT_opr ;
  AND_opr = (m&n);
  OR_opr = (m|n);
  NOT_opr = (~m);
  XOR_opr = (m^n);
  printf("AND_opr value = %d\n",AND_opr );
  printf("OR_opr value = %d\n",OR_opr );
  printf("NOT_opr value = %d\n",NOT_opr );
  printf("XOR_opr value = %d\n",XOR_opr );
  printf("left_shift value = %d\n", m << 1);
  printf("right_shift value = %d\n", m >> 1);
}
```

Let a = 5, b = 9

- a&b = 1
- a|b = 13
- a^b = 12
- b<<1 = 18
- b>>1 = 4

Let a = 5, b = 9

1. a&b =
2. a|b =
3. a^b =
4. b<<1 =
5. b>>1 =

& (AND)

5- 00000101

9- 00001001

1- 00000001

| (OR)

5- 00000101

9- 00001001

13-00001101

^ (XOR)

5- 00000101

9- 00001001

12-00001100

<< Left shift

9- 00001001

18-00010010

>> Right shift

9- 00001001

4-00000100

Bitwise Not (~)

- Tilde symbol (~) is used for **bitwise NOT**

```
a=2;
c=~a;
c=-3
```

To solve this You will first convert the value of a that is 2 into binary.

2 in binary is 00000010

You may consider 32 bits also

.....00000000010

(To keep things simple, I am just assuming 8 bits)

Now ~a will be (invert the bits)

11111101

You might think answer is the decimal representation of this value !! But its not

Why??

- **Because if you would have considered 16 bits then answer would have been different !!**
- **And if you would have considered 32 bits then answer would have been different !!**

So what would the answer be??

Bitwise Not in this case gives you a negative number actually **11111101**

See in 11111101, **Most significant bit is 1**, that means a negative number.

So to know the answer, keep the MSB intact and reverse all other bits → you get **10000010**

Now Add 1 to this → 10000010

+1

10000011

required answer.

Here Most significant bit 1 means – (minus) and rest bits are giving the value

So answer is **-3**

Thank You