Bit Wise

- The bitwise operators do not make sense when applied to **double** or **float**.

- Operation on Bits:
  - 1 Word : 16 bits (0xFFFF)
  - 1 Byte : 8 bits (0xFF)
  - 1 Nibble : 4 bits (0xF)
  - Bitwise AND : &
  - Logical AND : &&
  - Bitwise OR : |
  - Logical OR : ||
  - Bitwise and Logical XOR : ^
  - One's Compliment : ~
  - Left Shift : <<
  - Right Shift : >>

- Bitwise Operation:
  - **Set Bit / Switch ON No:**
    - Function Definition:-

```
int setBit (int x, int n)
{
    return (x | (1 << n));
}
```

    - Macro Definition:-

```
#define SET_BIT(x,n)        x|=(1<<n)
```

  - **Reset Bit / Switch OFF No:**
    - Function Definition:-

```
int resetBit (int x, int n)
{
    return (x & (~(1 << n)));

}
```

    - Macro Definition:-

```
#define RESET_BIT(x,n)      x&=(~(1<<n))
```

  - **Toggle Bit / Invert No:**
    - Function Definition:-

```
int toggleBit (int x, int n)
{
    return (x ^ (1 << n));
}
```

    - Macro Definition:-

```
#define TOGGLE_BIT(x,n)     x^=(1<<n)
```

- **Masking No:**
  - A = 0x34A7 => A = A & 0xFF0F => A = 0x3407
  - B = 0x37 => B = B & 0xEC => B = 0x24

- **Masking with XOR:**
  - var = var ^ 0xHEX; (Encryption)
  - var = var ^ 0xHEX; (Decryption)

- Logic of various Bitwise problems:
  - **Even Odd Number:-**

```
mask = 0x1;
if((number & mask) == 0)
        printf("Even No\n");
else
        printf("Odd No\n");
```

  - **Hex to Bin:-**

```
for(i = 15; i >= 0; i--)        // For Two Byte no. loop range is from 0 – 15.
{
        mask = 0x1;
        if((num & (mask << i)) == 0)
                printf("0");
        else
                printf("1");
}
```

  - **Swap Two Numbers:-**

```
int swap (int *num1, int *num2)
{
        *num1  = *num1 ^ *num2;
        *num2  = *num1 ^ *num2;
        *num1  = *num1 ^ *num2;
}
```

  - **Swap Nibbles:-**

```
int swapNibbles (int input)
{
        return (((input & 0x0F) << 4 ) | ((input & 0xF0) >> 4));
}
```

  - **Swap Even Odd Place:-**

```
int evenOddSwap (int input)
{
        return (((input & 0xAAAA) >> 1) | ((input & 0x5555) << 1));
}
```

Bit Field

Without using Bit Field

```
struct date
{
   int day;
   int month;
   int year;
}date;
```

- Size of date: **12** bytes

With using Bit Field

```
struct date
{
   int  day      : 5;
   int  month  :  4;
   int year : 16;
}date;
```

- Size of date: **4** bytes

Without using Bit Field

```
struct ops
{
   int op1;
   int op2;
   int op3;
   int op4;
   int op5;
}ops;
```

- Size of ops: **20** bytes

With using Bit Field

```
struct ops
{
   int op1     : 5;
   int op2     : 4;
   int op3     : 30;
   int op4     : 7;
   int op5     : 16;
}ops;
```

- Size of ops: **12** bytes

Without using Bit Field

```
struct ops
{
   int op1;
   int op2;

   int op3;
   int op4;
}ops;
```

- Size of ops: **16** bytes

With using Bit Field

```
struct ops
{
   int op1     : 5;
   int op2     : 4;
   int         : 0;    // Use new memory location
   int op3     : 12;
   int op4     : 23;
}ops;
```

- Size of ops: **12** bytes

Without using Bit Field

```
struct ops
{
   char op1;
   int  op2;
   char op3;
   int  op4;
   char op5;
}ops;
```

- Size of ops: **16** bytes

With using Bit Field

```
struct ops
{
   char op1     : 5;
   int  op2     : 14;
   char op3     : 7;
   int  op4     : 11;
   char op5     : 3;
}ops;
```

- Size of ops: **8** bytes