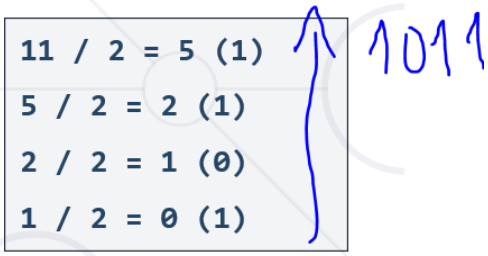
Литерал – начинът, по който изписваме съответния тип променлива / стойност = начинът по който компютъра го разбира

Позционна бройна система – има позиции

- Binary to Decimal

$1011_b = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 =$
$= 1*8 + 0*4 + 1*2 + 1*1 =$
$= 8 + 0 + 2 + 1 =$
$= 11$

- Decimal to Binary



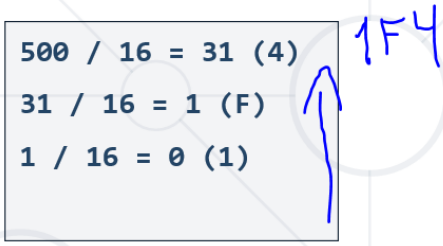
let binaryNum = **0b**11011;

let hecadecimalNum = **0x**11011;

- Hexadecimal to Decimal

$1F4_{hex} = 1*16^2 + 15*16^1 + 4*16^0 =$
$= 1*256 + 15*16 + 4*1 =$
$= 256 + 240 + 4 =$
$= 500$

- Decimal to Hexadecimal



Most significant bit – най-голямата позицията на число

Least significant bit – най-малката позиция на число

Representation of Integer Numbers

- Positive **8-bit** numbers have the format **0XXXXXXX**
  - The value is the decimal value of their last **7 bits** (**XXXXXXX**)
- Negative **8-bit** numbers have the format **1YYYYYYY**
  - The value is **128**(2^7) minus the decimal value of **YYYYYYY**

$$-2^7$$

$$10010010_b = -(2^7 - 10010_b) = -(128 - 18) = -110$$

Positive and Negative Integers

- The largest 8-bit integer is:

$$127 = (2^7 - 1) = 01111111_b$$

- The smallest negative 8-bit integer is:

$$-128 = (-2^7) = 10000000_b$$

$$2^7$$

- The largest 32-bit integer is:

$$2147483647 = (2^{31} - 1) = 0111\ldots1111_b$$

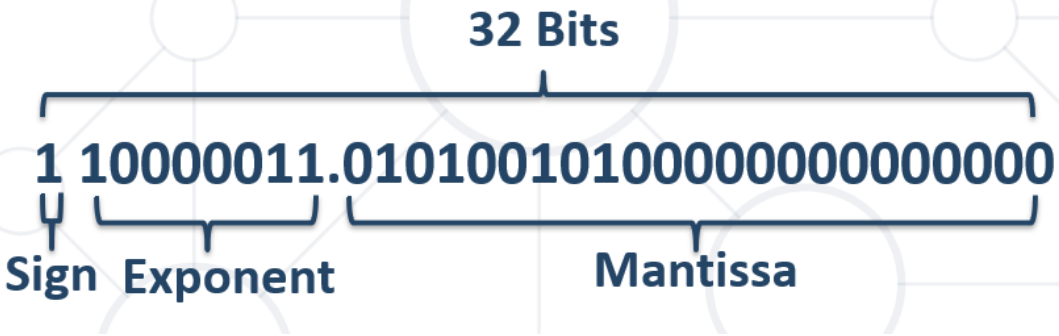
- The smallest negative 32-bit integer is:

$$-2147483648 = (-2^{31}) = 1000\ldots0000_b$$

$$2^{31}$$

Storing Floating-Point Numbers

- Sequence of bits
- Consists of **sign bit**, **exponent** and **mantissa**



- System that uses **binary numbers** (**0** and **1**) to represent chars
  - Letters, numerals, etc.
- In the **ASCII** each character consists of **8 bits**
- In the **Unicode** encoding each character consists of **16 bits**

Binary	Decimal	Character
01000001	65	A
01000010	66	B

- **String** is an **array of characters**

- Characters in the string can be:
  - 8 bit (ASCII)
  - 16 bit (UTF-16)

Bitwise Operators

- Bitwise operator ~ turns all 0 to 1 and all 1 to 0
  - Like ! for boolean expressions but bit by bit
- The operators |, & and ^ behave like ||, && and ^ for boolean expressions but bit by bit
- Behavior of the operators |, & and ^:

Operator	|	|	|	&	&	&	^	^	^
Operand	0	1	1	0	1	1	0	0	1
Operand2	0	0	1	0	0	1	0	1	1
Result	0	1	1	0	0	1	0	1	0

Bitwise Operators Examples

- Bitwise NOT (~)

```
5      //0101
~5     //1010
```

- Bitwise OR (|)

```
5      //0101
3      //0011
5 | 3  //0111
```

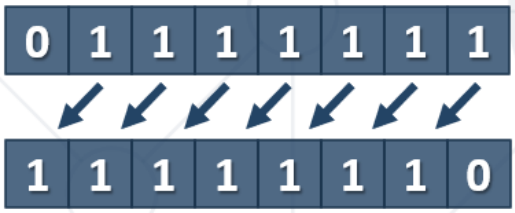
- Bitwise AND (&)

```
5      //0101
3      //0011
5 & 3  //0001
```

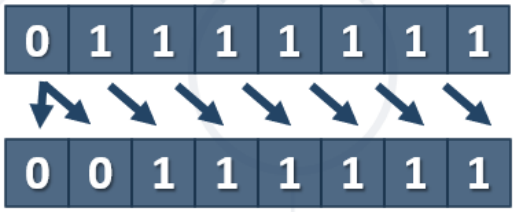
- Bitwise XOR(^)

```
5      //0101
3      //0011
5 ^ 3  //0110
```

- Bits that are shifted out of either end are discarded
- Left Arithmetic Shift (<< operator)



- Right Arithmetic Shift (>> operator)



В JavaScript ползваме:
От двоична в десетична система – по краткия начин
let number = '1101';
parseInt(number, 2); - 13

От десетична в двоична система – по краткия начин
let num = 2;
num.toString(2); - "10"

Simple Bitwise Operations

- How to get the bit at position **p** from a number **n**

```
p = 5                 //00000101
n = 125               //01111101
125 >> p              //00000011 = 3
3 & 1                 //1
```

- How to set the bit at position **p** to **0** or **1**

```
p = 5                 //00000101
n = 125               //01111101
mask = ~(1 << p)      //00100000
result = n & mask     //01011101
```

```
p = 5                 //00000101
n = 125               //01111101
mask = 1 << p         //00100000
result = n | mask     //01111101
```