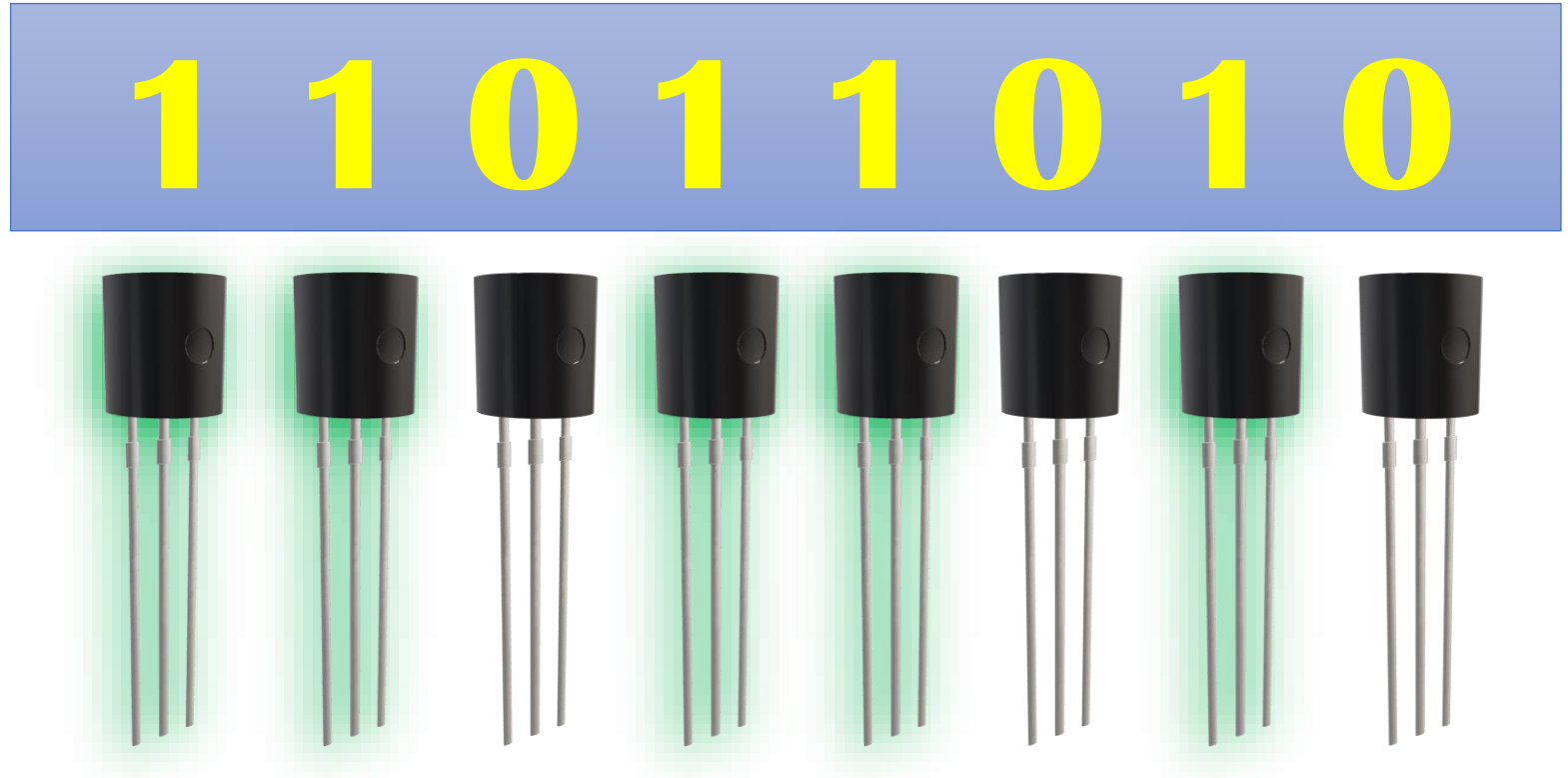


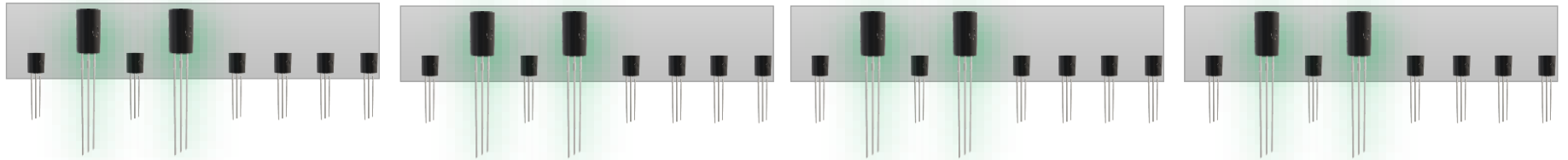
computer  
storage  
unit



8 transistors represent 8-bit (bit)  
binary numbers  
Tied together is a Byte

C++ programs need to occupy "memory" when they are running. Like files stored on disk, this "memory" is also limited in size.

`int a;`



`a=7;`



Through the different states of 32 transistors, one int can hold different 32-bit binary digit. And the first binary bit is sign bit 0 means the number is positive and 1 means negative.

The largest integer that the int type can hold is 01111111 11111111 11111111 11111111, which is  $2^{31}-1$ , which is 2147483647

The smallest integer that the int type can hold is 10000000 00000000 00000000 00000000, which is  $-2^{31}$ , which is -2147483648. Note that negative numbers : the actual integer represented is the binary number are inverted and then +1, except the sign bit.

# int

```
cout<<INT_MAX;
```

```
cout<<INT_MIN;
```

```
cout<<0x7fffffff;
```

```
// 0x means base-16 number, 7fffffff in hexadecimal is binary 01111111 11111111 11111111 11111111
```

```
cout<<0xffffffff;
```

这个是多少？

# Binary

- Representation: B, 0b, small number subscript
- For example, 0b101101, B101101, 101101<sub>2</sub>
- In the computer, 8 bits are one Byte, that is, from 0b0000 0000 to 0b1111 1111. The largest 8-bit binary number is 255. One Byte can represent a total of 256 different numbers from 0 to 255.

The Math of Base

Note: Now it's math!

# positional notation or place-value notation

- decimal integer is to use decimal method to represent numbers:

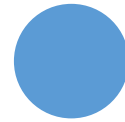
$$236 = 2 * 10^2 + 3 * 10^1 + 6 * 10^0$$

- Pay attention to the meaning of each digit of 236. The first digit is the number 2, which means that there are 2  $10^2$
- Decimal after decimal separator

$$236.236 = 2 * 10^2 + 3 * 10^1 + 6 * 10^0 + 2 * 10^{-1} + 3 * 10^{-2} + 6 * 10^{-3}$$

# Binary integer

- Here, start with 0b to indicate that this is in binary form
- Same place-value, pay attention to the meaning of each digit of the binary number
- $0b111.0101 = 1*2^2 + 1*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4}$
- Convert this binary to decimal
  - $1*2^2 + 1*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4} = 7.3125$
  - $2^{-4} = (2^4)^{-1} = 1/(2^4) = 1/16 = 0.0625$





# How to convert decimal to binary

- Convert from decimal to binary.

$$236 = \_ * 2^0 + \_ * 2^1 + \_ * 2^2 + \_ * 2^3 + \_ * 2^4 + \_ * 2^5 + \_ * 2^6 + \_ * 2^7 + \dots$$

- First look at how many high position value there are, such as  $2^7$  (at most one), then  $2^6$ ...
- Or the other way: how the number 236 in decimal gets each digit 6,3,2 from the back to the front? Because it is full of 10, the remainder obtained by %10 is the number of the current last digit less than 10. /10 three times (equal to /1000) is 0, so it's a three-digit number.
  - $236 \% 10 = (23 * 10 + 6) \% 10 = 23 * 10 \% 10 + 6 \% 10 = 6$  the value on position  $10^0$  is 6
  - $236 / 10 = 23$   $236 / 10$  gets 23, there are 23  $10^1$  in 236
  - $23 \% 10 = 3$  Among the 23  $10^1$ , there are 3  $10^1$  that are not enough to make up  $10^2$ , so value in position  $10^1$  is 3
  - $23 / 10 = 2$  there are 2  $10^2$  in 23  $10^1$
  - $2 \% 10 = 2$  2  $10^2$ , not enough  $10^3$ , so place value at  $10^2$  is 2
  - $2 / 10 = 0$  there are 0  $10^3$  in 2  $10^2$



The only difference between binary and decimal is that the base is different.

---

$$236 \% 2 = 0 \quad 236 / 2 = 118$$

---

$$118 \% 2 = 0 \quad 118 / 2 = 59$$

---

$$59 \% 2 = 1 \quad 59 / 2 = 29$$

---

$$29 \% 2 = 1 \quad 29 / 2 = 14$$

---

$$14 \% 2 = 0 \quad 14 / 2 = 7$$

---

$$7 \% 2 = 1 \quad 7 / 2 = 3$$

---

$$3 \% 2 = 1 \quad 3 / 2 = 1$$

---

$$1 \% 2 = 1 \quad 1 / 2 = 0$$

---

$$236 = 0b \ 1110 \ 1100$$

# How to convert the fractional part?

---

- What is the decimal representation of 0.125 in binary?
- Or think about how the number 0.236 in decimal can get the 3 digit 2, 3, and 6 after the decimal point?
- $0.236 / (10^{-1}) = 0.236 / (1/10) = 0.236 * 10 = 2.36$  Rounding it, it means that there are 2  $10^{-1}$ , so the first place after the decimal point is 2. means  $2 * 10^{-1}$
- Remove these 2  $10^{-1}$  from 0.236 now leaves  $0.36 * 10^{-1}$ , how many  $10^{-2}$  are there? Still the same method  $0.36 / (10^{-1}) = 0.36 * 10 = 3.6$ . It means that there are 3  $10^{-2}$ , so the second decimal place is 3. means  $3 * 10^{-2}$

# So, get binary after decimal point

---

- Continue doing  $\times 2$  rounded up (actually  $/2^{-1}$ , then leave the fractional part)

$0.125 \times 2 = 0.25$  rounded to 0, there are 0  $2^{-1}$  0.0 and 0.25 remaining  $2^{-2}$

$0.25 \times 2 = 0.5$  rounded to 0, there are 0  $2^{-2}$  0.00 and 0.5 remaining  $2^{-3}$

$0.5 \times 2 = 1$  rounded to 1, there is 1  $2^{-3}$  0.001 and no remaining

- Note that when binary represents a number:
  - Each digit from left to right after the decimal point indicates how many  $2^{-1}$ ,  $2^{-2}$ ,  $2^{-3}$ ...
  - Each digit from right to left before the decimal point indicates how many  $2^0$ ,  $2^1$ ,  $2^2$ , ...

# The particularity of conversion between binary to Octal(base 8) and Hexadecimal(base 16)

- Binary to hexadecimal: 4 digits together

$0001(B) = 1_{16}$     $0010(B) = 2_{16}$     $0011(B) = 3_{16}$     $0100(B) = 4_{16}$

$0101(B) = 5_{16}$     $0110(B) = 6_{16}$     $0111(B) = 7_{16}$     $1000(B) = 8_{16}$

$1001(B) = 9_{16}$     $1010(B) = A_{16}$     $1011(B) = B_{16}$     $1100(B) = C_{16}$

$1101(B) = D_{16}$     $1110(B) = E_{16}$     $1111(B) = F_{16}$

- Binary to Octal: 3 digit together

$001(B) = 1_8$     $010(B) = 2_8$     $011(B) = 3_8$     $100(B) = 4_8$     $101(B) = 5_8$     $110(B) = 6_8$

$111(B) = 7_8$

- For example, 0001 0101(B) to hexadecimal?

# practice

How to represent 176 to binary

0b10110000

What is 0b101010 to decimal?

42

What is 0b101010 to hexadecimal?

2A<sub>16</sub>

# What is 0b101010 to decimal?

- Processing by digits from left to right: what is the difference between 0b10 and 0b1?  
 $0b10 = 2 * 0b1$ . What is the difference between 0b11 and 0b1?  $0b11 = 0b1 * 2 + 1$

1	0	1	0	1	0	0
1						$0 * 2 + 1 = 1$
1	0					$1 * 2 + 0 = 2$
1	0	1				$2 * 2 + 1 = 5$
1	0	1	0			$5 * 2 + 0 = 10$
1	0	1	0	1		$10 * 2 + 1 = 21$
1	0	1	0	1	0	$21 * 2 + 0 = 42$

# Binary Operations in C++


- In C++, although we see decimals by default for each int, the internal storage in the computer is binary (it can be understood that the displayed decimals are converted).
- the char type, a byte, is also binary, but it is translated according to the ANSI code when displayed.
- It will be faster to do some binary operations directly

## binary operator: operate on bit

- Note that it is a bit operation, the operation of binary representation
- ^ XOR : 0 if 2 bits are equal; otherwise, 1
- & : 1 if 2 bits are all 1; otherwise, 0
- | : 0 if 2 bits are all 0; otherwise, 1
- << shift left
- >> shift right

operator	asm equivalent (汇编)	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





Char type variable is a byte, which is an integer less than 256 (8 binary bits), but when we output it, if it is char, it will be translated into characters according to ASCII code and displayed. What if you use int instead of char? int is 4 bytes, modify it to see the result, why?

---

```
char a, b; // Note that according to binary, the high bits are filled with 0 if no exist
a = 123; // In binary it is 01111011
b = 23; // // In binary it is 00010111 (same as 10111, but as char has 8 bits, fill 0 for first 3 bits.)
cout<<"a&b "<<(a&b)<<endl;//00010011
cout<<"a|b "<<(a|b)<<endl;//01111111
cout<<"a^b "<<(a^b)<<endl;//01101100
cout<<"a>>1 "<<(a>>1)<<endl;//00111101
cout<<"a<<1"<<(a<<1)<<endl;//11110110
```

& | is like the logical operator? 1&1 results in 1 , 1&&1 results in 1 (true)

However, one is based on bits, and the other is judged by logical values. For example, the result of 2&1 is 0, but the result of 2&&1 is 1 (true)

& | ^

- First look at binary representation, then align with the bit, add 0 if there is none, and then operate bitwise, 123 binary is 01111011 , 23 binary is 00010111

01111011

01111011

01111011

& 00010111

| 00010111

^ 00010111

---

00010011

01111111

01101100

>>

<<

Be careful not to  
confuse with  
cout cin

- First look at the binary representation, because inside the computer, virtually every integer is composed of binary 1s and 0s. It's just displayed in decimal form.
- 123 in binary is 01111011
- 23 binary is 00010111
- $123 \gg 1$  is the binary 01111011 shifted to the right by 1 bit, the highest bit is removed, so 0 is added, and the lowest bit is squeezed out. This operation is like we do  $123/2$ .

01111011 → 00111101 (61)

- $123 \ll 1$  means the binary 01111011 is shifted left by 1 bit, the lowest bit is removed, so 0 is added, and the highest bit is squeezed out. This operation is like we do  $123*2$

01111011 → 11110110 (246)

# Specific digit in binary

- $a \& 1$   $a \& 2$  ?  $a | 1$   $a | 2$  ?
- $1 \ll n$  ?

$1 \ll 2$  result is  $0b100$ , which is  $2^2$

- So, how to get a specific digit in binary?

int a=19873, how to get the 5<sup>th</sup> digit from lower to higher in binary?

way1:  $a \& (1 \ll 4)$

way2: loop  $a /= 2$  4times, then  $a \% 2$  is the answer.

# XOR any positive integer with 0, the result is itself

---

- Like accumulators. If many numbers are XOR together, the initialization should be 0
- XOR is addition of each bit without a carry.  $0+1=1$ ;  $0+0=0$ ;  
 $1+1=0$ ;
- What about the result of the XOR of any two identical numbers?

## homework1: do it by hand

1) The sum of the binary numbers 00101100 and 00010101 is ( ) 。

A. 00101000 B. 01000001 C. 01000100 D. 00111000

2) In binary,  $1011001 + ( ) = 1100110$  。

A . 1011      B . 1101      C . 1010      D . 1111

3) Exercises of Binary Operations in C++: What should be the output of the following sentences? Programs can also be used to verify that your manual calculations are correct.

```
cout<< (21&10) <<endl;
```

```
cout<<(21|10)<<endl;
```

```
cout<<(21^10)<<endl;
```

```
cout<< (21<<2) <<endl;
```

```
cout<< (21>>2) <<endl;
```

# Homework 2: (xor.cpp)

- Output no more than 1000 positive integers in the file and get their XOR sum. All integers are less than  $10^9$ .
- Input file: xor.in
- Output file: xor.out
- Input format: an integer  $n$  ( $1 \leq n < 1000$ ) in the first line,  $n$  positive integers separated by spaces in the second line.
- Output format: one line, one integer, the result of the XOR of these  $n$  positive integers.
- Sample input

3

11 21 99

- Sample output :

125

- Example description:  $11 \oplus 21 \oplus 99 = 125$

## Homework 3 findalone.cpp can you do it without array?

- Given  $n$  ( $1 \leq n \leq 2 \cdot 10^5$ ) integers, it is known that among the  $n$  integers, except for one "lonely number", other numbers will appear twice, please output this "lonely number", That is, the number that only occurs once. All integers are greater than  $-10^9$  and less than  $10^9$ .

- Input format:

- Two lines, the first line contains a positive integer  $n$ . The second line contains  $n$  integers, separated by spaces. All but one of the "lone numbers" appear twice.

- Output format: a positive integer, representing the number of 1s when  $n$  is represented in binary.

- Sample input

11

11 9 8 7 4 3 11 9 8 7 4

- Sample output

3

- Sample description: Obviously, only 3 of the 11 numbers in this input appear only once, so he is a "lone number" and outputs 3.

- Hint: The binary XOR (^) of any integer with itself is 0, and the XOR of any integer with 0 is the integer itself.



# Try this ( Example questions for the next lesson, can skip )

getnum.cpp

Enter a string with a length of not more than 100000. It is known that this string contains a unique positive integer. Please output the result of this integer modulo 147.

Input format: one line, a string without spaces, the length of the string does not exceed 1000. It contains a unique positive integer. Note:

"123@#", "###123", "###123.d" are all valid input strings, and "\$1#22.." this is Invalid string because there are two integers. You only need to deal with valid input strings.

Output format: an integer, which is the result of modulo 147 by the only positive integer contained in the string.

Sample input

%.1473.abf

Sample output

3

Example description: The only positive integer in this string is 1473, and the result of  $1473 \% 147$  is 3

How to convert binary numbers to decimal? From the highest bit onwards, see 1 bit and add it to the previously converted number.

由易到难，思维体系训练  
实战结合，创新协作培养  
兴趣导向，未来职业引领

<https://www.35tang.com>



扫码关注公众号

<https://www.三五堂.com>



添加辅导老师