

整数的表示

无符号整数

- 无符号整数：**① 全部二进制位都是数值位，没有符号位，第 i 位的位权是 2^{i-1}
② n bit 无符号整数表示范围 $0 \sim 2^n - 1$ ，超出则溢出，意味着该计算机无法一次处理这么多
③ 可以表示的最小的数 全0，可以表示的最大的数 全1。

计算机硬件如何做无符号整数的加法：从最低位开始，按位相加，并往更高位进位

计算机硬件如何做无符号整数的减法：

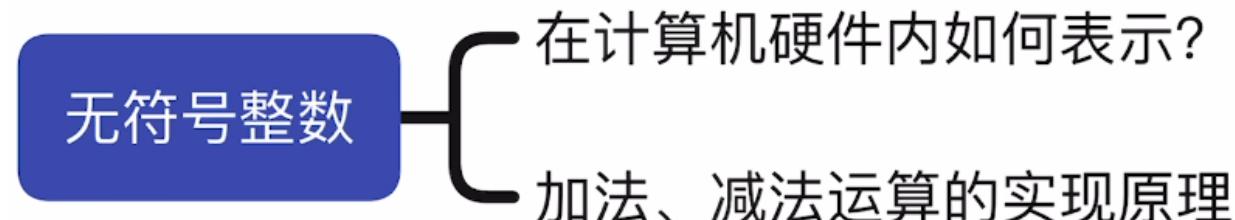
- ① “被减数”不变，“减数”全部位按位取反、末位+1，减法变加法
② 从最低位开始，按位相加，并往更高位进位

无符号整数，即“自然数”，0、1、2、3、4...

C语言中的无符号整数

```
unsigned short a = 1; //无符号整数（短整型，2B）  
unsigned int b = 2; //无符号整数（整型，4B）
```

- 无符号整数，在计算机硬件内，如何表示？
- 无符号整数的加法、减法运算是怎么用硬件实现的？



无符号整数的表示

无符号整数的表示

该计算机硬件能支持的无符号整数位数有上限

通用寄存器只能存8位

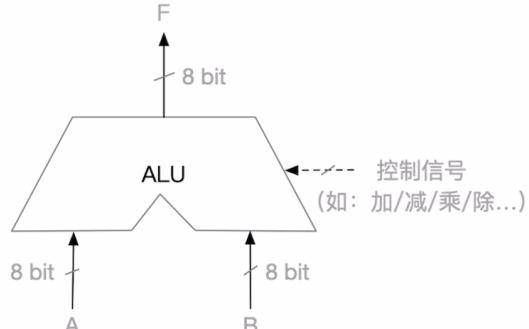
Tips: 现在的个人计算机机器字长通常是64位，或至少32位



机器字长8位



运算结果



最多只能同时进行8位运算

各个数值位的“位权”

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

8bit寄存器

真值: 0 → 二进制: 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

8bit寄存器

真值: 99 → 二进制: 1100011

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

真值: 255 → 二进制: 11111111

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

8bit寄存器

真值: 256 → 二进制: 100000000

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

8bit寄存器

溢出 1 0 0 0 0 0 0 0 0

强行硬塞: 只能保存低8位



无符号整数: ① 全部二进制位都是数值位，没有符号位，第 i 位的位权是 2^{i-1}

② n bit 无符号整数表示范围 $0 \sim 2^n - 1$ ，超出则溢出，意味着该计算机无法一次处理这么多

③ 可以表示的最小的数 全0，可以表示的最大的数 全1.

无符号整数的加法运算

A : 99 → 二进制: 1100011

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

B : 9 → 二进制: 1001

0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

8bit寄存器

A+B=108 → 二进制: 1101100

0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

8bit寄存器

计算机硬件如何做无符号整数的加法: 从最低位开始, 按位相加, 并往更高位进位

无符号整数的减法运算

bilibili

无符号整数的减法运算

A: 99 → 二进制: 1100011

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

B: 9 → 二进制: 1001

0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

Tips: 加法电路
造价便宜, 减法
电路造价昂贵。
若可将减法转变为加法, 省钱!

A-B=90 → 二进制: 1011010

0	1	0	1	1	0	1	0
---	---	---	---	---	---	---	---

8bit寄存器

计算机硬件如何做无符号整数的减法:

- ① “被减数”不变, ~~(减数)全部位按位取反、未位+1, 减法变加法~~
- ② 从最低位开始, 按位相加, 并往更高位进位



A : 99 —> 二进制: 1100011

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

B : 9 —> 二进制: 1001

0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

8bit寄存器

A : 99 —> 二进制: 1100011

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

1	1	1	1	0	1	1	1
---	---	---	---	---	---	---	---

8bit寄存器

“减数”B的变形

减法变加法

A-B=90 —> 二进制: 1011010

1	0	1	0	1	1	0	1	0
---	---	---	---	---	---	---	---	---

8bit寄存器

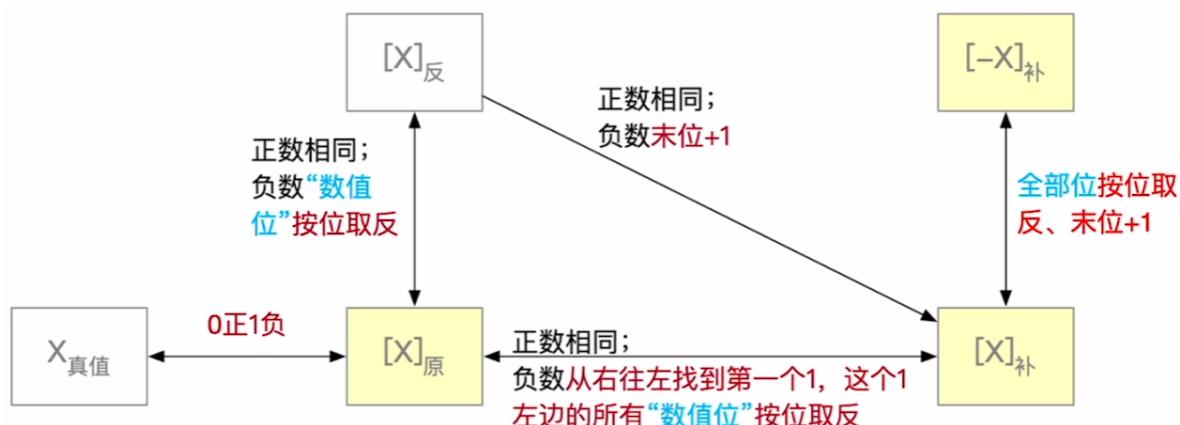
计算机硬件如何做无符号整数的减法:

- ① “被减数”不变, “减数”全部位按位取反、末位+1, 减法变加法
- ② 从最低位开始, 按位相加, 并往更高位进位

带符号整数

计算机内部, 所有带符号整数的加、减法都要先转化为补码

快速转换技巧: 从右往左找到第一个1, 这个1左边的全部位按位取反



计算机硬件如何做带符号数补码的加法: 从最低位开始, 按位相加 (符号位参与运算), 并往更高位进位

计算机硬件如何做带符号数补码的减法:

- ① “被减数”不变, “减数”全部位按位取反、末位+1, 减法变加法
- ② 从最低位开始, 按位相加, 并往更高位进位

带符号整数在计算机中的应用

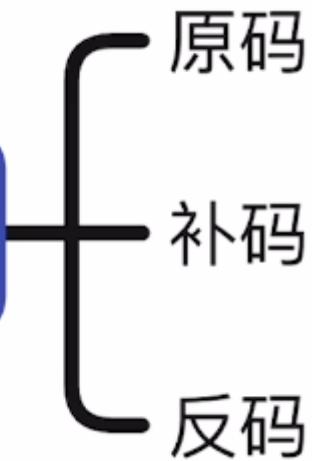
带符号整数, 即“整数”, -2, -1, 0, 1, 2, 3, 4...

C语言中的带符号整数:

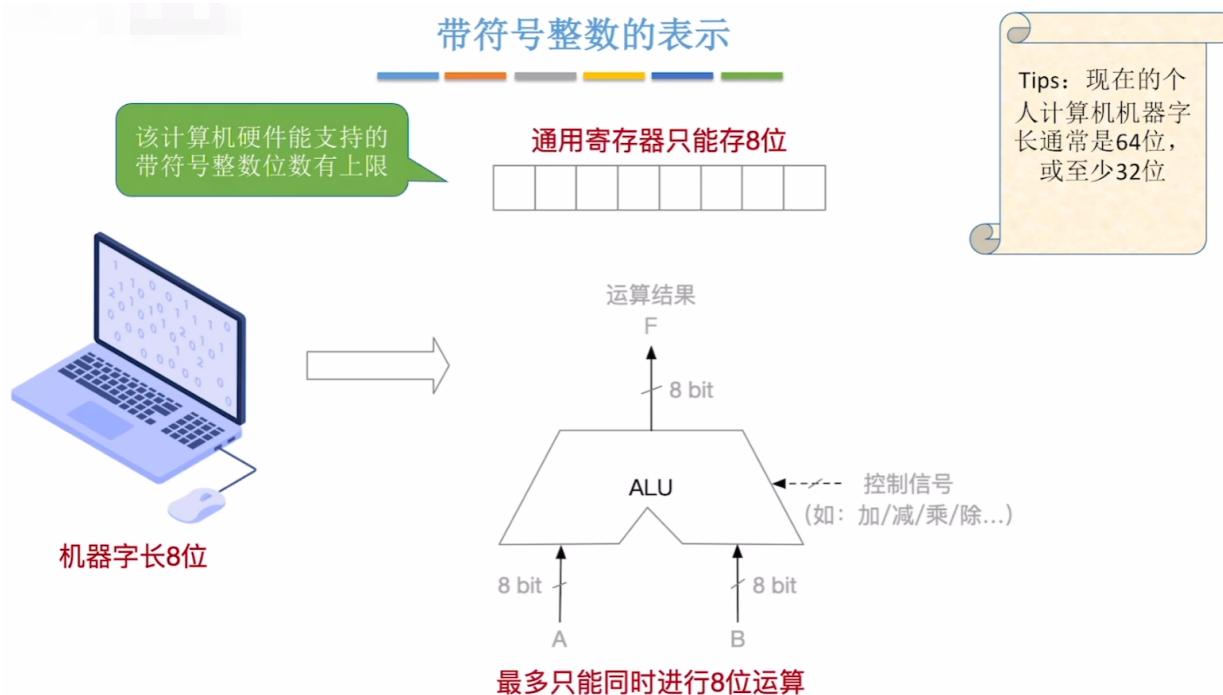
```
short a=1;//带符号整数（短整型，2B）  
int b=-2;//带符号整数（整型，4B）  
//位数不同，可表示数据范围不同
```

- 带符号整数，在计算机硬件内，如何表示？
- 带符号整数的加法、减法运算是怎么用硬件实现的？

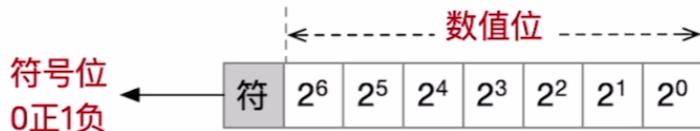
带符号整数的表示



同一个含义，用不同的编码方式表示



原码表示



真值: +19 → 二进制: +10011

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

真值: -19 → 二进制: -10011

1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

原码:

1. 符号位0/1对应正/负，剩余的数值位表示真值的绝对值

2. 若机器字长n+1位，带符号整数的原码表示范围：

$$-(2^n - 1) \leq x \leq 2^n - 1$$

3. 真值0有两种形式：+0和-0

$$[+0]_{\text{原}} = 0,0000000$$

$$[-0]_{\text{原}} = 1,0000000$$

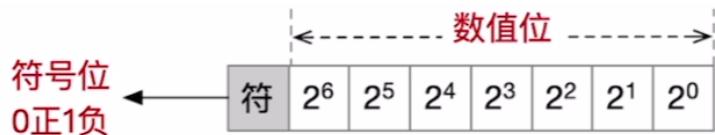
常见书面写法: $x = -19$

$$[x]_{\text{原}} = 1,0010011$$

若未指明机器字长，也可写为：

$$[x]_{\text{原}} = 1,10011$$

原码缺点



A: +19 → 二进制: +10011

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

B: -19 → 二进制: -10011

1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

$A+B=0$



1	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

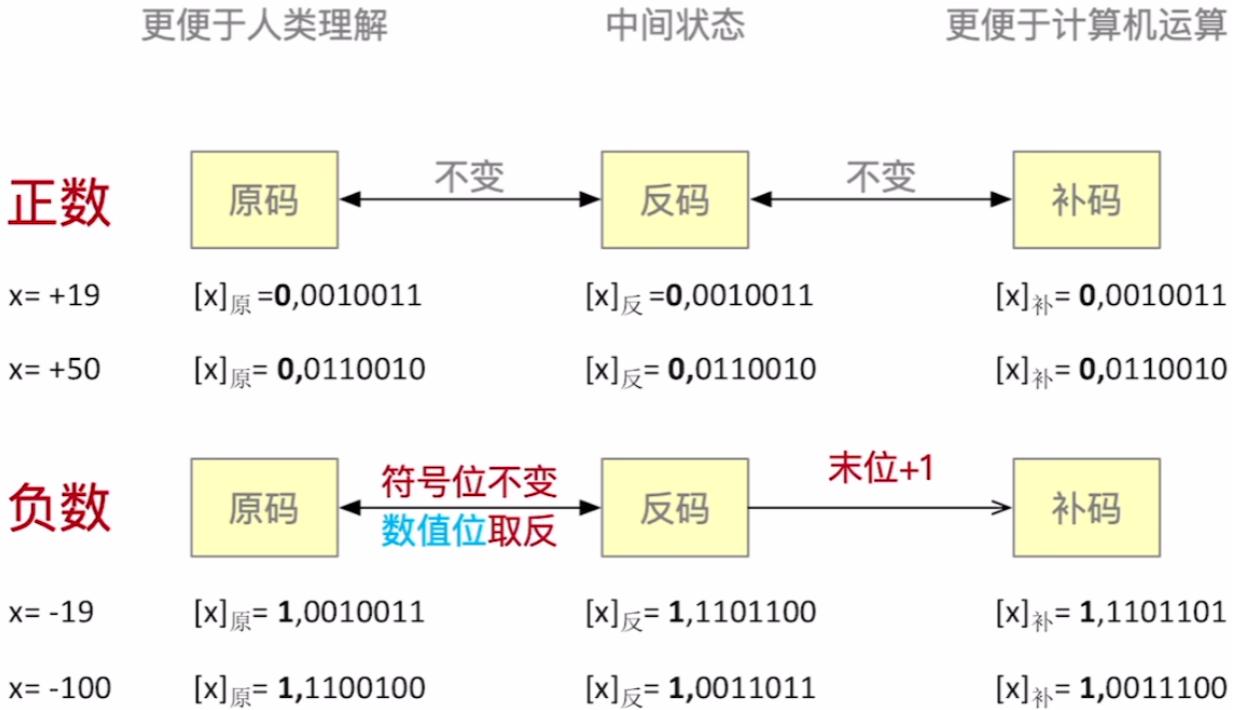
8bit寄存器

按位
相加

原码的缺点：符号位不能参与运算，需要设计复杂的硬件电路才能处理。

用补码表示真值——符号位可以参与运算

原码->反码->补码的转换 (机算)



原码、补码快速转换技巧 (手算)



补码的加法运算

计算机硬件如何做补码的加法：从最低位开始，按位相加（符号位参与运算），并往更高位进位



A: +19 → 补码

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

8bit寄存器

B: -19 → 补码

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

8bit寄存器

按位
相加

A+B=0 → 补码

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

8bit寄存器

A: -19 → 补码

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

8bit寄存器

B: -19 → 补码

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

8bit寄存器

按位
相加

A+B=-38 → 补码

1	1	0	1	1	0	1	0
---	---	---	---	---	---	---	---

8bit寄存器

补码的减法运算

加法电路造价便宜，减法电路造价昂贵。若可将减法转变为加法，省钱。

优点：用同一套电路即可处理所有的加减法，省钱。

$$\begin{aligned}
 [A]_{\text{补}} - [B]_{\text{补}} &=? \\
 A - B &= A + (-B) \\
 [A]_{\text{补}} - [B]_{\text{补}} &= [A]_{\text{补}} + [-B]_{\text{补}}
 \end{aligned}$$

要解决的问题：已知“减数”的补码，如何求其负值的补码表示？



$x = 19 \quad [x]_{\text{补}} = 0,0010011 \xrightarrow{\text{全部位按位取反}} 1,1101100 \xrightarrow{\text{末位+1}} 1,1101101$

$-x = -19 \quad [-x]_{\text{补}} = 1,1101101 \xrightarrow{\text{全部位按位取反}} 0,0010010 \xrightarrow{\text{末位+1}} 0,0010011$



补码的减法运算（例3）

优点：用同一套电路即可处理所有的加减法，省钱！

A: +19 → 补码

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

 8bit寄存器

B: -19 → 补码

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

A: +19 → 补码

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

-B: +19 → 补码

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

“减数”全部位按位取反、末位+1

8bit寄存器

[$-B$] 的补码

按位相加

$$[A]_{\text{补}} - [B]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}}$$

A-B=38 → 补码

0	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

 8bit寄存器

计算机硬件如何做带符号数补码的减法

- ① “被减数”不变，“减数”全部位按位取反、末位+1，减法变加法
- ② 从最低位开始，按位相加，并往更高位进位

各种码的基本特性总结

小题考点：几种码的特性对比

★ n+1 bit 的合法表示范围

最大的数怎么表示、最小的数怎么表示

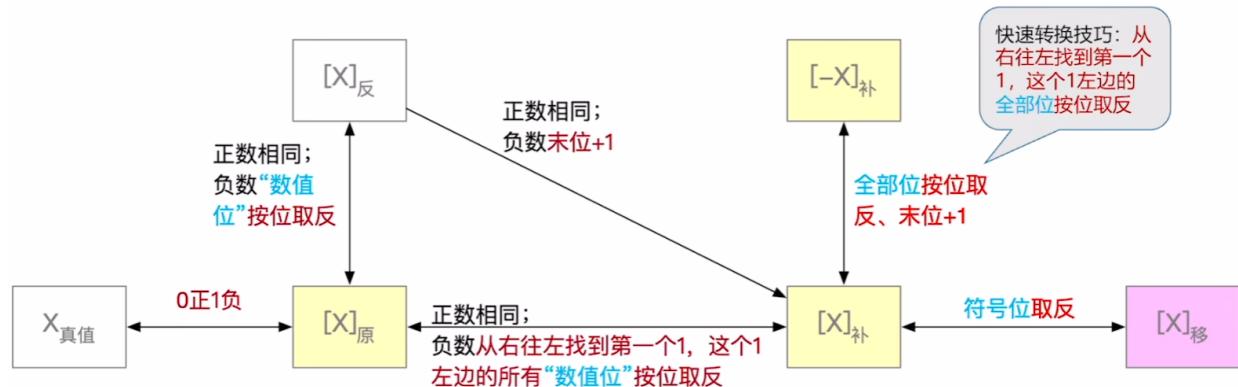
真值0的表示

$n+1$ bit	合法表示范围	最大的数	最小的数	真值0的表示
带符号整数:原码	$-(2^n-1) \leq x \leq 2^n-1$	$0,111\dots111 = 2^n-1$	$1,111\dots111 = -(2^n-1)$	$[+0]_{\text{原}}=0,000\dots000$ $[-0]_{\text{原}}=1,000\dots000$
带符号整数:反码	$-(2^n-1) \leq x \leq 2^n-1$	$0,111\dots111 = 2^n-1$	$1,000\dots000 = -(2^n-1)$	$[+0]_{\text{反}}=0,000\dots000$ $[-0]_{\text{反}}=1,111\dots111$
带符号整数:补码	$-2^n \leq x \leq 2^n-1$	$0,111\dots111 = 2^n-1$	$1,000\dots000 = -2^n$	$[0]_{\text{补}}=0,000\dots000$ 真值0只有一种补码
无符号整数	$0 \leq x \leq 2^{n+1}-1$	$1111\dots111 = 2^{n+1}-1$	$0000\dots000 = 0$	$0000\dots000$

原码和反码的合法表示范围完全相同，都有两种方法表示真值0
补码的合法表示范围比原码多一个负数，只有一种方法表示真值0

常见考点：两个数A和B进行某种运算后，是否发生溢出？——手算做题可以带入十进制验证，是否超出合法范围

原、反、补、移码的转换



移码

移码：补码的基础上将符号位取反。注意：移码只能用于表示整数

$x = +19D$	$[x]_{\text{原}} = 0,0010011$	$[x]_{\text{反}} = 0,0010011$	$[x]_{\text{补}} = 0,0010011$	$[x]_{\text{移}} = 1,0010011$	$[+0]_{\text{原}} = 00000000$	$[+0]_{\text{反}} = 00000000$	$[+0]_{\text{补}} = 00000000$	$[+0]_{\text{移}} = 10000000$
$x = -19D$	$[x]_{\text{原}} = 1,0010011$	$[x]_{\text{反}} = 1,1101100$	$[x]_{\text{补}} = 1,1101101$	$[x]_{\text{移}} = 0,1101101$	$[-0]_{\text{原}} = 10000000$	$[-0]_{\text{反}} = 11111111$	$[0]_{\text{补}} = 00000000$	$[0]_{\text{移}} = 10000000$
带符号整数的表示								

注意！真值0只有一种表示形式

若机器字长 $n+1$ 位，移码整数的表示范围：
 $-2^n \leq x \leq 2^n-1$ (与补码相同)

移码表示的整数很方便用硬件电路对比大小

True Value (十进制)

BCD (补码)

TBCD (移码)

True Value 增大

-128	1000 0000	0000 0000
-127	1000 0001	0000 0001
-126	1000 0010	0000 0010
...
-3	1111 1101	0111 1101
-2	1111 1110	0111 1110
-1	1111 1111	0111 1111
0	0000 0000	1000 0000
1	0000 0001	1000 0001
2	0000 0010	1000 0010
3	0000 0011	1000 0011
...
124	0111 1100	1111 1100
125	0111 1101	1111 1101
126	0111 1110	1111 1110
127	0111 1111	1111 1111

n+1 bit	合法表示范围	最大的数	最小的数	真值0的表示
带符号整数:原码	$-(2^n-1) \leq x \leq 2^n-1$	$0,111\dots111 = 2^n-1$	$1,111\dots111 = -(2^n-1)$	$[+0]_{原} = 0,000\dots000$ $[-0]_{原} = 1,000\dots000$
带符号整数:反码	$-(2^n-1) \leq x \leq 2^n-1$	$0,111\dots111 = 2^n-1$	$1,000\dots000 = -(2^n-1)$	$[+0]_{反} = 0,000\dots000$ $[-0]_{反} = 1,111\dots111$
带符号整数:补码	$-2^n \leq x \leq 2^n-1$	$0,111\dots111 = 2^n-1$	$1,000\dots000 = -2^n$	$[0]_{补} = 0,000\dots000$ 真值0只有一种补码
带符号整数:移码	$-2^n \leq x \leq 2^n-1$	$1111\dots111 = 2^n-1$	$0000\dots000 = -2^n$	$[0]_{移} = 1000\dots000$ 真值0只有一种移码
无符号整数	$0 \leq x \leq 2^{n+1}-1$	$1111\dots111 = 2^{n+1}-1$	$0000\dots000 = 0$	0000...000

原码和反码的合法表示范围完全相同，都有两种方法表示真值0

补码的合法表示范围比原码多一个负数，只有一种方法表示真值0

移码的合法表示范围比原码多一个负数，只有一种方法表示真值0

用几种码表示整数

原码和反码的真值0有两种表示

补码和移码的真值0只有一种表示

补码和移码可以多表示一个负数

行数	机器数	真值(十进制)				
		无符号数	原码	反码	补码	移码
1	0000 0000	0	+0	+0	+0,-0	-128
2	0000 0001	1	+1	+1	+1	-127
3	0000 0010	2	+2	+2	+2	-126
...
126	0111 1101	125	+125	+125	+125	-3
127	0111 1110	126	+126	+126	+126	-2
128	0111 1111	127	+127	+127	+127	-1
129	1000 0000	128	-0	-127	-128	0
130	1000 0001	129	-1	-126	-127	1
131	1000 0010	130	-2	-125	-126	2
...
253	1111 1100	252	-124	-3	-4	124
254	1111 1101	253	-125	-2	-3	125
255	1111 1110	254	-126	-1	-2	126
256	1111 1111	255	-127	-0	-1	127