

结构概览

```
0000 0000 0000 0000 0000 0000 0000 0011
||
||--|-----|
|  |
|  +--- 31          1*2^0+1*2^1=3
|
+----- 0 1
```

相关命令

```
lldb
(lldb) command script import /path/to/lldbsh.py
lldbsh.py download url
https://raw.githubusercontent.com/ihnorton/lldb.sh/master/lldbsh.py
```

```
=
(lldb) sh p/t 1|sed 's/^.*= 0b// ; s/.\{4\}/& /g'
0000 0000 0000 0000 0000 0000 0000 0001
```

```
=
(lldb) sh p/t -1-1|gsed 's/^.*= 0b// ; /~1/ s/1/N/ ; /~0/ s/0/P/ ; s/1/2/g ; s/0/1/g ; s/2/0/g'
1000 0000 0000 0000 0000 0000 0000 0001
```

```
=
(lldb) sh p/t -1-1|sed 's/^.*= 0b// ; s/.\{4\}/& /g'
1111 1111 1111 1111 1111 1111 1111 1110
```

```
= + 1
(lldb) sh p/t -1|sed 's/^.*= 0b// ; s/.\{4\}/& /g'
1111 1111 1111 1111 1111 1111 1111 1111
```

原码反码补码

数字	二进制
+1 原码	0000 0000 0000 0000 0000 0000 0000 0001
+1 反码	0000 0000 0000 0000 0000 0000 0000 0001
+1 补码	0000 0000 0000 0000 0000 0000 0000 0001

数字	二进制
-1 原码	1000 0000 0000 0000 0000 0000 0000 0001
-1 反码	1111 1111 1111 1111 1111 1111 1111 1110

数字	二进制
-1 补码	1111 1111 1111 1111 1111 1111 1111 1111

-1 + 1 错误结果

数字	二进制
+1 原码	0000 0000 0000 0000 0000 0000 0000 0001
-1 原码	1000 0000 0000 0000 0000 0000 0000 0001
计算结果	1000 0000 0000 0000 0000 0000 0000 0010

-1 + 1 正确结果

数字	二进制
-1 补码	1111 1111 1111 1111 1111 1111 1111 1111
+1 补码	0000 0000 0000 0000 0000 0000 0000 0001
溢出结果	1 0000 0000 0000 0000 0000 0000 0000 0000
正确结果	0000 0000 0000 0000 0000 0000 0000 0000

负数补码转回原码 负数补码转回原码：符号位不变，-1，数据位取反 | 数字 | 二进制 | -1 补码 | 1111 1111 1111 1111 1111 1111 1111 1111 | -1 取反 | 1111 1111 1111 1111 1111 1111 1111 1110 | -1 原码 | 1000 0000 0000 0000 0000 0000 0000 0000 0001 |

有符号负数左移运算

数字	二进制
-2 原码	1000 0000 0000 0000 0000 0000 0000 0010
-2 反码	1111 1111 1111 1111 1111 1111 1111 1101
-2 补码	1111 1111 1111 1111 1111 1111 1111 1110
-2 >> 1 补码	0111 1111 1111 1111 1111 1111 1111 1111
-2 >> 1 反码	1000 0000 0000 0000 0000 0000 0000 0000
-2 >> 1 原码	1000 0000 0000 0000 0000 0000 0000 0001

有符号负数右移运算

数字	二进制
-2 原码	1000 0000 0000 0000 0000 0000 0000 0010
-2 反码	1111 1111 1111 1111 1111 1111 1111 1101
-2 补码	1111 1111 1111 1111 1111 1111 1111 1110
-2 < 1 补码	1111 1111 1111 1111 1111 1111 1111 1100
-2 < 1 反码	1111 1111 1111 1111 1111 1111 1111 1011
-2 < 1 原码	1000 0000 0000 0000 0000 0000 0000 0100

无符号负数位移运算

数字	二进制
-2 原码	1000 0000 0000 0000 0000 0000 0000 0010
-2 反码	1111 1111 1111 1111 1111 1111 1111 1101
-2 补码	1111 1111 1111 1111 1111 1111 1111 1110
-2 >> 1 补码	0111 1111 1111 1111 1111 1111 1111 1111
-2 >> 1 反码	0111 1111 1111 1111 1111 1111 1111 1111
-2 >> 1 原码	0111 1111 1111 1111 1111 1111 1111 1111
-2 >> 1 结果	2147483647

利用补码进行正减法 $12345 - 1 == 12345 + -(1)$ | 数字 | 二进制 || — | — |
 |12345 补码|0000 0000 0000 0000 0011 0000 0011 1001| |-1 补码|1111 1111
 1111 1111 1111 1111 1111 1111| |溢出结果|0000 0000 0000 0000 0011 0000 0011
 1000| |正确结果|0000 0000 0000 0000 0011 0000 0011 1000|

利用补码进行负减法 $1 - 12345 == 1 + (-12345)$
 | 数字 | 二进制 || — | — | |1 补码|0000 0000 0000 0000 0000 0000 0000
 0001| |12345 补码|1111 1111 1111 1111 1100 1111 1100 0111| |计算结果|1111 1111
 1111 1111 1100 1111 1100 1000|

整数在程序中的存储方式：补码

```
vim int.c
#include <stdio.h>

int main(){
    int signed_int = -12345;           // 0xffffcfc7
    unsigned int unsigned_int = 12345; // 0x3039
    printf("%d %u\n", signed_int, unsigned_int);
    return 0;
}
```

[ESC] :wq

```
gcc int.c -o int
```

```
objdump -d -j .text -M intel int | gsed '1,7d ; s/^.\{3\} // ; s/:\.\{23\}/ / ; /\.\{14\}/ s/
100003f50 push rbp
100003f51 mov rbp, rsp
100003f54 sub rsp, 0x10
100003f58 mov DWORD PTR [rbp-0x4], 0x0
100003f5f mov DWORD PTR [rbp-0x8], 0xffffffffcfc7
100003f66 mov DWORD PTR [rbp-0xc], 0x3039
100003f6d mov esi, DWORD PTR [rbp-0x8]
100003f70 mov edx, DWORD PTR [rbp-0xc]
100003f73 lea rdi, [rip+0x34] # 100003fae <_main+0x5e>
100003f7a mov al, 0x0
100003f7c call 100003f8e <_main+0x3e>
100003f81 xor ecx, ecx
100003f83 mov DWORD PTR [rbp-0x10], eax
100003f86 mov eax, ecx
100003f88 add rsp, 0x10
100003f8c pop rbp
100003f8d ret
```

```
(lldb) p/x -12345
```

```
(int) $15 = 0xffffffffcfc7
```

```
(lldb) p/x 12345
```

```
(int) $17 = 0x00003039
```

口算技巧

$2 \ll 1 = 4 = 2 * 2$

$2 \ll 2 = 8 = 2 * 2 * 2$

$2 \ll n = 2 * (2^n)$

$m \ll n = m * (2^n)$

参考资料 程序中的整数

一文搞明白位运算、补码、反码、原码

How do I convert an integer to binary in JavaScript?

历史上有哪些因为编程开发中的错误或者漏洞造成惨痛损失的案例?