a. 除数为无符号非2的幂 (上)

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / 3 = %u", (unsigned)argc / 3);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, 0AAAAAAABh ;eax=M
00401015 mul dword ptr [esp+4] ;无符号，edx.eax=argc*M
00401019 shr edx, 1 ;无符号右移，edx=argc*M >>32>>1
0040101B push edx ;参数 2
0040101C push offset aArgc3U ;参数 1 "argc / 3 = %u"
00401021 call sub_401030 ;调用printf函数
00401026 add esp, 8 ;平衡栈
00401029 xor eax, eax
0040102B retn
```

解方程得：$c = \frac{2^n}{M} = \frac{2^{33}}{\text{AAAAAAABh}} = 2.999999\cdots\cdots \approx 3$（注：此处的“约等于”在后面讨论除法优化原则处详细解释）。

于是，我们反推出优化前的高级代码如下：

```
argc/3
```

b. 除数为无符号非2的幂 (下)

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / 7 = %u", (unsigned)argc / 7);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov ecx, [esp+4] ;ecx=argc
00401014 mov eax, 24924925h ;eax=M
00401019 mul ecx ;无符号乘法，edx.eax=argc*M
0040101B sub ecx, edx ;ecx=argc-(argc*M>>32)
0040101D shr ecx, 1 ;无符号右移，ecx=(argc-(argc*M>>32))>>1
0040101F add ecx, edx ;ecx=((argc-(argc*M>>32))>>1)+(argc*M>>32)
00401021 shr ecx, 2 ;ecx=(((argc-(argc*M>>32))>>1)+(argc*M>>32))>>2
00401024 push ecx ;参数2
00401025 push offset aArgc7U ;参数1 "argc / 7 = %u"
0040102A call sub_401040 ;调用printf函
0040102F add esp, 8 ;平衡栈
00401032 xor eax, eax
```

解方程求c：

$$c = \frac{2^{32+n}}{2^{32}+M} = \frac{2^{35}}{2^{32}+24924925\text{h}} = 6.99999\cdots\cdots \approx 7$$

于是，我们反推出优化前的高级代码为：

```
argc/7
```

总结

当遇到数学优化公式$\frac{x}{c} = \{[(x-(x\cdot M >> 32) >> n1] + (x\cdot M >> 32)\} >> n2$时，基本可判定其是除法优化后的代码，除法原型为$x$除以常量$c$，mul可表明是无符号计算，操作数是优化前的被除数x，接下来统计右移的总次数以确定公式中的n值，然后使用公式$c = \frac{2^{32+n}}{2^{32}+M}$将魔数作为$M$值代入公式求解常量除数$c$，即可恢复除法原型。

c. 除数为有符号2的幂

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / 8 = %d", argc / 8);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, [esp+4] ;eax = argc
00401014 cdq ;eax符号位扩展，正数edx=0，负数edx=0xffffffff
00401015 and edx, 7 ;负数edx=7，正数edx=0
00401018 add eax, edx ;if(argc<0)，eax=argc+7
 ;if(argc>=0)，eax=argc+0
0040101A sar eax, 3 ;if(argc<0)，eax=(argc+7)>>3
 ;if(argc >= 0)，eax=argc>>3
0040101D push eax ;参数 2
0040101E push offset aArgc8D ;参数 1 "argc / 8 = %d"
00401023 call sub_401030 ;调用printf函数
00401028 add esp, 8 ;平衡栈
0040102B xor eax, eax
0040102D retn
```

对于$\left[\frac{x}{2^n}\right]$，当$x \geqslant 0$时，有：

$$\left[\frac{x}{2^n}\right] = \left\lfloor\frac{x}{2^n}\right\rfloor \Leftrightarrow x >> n$$

当$x < 0$时，有：

$$\left[\frac{x}{2^n}\right] = \left\lceil\frac{x}{2^n}\right\rceil$$

根据推导7可得：

$$\left[\frac{x}{2^n}\right] = \left\lceil\frac{x}{2^n}\right\rceil = \left\lfloor\frac{x+2^n-1}{2^n}\right\rfloor \Leftrightarrow [x+(2n-1)] >> n$$

例如：$\left\lceil\frac{x}{8}\right\rceil \Leftrightarrow \left\lfloor\frac{x+2^3-1}{2^3}\right\rfloor \Leftrightarrow (x+7) >> 3$。

总结

当遇到数学优化公式：如果$x \geqslant 0$，则$\frac{x}{2^n} = x >> n$，如果$x < 0$，则时$\frac{x}{2^n} = [x+(2^n-1)] >> n$，基本可判定是除法优化后的代码，根据$n$（右移次数）即可恢复除法原型。

d. 除数为有符号非2的幂

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / 9 = %d", argc / 9);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, 38E38E39h ;eax = M
00401015 imul dword ptr [esp+4] ;edx.eax=argc*M
00401019 sar edx, 1 ;edx=(argc*M>>32)>>1
0040101B mov eax, edx
0040101D shr eax, 1Fh ;eax=eax>>31取符号位
00401020 add eax, edx ;if(edx < 0)，eax=((argc*M>>32)>>1)+1
 ;if(edx >=0)，eax=(argc*M>>32)>>1
00401022 push eax ;参数 2
00401023 push offset aArgc9D ;参数 1 "argc / 9 = %d"
00401028 call sub_401040 ;调用printf函数
0040102D add esp, 8 ;平衡栈
00401030 xor eax, eax
00401032 retn
```

解方程得：

$$c = \frac{2^n}{M} = \frac{2^{33}}{38E38E39h} = 8.999999\cdots\cdots \approx 9$$

于是，我们反推出优化前的高级代码为：

```
argc/9
```

总结

当遇到数学优化公式：如果$x \geqslant 0$，则$\frac{x}{c} = x \cdot M >> 32 >> n$；如果$x < 0$，则$\frac{x}{c} = (x \cdot M >> 32 >> n) + 1$时，基本可判定是除法优化后的代码，其除法原型为$x$除以常量$c$，imul可表明是有符号计算，其操作数是优化前的被除数x，接下来统计右移的总次数以确定公式中的n值，然后使用公式$c = \frac{2^n}{M}$，将魔数作为M值代入公式求解常量除数c的近似值，四舍五入取整后，即可恢复除法原型。

e. 除数为有符号非2的幂

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / 7 = %d", argc / 7);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, 92492493h ;eax = M
00401015 imul dword ptr [esp+4] ;edx.eax=argc*M
00401019 add edx, [esp+4] ;edx=(argc*M>>32)+argc
0040101D sar edx, 2 ;edx=((argc*M>>32)+argc)>>2
00401020 mov eax, edx ;eax=edx
00401022 shr eax, 1Fh ;eax=eax>>31取符号位
00401025 add eax, edx ;if(edx<0), eax=((argc*M>>32)+argc)>>2+1
 ;if(edx >=0), eax=((argc*M>>32)+argc)>>2
00401027 push eax ;参数  1
00401028 push offset aArgc7D ;参数  2 "argc / 7 = %d"
0040102D call sub_401040 ;调用printf函数
00401032 add esp, 8 ;平衡栈
00401035 xor eax, eax
00401037 retn
```

解方程得：

$$c = \frac{2^n}{M} = \frac{2^{34}}{92492493h} = 6.999999\cdots\cdots \approx 7$$

于是，我们反推出优化前的高级代码为：

```
argc / 7
```

注意，这里的arg c是有符号整型，因为指令中使用的是imul有符号乘法指令。

总结

当遇到数学优化公式：如果$x \geqslant 0$，则$\frac{x}{c} = (x \cdot M >> 32) + x >> n$；如果$x < 0$，则$\frac{x}{c} = [(x \cdot M >> 32) + x >> n] + 1$时，基本可判定是除法优化后的代码，其除法原型为$x$除以常量$c$，imul表明是有符号计算，其操作数是优化前的被除数x，接下来统计右移的总次数以确定公式中的n值，然后使用公式$c = \frac{2^n}{M}$，将魔数作为M值代入公式求解常量除数c，即可恢复除法原型。

f. 除数为有符号非2的幂

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / -4 = %d", argc / -4);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, [esp+4] ;eax = argc
00401014 cdq ;eax符号位扩展, 正 edx=0, 负 edx=0xffffffff
00401015 and edx, 3 ;负 edx=3, 正 edx=0
00401018 add eax, edx ;if(argc < 0), eax=argc+3
 ;if(argc >= 0), eax=argc+0
0040101A sar eax, 2 ;if(argc < 0), eax=(argc+3)>>2
 ;if(argc >= 0), eax=argc>>2
0040101D neg eax ;eax = -eax
0040101F push eax ;参数  2
00401020 push offset aArgc4D ;参数  1 "argc / -4 = %d"
00401025 call sub_401030 ;调用printf函数
0040102A add esp, 8 ;平衡栈
0040102D xor eax, eax
0040102F retn
```

总结

当遇到数学优化公式：如果$x \geqslant 0$，则$\frac{x}{-2^n} = -(x >> n)$；如果$x < 0$，则$\frac{x}{-2^n} = -\{[x + (2^n - 1)] >> n\}$时，基本可判定是除法优化后的代码，根据$n$的值，可恢复除法原型。

g. 除数为有符号负非2的幂

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / -5 = %d", argc / -5);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, 99999999h ;eax = M
00401015 imul dword ptr [esp+4] ;edx.eax=argc*M
00401019 sar edx, 1 ;edx=argc*M>>32>>1
0040101B mov eax, edx
0040101D shr eax, 1Fh ;eax=edx>>31取符号位
00401020 add eax, edx ;if(edx < 0), eax=(argc*M>>32>>1)+1
 ;if(edx >=0), eax=argc*M>>32>>1
00401022 push eax ;╮ 2
00401023 push offset aArgc5D ;╮ 1 "argc / -5 = %d"
00401028 call sub_401040 ;调○printf函
0040102D add esp, 8 ;平掺栈
00401030 xor eax, eax
00401032 retn
```

$$|c|=\frac{2^n}{2^{32}-M}=\frac{2^{33}}{2^{32}-99999999\text{h}}=\frac{2^{33}}{66666667\text{h}}=4.999999\cdots\cdots\approx 5$$

分析以上代码时，很容易与除数为正的情况混淆，我们先看这两者之间的重要差异。关键在于上述代码中魔数的值。在前面讨论的除以7的例子中，当魔数最高位为1时，对于正除数，编译器会在imul和sar之间产生调整作用的add指令，而本例没有，故结合上下流程可分析魔数为补码形式，除数为负。这点应作为区分负除数的重要依据。

于是，我们反推出优化前的高级代码如下。

```
argc / -5
```

总结

当遇到数学优化公式：如果$x\geqslant 0$，则$\frac{x}{c}=x\cdot M>>32>>n$；如果$x<0$，则$\frac{x}{c}=(x\cdot M>>32>>n)+1$时，则基本可判定是除法优化后的代码，其除法原型为$x$除以常量$c$，imul可表明是有符号计算，其操作数是优化前的被除数x。由于魔数取值大于7fffffffh，而imul和sar之间未见任何调整代码，故可认定除数为负，且魔数为补码形式，接下来统计右移的总次数，以确定公式中的n值，然后使用公式$|c|=\frac{2^n}{2^{32}-M}$，将魔数作为M值代入公式求解常量除数$|c|$，即可恢复除法原型。

g. 除数为有符号负非2的幂（下）

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / -7 = %d", argc / -7);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, 6DB6DB6Dh ;eax=M
00401015 imul dword ptr [esp+4];edx.eax=argc*M
00401019 sub edx, [esp+4] ;edx=(argc*M>>32)-argc
0040101D sar edx, 2 ;edx=(argc*M>>32)-argc>>2
00401020 mov eax, edx
00401022 shr eax, 1Fh ;eax=edx>>31取符号位
00401025 add eax, edx ;if(edx < 0), eax=((argc*M>>32)-argc>>2) + 1
 ;if(edx >=0), eax=(argc*M>>32)-argc>>2
00401027 push eax ;╮ 2
00401028 push offset aArgc7D ;╮ 1 "argc / -7 = %d"
0040102D call sub_401040 ;调○printf函
00401032 add esp, 8 ;平掺栈
00401035 xor eax, eax
00401037 retn
```

接下来，我们求解$|c|$：

$$|c|=\frac{2^n}{2^{32}-M}=\frac{2^{34}}{2^{32}-6\text{DB}6\text{DB}6\text{Dh}}=\frac{2^{34}}{92492493\text{h}}=6.999999\cdots\cdots\approx 7$$

于是，我们反推出优化前的高级代码为：

```
argc / -7
```

总结

当遇到数学优化公式：如果$x\geqslant 0$，则$\frac{x}{c}=(x\cdot M>>32)-x>>n$，如果$x<0$，则$\frac{x}{c}=[(x\cdot M>>32)-x>>n]+1$，可判定是除法优化后的汇编代码，其除法原型为$x$除以常量$c$，imul可表明是有符号计算，其操作数是优化前的被除数x。由于魔数取值小于等于7fffffffh，而imul和sar之间有sub指令调整乘积，故可认定除数为负，且魔数为补码形式，接下来统计右移的总次数以确定公式中的n值，然后使用公式$|c|=\frac{2^n}{2^{32}-M}$将魔数作为M值代入公式求解常量除数$|c|$，即可恢复除法原型。

g. 除数为有符号负非2的幂

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / -5 = %d", argc / -5);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, 99999999h ;eax = M
00401015 imul dword ptr [esp+4] ;edx.eax=argc*M
00401019 sar edx, 1 ;edx=argc*M>>32>>1
0040101B mov eax, edx
0040101D shr eax, 1Fh ;eax=edx>>31取符号位
00401020 add eax, edx ;if(edx < 0), eax=(argc*M>>32>>1)+1
 ;if(edx >=0), eax=argc*M>>32>>1
00401022 push eax ;ヽ 2
00401023 push offset aArgc5D ;ヽ 1 "argc / -5 = %d"
00401028 call sub_401040 ;调○printf函
0040102D add esp, 8 ;平襂栈
00401030 xor eax, eax
00401032 retn
```

$$|c| = \frac{2^n}{2^{32} - M} = \frac{2^{33}}{2^{32} - 99999999\text{h}} = \frac{2^{33}}{66666667\text{h}} = 4.999999\cdots\cdots \approx 5$$

分析以上代码时，很容易与除数为正的情况混淆，我们先看这两者之间的重要差异。关键在于上述代码中魔数的值。在前面讨论的除以7的例子中，当魔数最高位为1时，对于正除数，编译器会在imul和sar之间产生调整作用的add指令，而本例没有，故结合上下流程可分析魔数为补码形式，除数为负。这点应作为区分负除数的重要依据。

于是，我们反推出优化前的高级代码如下。

```
argc / -5
```

总结

当遇到数学优化公式：如果$x \geqslant 0$，则$\frac{x}{c} = x \cdot M >> 32 >> n$；如果$x < 0$，则$\frac{x}{c} = (x \cdot M >> 32 >> n) + 1$时，则基本可判定是除法优化后的代码，其除法原型为$x$除以常量$c$，imul可表明是有符号计算，其操作数是优化前的被除数x。由于魔数取值大于7fffffffh，而imul和sar之间未见任何调整代码，故可认定除数为负，且魔数为补码形式，接下来统计右移的总次数，以确定公式中的n值，然后使用公式$|c| = \frac{2^n}{2^{32} - M}$，将魔数作为$M$值代入公式求解常量除数$|c|$，即可恢复除法原型。

f. 除数为有符号负非2的幂（下）

```
// C++源·
#include <stdio.h>
int main(int argc, char* argv[]) {
 printf("argc / -7 = %d", argc / -7);
 return 0;
}
```

```
//x86_vs对应汇编代·讲解
00401010 mov eax, 6DB6DB6Dh ;eax=M
00401015 imul dword ptr [esp+4];edx.eax=argc*M
00401019 sub edx, [esp+4] ;edx=(argc*M>>32)-argc
0040101D sar edx, 2 ;edx=(argc*M>>32)-argc>>2
00401020 mov eax, edx
00401022 shr eax, 1Fh ;eax=edx>>31取符号位
00401025 add eax, edx ;if(edx < 0), eax=((argc*M>>32)-argc>>2) + 1
 ;if(edx >=0), eax=(argc*M>>32)-argc>>2
00401027 push eax ;ヽ 2
00401028 push offset aArgc7D ;ヽ 1 "argc / -7 = %d"
0040102D call sub_401040 ;调○printf函
00401032 add esp, 8 ;平襂栈
00401035 xor eax, eax
00401037 retn
```

接下来，我们求解$|c|$：

$$|c| = \frac{2^n}{2^{32} - M} = \frac{2^{34}}{2^{32} - 6\text{DB6DB6Dh}} = \frac{2^{34}}{92492493\text{h}} = 6.999999\cdots\cdots \approx 7$$

于是，我们反推出优化前的高级代码为：

```
argc / -7
```

总结

当遇到数学优化公式：如果$x \geqslant 0$，则$\frac{x}{c} = (x \cdot M >> 32) - x >> n$，如果$x < 0$，则$\frac{x}{c} = [(x \cdot M >> 32) - x >> n] + 1$，可判定是除法优化后的汇编代码，其除法原型为$x$除以常量$c$，imul可表明是有符号计算，其操作数是优化前的被除数x。由于魔数取值小于等于7fffffffh，而imul和sar之间有sub指令调整乘积，故可认定除数为负，且魔数为补码形式，接下来统计右移的总次数以确定公式中的n值，然后使用公式$|c| = \frac{2^n}{2^{32} - M}$将魔数作为$M$值代入公式求解常量除数$|c|$，即可恢复除法原型。