C/C++32位整数除法优化学习笔记 (/2017/04/C-C++32%E4%BD%8D%E6%95%B4%E6%95%B0%E9%99%A4%E6%B3%95%E4%BC%98%E5

01 Apr 2017 逆向 (/tag/逆向)

C/C++32位整数除法优化学习笔记

引子

> 整数除法是一项消耗很大的运算，每个除法指令所要占用的时间周期是加法和乘法的好几倍，所以编译器在进行速度优化编译时会对整数的除法进行优化，尽量将其转化为加法减法除法或者移位运算。而优化后得到的代码与源代码有很大的差别，若不了解其中细节，在逆向时碰到这类代码就会眼花缭乱不知所云。
> (PS:本文中所有代码都是在64位linux下使用gcc -m32 -O2编译的，汇编代码是通过gdb反编译的intel风格代码)

除法优化

以下var表示不会被编译器解释为常量的整形变量，const表示常量与被编译器优化为常量的整型数

const / const

这一类除法的优化最为简单，直接在编译阶段就将除法的值算出即可。

var / var

由于两个变量都未知，所以编译器无法对其进行优化，其debug和release版本的汇编代码相同

const / var

除数为变量，此情况下编译器也不做优化

var / const

除数为常量，则可以对其进行优化，以下详细讨论这种情况的优化

仅除数为常量的整形除法优化

> 我们可以将这种情况的优化细分为以下两种
>
> - 除数为正的2的整数次幂
> - 除数为正的非2的整数次幂
> - 除数为负的2的整数次幂
> - 除数为负的非2的整数次幂

除数为正的2的整数次幂

代码分析

```
//例1
#include <stdio.h>
int main(int argc,char *argv[])
{
    printf("%f", argc / 8);
    return 0;
}
```

将例1代码用gcc O2模式编译，在用gdb反汇编得到如下代码

```
;例1 main函数反汇编代码
   0x08048310 <+0>:     lea    ecx,[esp+0x4]
   0x08048314 <+4>:     and    esp,0xfffffff0
   0x08048317 <+7>:     push   DWORD PTR [ecx-0x4]
   0x0804831a <+10>:    push   ebp
   0x0804831b <+11>:    mov    ebp,esp
   0x0804831d <+13>:    push   ecx
   0x0804831e <+14>:    sub    esp,0xc
   0x08048321 <+17>:    mov    edx,DWORD PTR [ecx]
   0x08048323 <+19>:    lea    eax,[edx+0x7]
   0x08048326 <+22>:    test   edx,edx
   0x08048328 <+24>:    cmovns eax,edx
   0x0804832b <+27>:    sar    eax,0x3
   0x0804832e <+30>:    push   eax
   0x0804832f <+31>:    push   0x80484c0
   0x08048334 <+36>:    call   0x80482e0 <printf@plt>
   0x08048339 <+41>:    mov    ecx,DWORD PTR [ebp-0x4]
   0x0804833c <+44>:    add    esp,0x10
   0x0804833f <+47>:    xor    eax,eax
   0x08048341 <+49>:    leave
   0x08048342 <+50>:    lea    esp,[ecx-0x4]
   0x08048345 <+53>:    ret
```

有几步是值得我们关注的,我将其提炼出来

```
;上述代码关键部分
lea    ecx,[esp+0x4]       ;ecx = &argc
;...
mov    edx,DWORD PTR [ecx] ;edx = *ecx
lea    eax,[edx+0x7]       ;eax = edx + 7
test   edx,edx             ;检查edx是否为负数
cmovns eax,edx             ;不为负数则 eax = edx
sar    eax,0x3             ;eax >> 3（算数右移）
;...
```

将以上代码写成算式： $\frac{argc}{8} = \begin{cases} argc >> 8 & \text{argc}>=0, \\ (argc+7) >> 8 & argc < 0, \end{cases}$

解释：

C/C++对整数除法采用**向0取整**的方式，非0整数做整数除以2^n时只需右移n位。而对于负数而言，负数除法向0取整值将增大，在数学中有以下结论(这里就直接拿来用了)： $\lceil \frac{x}{2^n} \rceil = \lfloor \frac{x+2^n-1}{2^n} \rfloor$

因此，在计算除数为2的整数次幂的整数除法时可以通过以下方式：

$$\frac{x}{2^n} = \begin{cases} x >> n & \text{x}>=0, \\ (x+2^n-1) >> n & x < 0, \end{cases}$$

结论：

对于类似以下结构的汇编代码基本可以判断为除数为2的整数次幂的除法

```
mov   reg0,var
lea   reg,[var + k] ;其中var是变量，$$k=2^n-1$$
test  reg0,reg0
cmovns reg,reg0
sar   reg,n
```

除数为正的非2的整数次幂

代码分析

```
//例2
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("%d", argc / 9);
    return 0;
}
```

```
   0x08048310 <+0>:     lea    ecx,[esp+0x4]
   0x08048314 <+4>:     and    esp,0xfffffff0
   0x08048317 <+7>:     mov    edx,0x38e38e39
   0x0804831c <+12>:    push   DWORD PTR [ecx-0x4]
   0x0804831f <+15>:    push   ebp
   0x08048320 <+16>:    mov    ebp,esp
   0x08048322 <+18>:    push   ecx
   0x08048323 <+19>:    sub    esp,0xc
   0x08048326 <+22>:    mov    ecx,DWORD PTR [ecx]
   0x08048328 <+24>:    mov    eax,ecx
   0x0804832a <+26>:    sar    ecx,0x1f
   0x0804832d <+29>:    imul   edx
   0x0804832f <+31>:    sar    edx,1
   0x08048331 <+33>:    sub    edx,ecx
   0x08048333 <+35>:    push   edx
   0x08048334 <+36>:    push   0x80484c0
   0x08048339 <+41>:    call   0x80482e0 <printf@plt>
   0x0804833e <+46>:    mov    ecx,DWORD PTR [ebp-0x4]
   0x08048341 <+49>:    add    esp,0x10
   0x08048344 <+52>:    xor    eax,eax
   0x08048346 <+54>:    leave
   0x08048347 <+55>:    lea    esp,[ecx-0x4]
   0x0804834a <+58>:    ret
```

提取关键代码如下：

```
lea    ecx,[esp+0x4]          ;ecx=&argc
;...
mov    edx,0x38e38e39         ;这是一个Magic number
;...
mov    ecx,DWORD PTR [ecx]    ;ecx=*ecx(exc = argc)
mov    eax,ecx                ;eax=ecx
sar    ecx,0x1f               ;ecx=ecx>>31
imul   edx                    ;edx=eax*edx >> 32
sar    edx,1                  ;edx=edx >> 1
sub    edx,ecx                ;edx-=ecx
```

把这段代码等价成表达式(用m表示0x38e38e39)： $\frac{x\cdot m}{2^{33}}$ x的符号位

我又写了一个计算argc/7的程序，直接写出计算式如下(m=0x92492493) $\frac{\frac{x\cdot m}{2^{32}}+x}{2^2}$ x的符号位

解释：

我们看到argc/7和argc/9产生的代码是有较大不同的。究其原因在于argc/7的MagicNumber大于0x80000000，相当与一个负数的补码，最终计算结果会是一个负数，而我们做正整数除法所得到的结果应该是一个正数，故我们对其进行特殊处理,我们进行以下近似变化

$\frac{\frac{x\cdot m}{2^{32}}+x}{2^n}-\frac{x}{2^{31}}=\frac{x\cdot m-2^n\cdot x}{2^{32+n}}=\frac{x}{o}$ 我们可以看到用这种方式等效除法得到的除数可表示为$o=\frac{2^{32+n}}{m}$四舍五入
误差分析略去

结论

类似以下代码可判断为整型除法

```
;寄存器不一定与下面相同
mov edx,MagicNumber     ;此处MagicNumber<0x80000000
mov ecx,[var]           ;var中存放被除数
mov eax,ecx
sar ecx,0x1f
imul edx
sar edx,n
sub edx,ecx
;edx中值为除法结果

;或者当MagicNumber>=0x80000000时
mov edx,MagicNumber
mov ecx,[var]
imul edx
add edx,ecx
sar ecx,0x1f
sar edx,n
sub edx,ecx
;edx中存放结果
```

$o\approx\frac{2^{32+n}}{m}$ 由此可以还原出除法$\frac{a}{o}$

除数为负的2的整数次幂

与除数为正的2的整数次幂优化方式相同，只不过会增加一句取反操作

除数为负的非2的整数次幂

与除数为正的非2的整数次幂优化方式相同，不同的是最后sub语句寄存器顺序相反

小结

对于C/C++整数除法的优化主要内容在于被除数为变量而除数为常量除法的优化。其中有很多细节可以探究，本文是从反汇编代码中总结出一些规律，这些规律可以通过严格的数学证明加以论证。由于我数学证明很渣，无法详细地叙述，感兴趣的读者可以参考《C++反汇编与逆向分析技术揭秘》这本书的第四章。书中反反汇编得到的代码结果和我反汇编得到的代码结果是不同的，这大概和编译器有关，但实际上的优化策略的原理相同。

参考资料

1. 《C++反汇编与逆向分析技术揭秘》