加减乘除

IDA Pro

相关使用资料：

https://23r3f.github.io/2019/02/15/IDA%E4%BD%BF%E7%94%A8%E6%8A%80%E5%B7%A7%E6%80%BB%E7%BB%93/

制作签名：https://bbs.pediy.com/thread-224499.htm

基本运算

加法

1. 常量 + 常量
    - 常量传播：将编译期间可计算出结果的变量转化为常量

```
int n = 1;
printf("%d\n", n);
// 变为
printf("%d\n", 1);
```

    - 常量折叠：当计算公式中出现多个常量进行计算的情况时候，且编译器可以在编译期计算处结果，则常量计算会被计算结果替代

```
int n = 1 + 2 + 3;
printf("%d\n", n);
// 变为
printf("%d\n", 6);
```

2. 常量 + 变量
    - 按指令翻译
3. 变量 + 变量
    - 按指令翻译
4. 减法
    - 按指令翻译
    - 可能将减法转化为加法

乘法

1. 乘2的幂
    - 转化为shl指令
2. 乘非2的幂
    - 转化为加法(add、lea)和位移(shl)指令
3. 变量 * 变量
    - imul、mul指令，可能双操作数也可能单操作数

除法

有符号和无符号混除，使用的都是无符号div

1. 常量 / 变量
    - div、idiv，没有优化空间的
2. 变量 / 变量
    - div、idiv，没有优化空间的
3. 变量 / 常量
    - 无符号除以2的幂，shr
    - 数学优化：
        - 向下取整：往**负无穷**的方向最接近 x 的值，`floor`函数
        - 向上取整：往**正无穷**的方向最接近 x 的值，`ceil`函数，`>>`是向下取整
        - 向0取整：往**0**方向最接近 x 的值
        - 取模的结果符号看**被除数**

推导 7　设有 a、b 两整数，

当 $b>0$ 时，有：

$$\left\lfloor \frac{a}{b} \right\rfloor = \left\lceil \frac{a-b+1}{b} \right\rceil \quad 且 \quad \left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a+b-1}{b} \right\rfloor$$

当 $b<0$ 时，有：

$$\left\lfloor \frac{a}{b} \right\rfloor = \left\lceil \frac{a-b-1}{b} \right\rceil \quad 且 \quad \left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a+b+1}{b} \right\rfloor$$

        - 例子1：除以2的幂

```
// i / 8
int res;
__asm {
    mov eax, i
    cdq // 符号扩展，正数edx为0，  负数edx为0xffffffff
    and edx, 7  // 正数edx为0，  负数edx为7，属于b > 0，负数计算取上整，正数计算不变
    add eax, edx
    sar eax, 3
    mov res, eax
}

// 代码公式，验证2^n - 1 = m 成立，则还原为 i / 2 ^ n
```

```
mov eax, i
cdq
and edx, m
add eax, edx
sar eax, n

// 幺蛾子 i / 2
mov eax, i
cdq
sub eax, edx
sar eax, 1
```

- 例子2：除以非2的幂

```
// A / C
// -> (A * (2 ^ n / C)) >> n
// -> (2 ^ n / C) == MagicNum
// -> (A * MagicNum) >> n，n由编译器决定
// 则C = (2 ^ n) / MagicNum，结果C是向上取整

// i（无符号） / 3
mov eax, 0aaaaaaabh
mul i
shr edx, 1  // (A * Magic) >> 33，直接丢弃了eax
            // 直接使用edx相当于移了32位，对edx移动n位相当于移了32 + n位

// i（有符号） / 5
mov eax, 66666667h  // 第一部分照常计算
imul i
sar edx, 1

mov eax, edx    // 调整，下整 -> 上整
shr eax, 1Fh    // 正数，eax = 0，  负数eax = 1
add eax, edx
                // gcc的调整可能是
                // mov eax, edx
                // cdq
                // sub eax, edx
```

总结：

```
mov eax, MagicNumber
imul …
sar edx, …
mov reg, edx
shr reg, 1Fh
add edx, reg
; 此后直接使用 edx 的值，eax 弃而不用
```

当遇到以上指令序列时，基本可判定是除法优化后的代码，其除法原型为 a 除以常量 o，imul 可表明是有符号计算，其操作数是优化前的被除数 a。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 $o = \frac{2^n}{c}$，将 MagicNumber 作为 c 值代入公式求解常量除数 o 的近似值，四舍五入取整后，即可恢复除法原型。

- 例子3：Magic超过4字节

```
// i（无符号） / 7
mov ecx, i
mov eax, 24924925h
mul ecx
sub ecx, edx
shr ecx, 1
add ecx, edx
shr ecx, 2
```

数数一共移动了几次，ecx 一共右移了 3 位，而且直接与 edx 运算并作为结果使用，因此还得加上乘法的低 32 位，共计 3 5 位，故 n 的取值为 3。已知 c 值为常量 24924925h，根据上述推导，可得：

$$c = \frac{2^{32+n}}{o} - 2^{32} = \frac{2^{32+3}}{o} - 2^{32} = 24924925\text{h}$$

解方程求得：

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

于是，我们反推出优化前的高级代码为：

```
printf("nVarTwo / 7 = %d\r\n", argc / 7);
```

这个例子就是`Magic = Magic + 2^32`，适用于`Magic`超过**4字节**使用此优化，将原本的**超过4字节**的`Magic`转换为**4字节内**的`Magic + 2^32`

以后遇到这种**乘减移加移**序列，Magic最高位（第33位）补1当新的Magic根据以前的公式计算

- 例子4：照常还原

```
printf("%d", argc / 7);
```

注意，这里的 argc 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：

```
mov eax, MagicNumber（大于 7fffffffh）
imul reg
add edx, reg
sar edx, ···
mov reg, edx
shr reg, 1Fh
add edx, reg
; 此后直接使用 edx 的值
```

- 例子5：除数为负，且为2的幂

```
// i / -4 = -(i / 4)
// 对比以前多了一个neg指令
```

- 例子6：除数为负，且为非2的幂

Magic为负，且不存在add调整，对Magicm取反加一得新的Magic，照常还原，得到的除数是正，记得加负号

```
printf("%d", argc / -5);
```

总结：

```
mov eax, MagicNumber（大于 7fffffffh）
imul reg
sar edx, …
mov reg, edx
shr reg, 1Fh
add edx, reg
; 此后直接使用 edx 的值
```

- 例子7：除以-7，但Magic为正，且有sub调整代码

  对Magicm取反加一得新的Magic，照常还原，得到的除数是正，记得加负号

```
printf("%d", argc / -7);
```

总结：

```
mov eax, MagicNumber（小于等于 7fffffffh）
imul reg
sub edx, reg
sar edx, …
mov reg, edx
shr reg, 1Fh
add edx, reg
; 此后直接使用 edx 的值
```

除法总结

- 判断除数正负
  1. Magic为正，`imul`和`sar`之间无调整，则除数为正
  2. Magic为正，`imul`和`sar`之间有`sub edx, xxx`调整，则除数为负
  3. Magic为负，`imul`和sar之间无调整，则除数为负
  4. Magic为负，`imul`和`sar`之间有`add edx, xxx`调整，则除数为正
- 还原除数
  1. 除数为正，则按之前的知识点还原
  2. 除数为负，则对Magic求补后还原出除数的绝对值