德·摩根定律

德.摩根定律的定义如下:

$(\neg A) \vee (\neg B) = \neg(A \wedge B)$

$(\neg A) \wedge (\neg B) = \neg(A \vee B)$

文字描述如下:

“非A”或者“非B”，和非“A与B”是等价的。

“非A”并且“非B”，和非“A或B”是等价的。

使用对偶性可以很方便的记忆和使用这个定律。.

我们知道如下关系呈现对偶关系，可以认为是“非”的关系:

true	↔	false
A	↔	¬A
∧	↔	∨

那么将

$A \wedge B$

利用对偶关系对应改写可以得到:

$(\neg A) \vee (\neg B)$

.那么可以对应得到:

$(\neg A) \vee (\neg B) = \neg(A \wedge B)$

通过应用德.摩根定律，有时可以简化一些逻辑问题，而使用对偶性来理解这个定律，更好的理解的同时也有利于记忆。
扩展一下：三值的德.摩根定律,

(!A) || (!B) = ! (A && B)
(!A) && (!B) = ! (A || B)

运用德·摩根定律，可将 if 语句进行如下变形。

```
if (!(x >= 0 && y >= 0)) {
    ...
}
      ↓
if (x < 0 || y < 0)) {
    ...
}
```


vmp里面只有1个逻辑运算指令 not_not_and 设这条指令为P
P(a,b) = ~a & ~b

这条指令的神奇之处就是能模拟 not and or xor 4条常规的逻辑运算指令
怕忘记了，直接给出公式，后面的数字指需要几次P运算

not(a) = P(a,a) 1
and(a,b) = P(P(a,a),P(b,b)) 3
or(a,b) = P(P(a,b),P(a,b)) 2
xor(a,b) = P(P(P(a,a),P(b,b)),P(a,b)) 5

上面的次数应该是最少需要的次数了，当然也可以展开，那样就更加复杂了
vmp用1条指令模拟了4条指令，因此逆向起来比较复杂，如果中间夹杂垃圾运算，那么工程量非同小可
下面来证明一下上面4条等式

not(a) = ~a = ~a & ~a = P(a,a)
and(a,b) = a & b = ~(~a) & ~(~b) = P(not(a),not(b)) = P(P(a,a),P(b,b))
or(a,b) = a | b = ~(~(a|b)) = ~(~a & ~b) = ~P(a,b) = P(P(a,b),P(a,b))
xor(a,b) = ~a & b | a & ~b = ~(~(~a & b | a & ~b)) = ~(~(~a & b) & ~(a & ~b)) = ~((a | ~b) & (~a | b)) = ~(1 | 1 | a & b | ~a & ~b) = ~(a & b) & ~(~a & ~b) = P(and(a,b),P(a,b)) = P(P(P(a,a),P(b,b)),P(a,b))

上面的xor是最复杂的，不过简化后也只需要5次运算就可以实现了

至于eflag，eflag是根据结果来定的，由于都是逻辑运算，所以最后取一下eflag即可

在某修改版的vm中，还可以看到另一个强大的指令 not_not_or 设这条指令为Q
Q(a,b) = ~a | ~b

同样，这一条指令可以模拟4条常规的逻辑运算指令
怕忘记了，直接给出公式，后面数字表示需要几次Q运算

not(a) = Q(a,a) 1
and(a,b) = Q(Q(a,b),Q(a,b)) 2
or(a,b) = Q(Q(a,a),Q(b,b)) 3
xor(a,b) = Q(Q(Q(a,a),b),Q(a,Q(b,b))) 5

基本和上面P指令相同，效率没什么变化
只对最复杂的xor证明一下，以防忘记

xor(a,b) = ~a & b | a & ~b = ~(~(~a & b | a & ~b)) = ~(~(~a & b) & ~(a & ~b)) = ~((~(~a) | ~b) & (~a | ~(~b))) = ~(~(~a) | ~b) | ~(~a | ~(~b)) = Q(Q(not(a),b),Q(a,not(b))) = Q(Q(Q(a,a),b),Q(a,Q(b,b)))

实在太难了，完全搞不定啊