15-2

1

a

```
gen[B1] = {d1, d2}
kill[B1] = {d8,d10,d11}
gen[B2] = {d3,d4}
kill[B2] = {d5,d6}
gen[B3] = {d5}
kill[B3] = {d4,d6}
gen[B4] = {d6,d7}
kill[B4] = {d4,d5,d9}
gen[B5] = {d8, d9}
kill[B5] = {d2,d11,d7}
gen[B6] = {d10,d11}
kill[B6] = {d1,d2,d8}
```

IN，OUT轮一：

	IN	OUT
B1		{d1, d2}
B2	{d1, d2}	{d1, d2, d3, d4}
B3	{d1, d2, d3, d4}	{d1, d2, d3, d5}
B4	{d1, d2, d3, d5}	{d1, d2, d3, d6, d7}
B5	{d1, d2, d3, d4, d5}	{d1, d3, d4, d5, d8, d9}
B6	{d1, d3, d4, d5, d8, d9}	{d3, d4, d5, d9, d10, d11}

IN，OUT轮二：

	IN	OUT
B1		{d1, d2}
B2	{d1, d2, d3, d4, d5, d8, d9}	{d1, d2, d3, d4, d8, d9}
B3	{d1, d2, d3, d4, d6, d7, d8, d9}	{d1, d2, d3, d5, d7, d8, d9}
B4	{d1, d2, d3, d5, d7, d8, d9}	{d1, d2, d3, d6, d7, d8}
B5	{d1, d2, d3, d4, d5, d7, d8, d9}	{d1, d3, d4, d5, d8, d9}
B6	{d1, d3, d4, d5, d8, d9}	{d3, d4, d5, d9, d10, d11}

IN，OUT轮三：

	IN	OUT
B1		{d1, d2}
B2	{d1, d2, d3, d4, d5, d8, d9}	{d1, d2, d3, d4, d8, d9}
B3	{d1, d2, d3, d4, d6, d7, d8, d9}	{d1, d2, d3, d5, d7, d8, d9}
B4	{d1, d2, d3, d5, d7, d8, d9}	{d1, d2, d3, d6, d7, d8}
B5	{d1, d2, d3, d4, d5, d7, d8, d9}	{d1, d3, d4, d5, d8, d9}
B6	{d1, d3, d4, d5, d8, d9}	{d3, d4, d5, d9, d10, d11}

所以四中的IN，OUT就是最终的IN，OUT。

b

```
e_gen[B1] = {1, 2}
e_kill[B1] = {a + b, c - a, a - d, b * d, b + d}
e_gen[B2] = {a + b, c - a}
e_kill[B2] = {b + d, b * d, c - a, a - d}
e_gen[B3] = {}
e_kill[B3] = {b + d, b * d, a - d}
e_gen[B4] = {a + b}
kill[B4] = {b * d, a - d, b + d}
e_gen[B5] = {c - a}
e_kill[B5] = {a + b, b + d, b * d, e + 1}
e_gen[B6] = {a - d}
e_kill[B6] = {b * d, b + d, a + b, c - a}
```

U = {1, 2, a + b, c - a, b + d, e + 1, b * d, a - d}

IN，OUT轮一：

	IN	OUT
B1		{1, 2}
B2	{1, 2}	{1, 2, a + b, c - a}
B3	{1, 2, a + b, c - a}	{1, 2, a + b, c - a}
B4	{1, 2, a + b, c - a}	{1, 2, a + b, c - a}
B5	{1, 2, a + b, c - a}	{1, 2, c - a}
B6	{1, 2, c - a}	{1, 2, a - d}

IN，OUT轮二：

	IN	OUT
B1		{1, 2}
B2	{1, 2}	{1, 2, a + b, c - a}
B3	{1, 2, a + b, c - a}	{1, 2, a + b, c - a}
B4	{1, 2, a + b, c - a}	{1, 2, a + b, c - a}
B5	{1, 2, a + b, c - a}	{1, 2, c - a}
B6	{1, 2, c - a}	{1, 2, a - d}

故而IN，OUT就是轮二中的IN，OUT。

C

```
use[B1] = {}
def[B1] = {a, b}
use[B2] = {a, b}
def[B2] = {c, d}
use[B3] = {b, d}
def[B3] = {}
use[B4] = {a, b, e}
def[B4] = {d}
use[B5] = {a, b, c}
def[B5] = {e}
use[B6] = {b, d}
def[B6] = {a}
```

IN，OUT轮一：

	OUT	IN
B6		{b, d}
B5	{b, d}	{a, b, c, d}
B4		{a, b, e}
B3	{a, b, c, d, e}	{a, b, c, d, e}
B2	{a, b, c, d, e}	{a, b, e}
B1	{a, b, e}	{e}

IN，OUT轮二：

	OUT	IN
B6		{b, d}
B5	{a, b, d, e}	{a, b, c, d}
B4	{a, b, c, d, e}	{a, b, c, e}
B3	{a, b, c, d, e}	{a, b, c, d, e}
B2	{a, b, c, d, e}	{a, b, e}
B1	{a, b, e}	{e}

IN，OUT轮三：

	OUT	IN
B6		{b, d}
B5	{a, b, d, e}	{a, b, c, d}
B4	{a, b, c, d, e}	{a, b, c, e}
B3	{a, b, c, d, e}	{a, b, c, d, e}
B2	{a, b, c, d, e}	{a, b, e}
B1	{a, b, e}	{e}

故而IN，OUT就是轮三中的IN，OUT。

2

现象

编译没有开二级优化的时候，生成的目标程序运行起来会报segmentation fault。

编译开了二级优化之后，生成的目标程序会陷入死循环，但是不会报segmentation fault。

原因

开了二级优化之后，f函数中的`call *%rdi`在二级优化后的汇编中变成了`jmp *%rdi`，故而一个无休止的递归就在结构上转变成了一个死循环，对于一个死循环结构，是不会报`segmentation fault`的。