



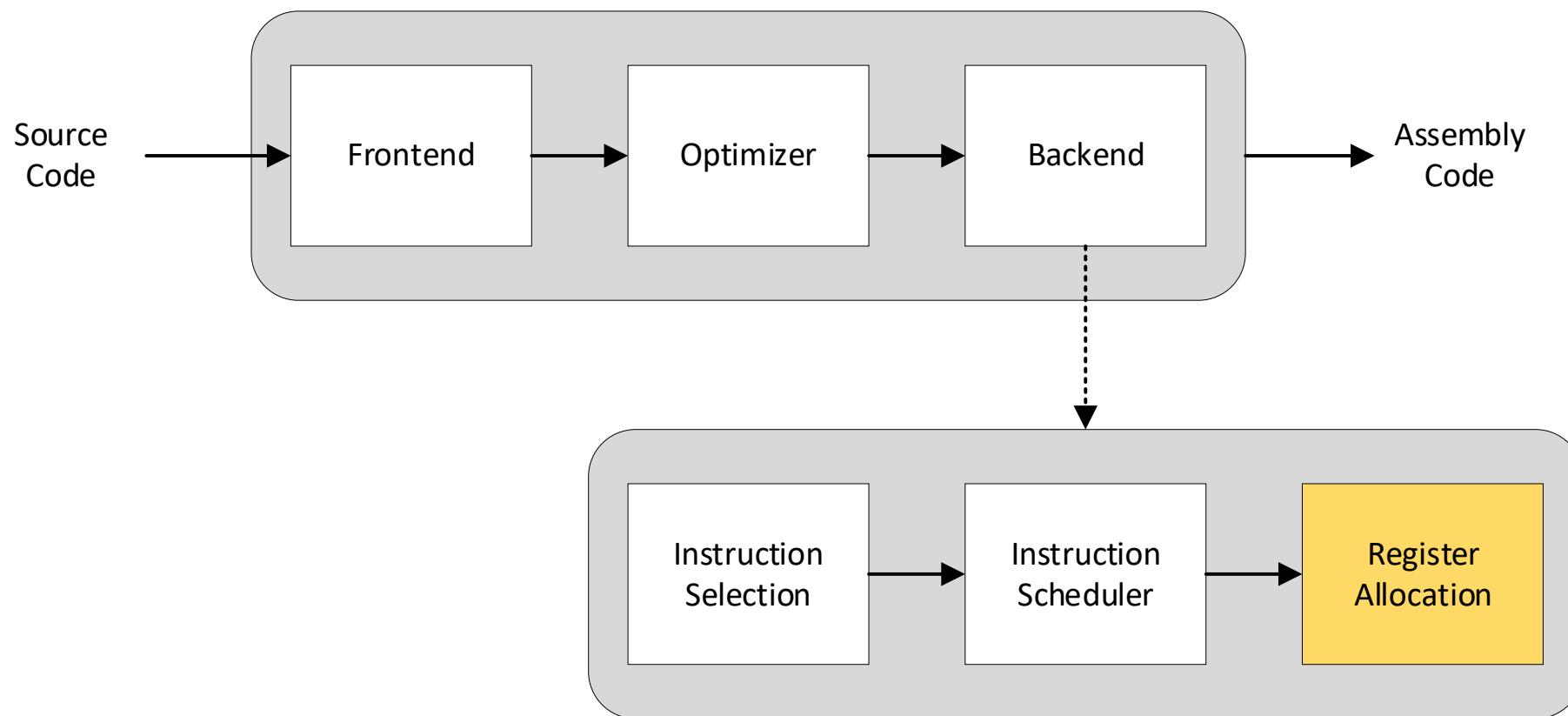
寄存器分配-图着色

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心
先进技术研究院、计算机科学与技术学院

2025年12月08日

- 寄存器是CPU中的稀有资源，如何高效的分配这一资源是一个至关重要的问题。

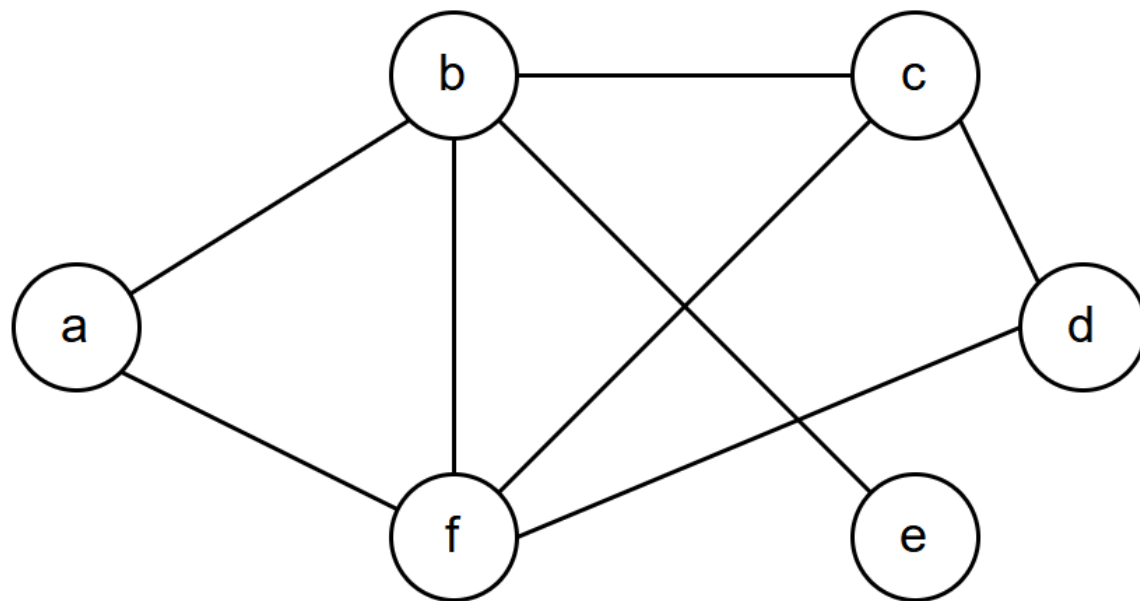


□图着色算法

- 最经典的寄存器分配算法之一
- 将寄存器分配问题转化为图的节点着色问题
- 该算法适用于复杂的控制流程和循环结构，但计算复杂度较高

□图着色算法

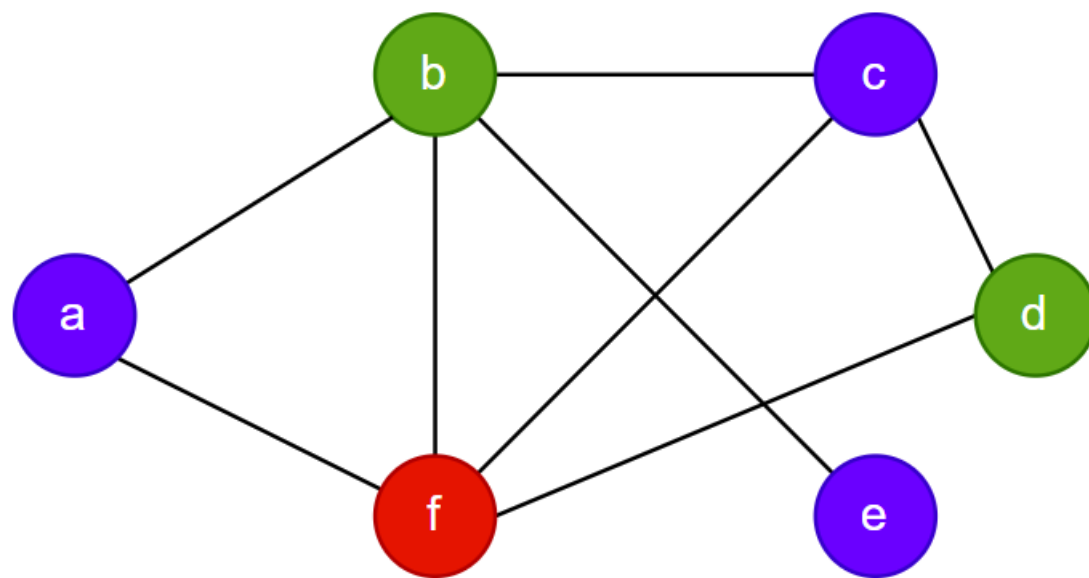
- 图中一共6个结点，被边相连的结点表示有相邻的关系
- 有相邻关系的结点不可以被染成同一种颜色
- 至少需要多少种颜色完成图着色？



图着色问题

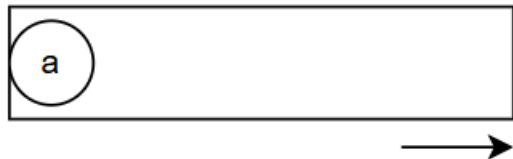
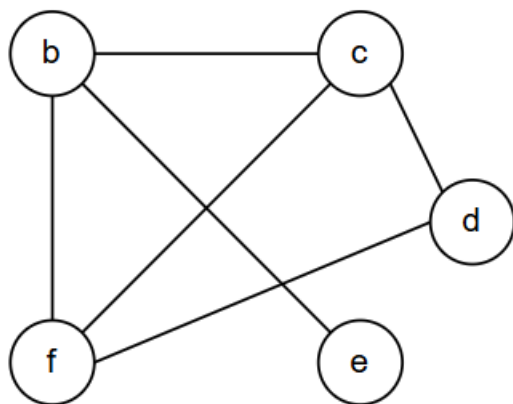


- 假设一共有红绿蓝3种颜色可供选择，一个可能的染色的结果
- 若用3个颜色可完成着色，可以称此图为3可着色图

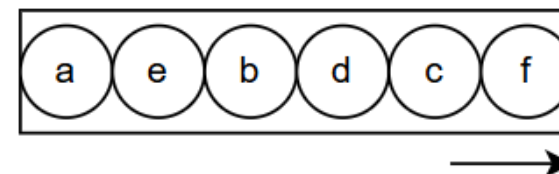


判断图为 k 可着色图的算法

- 将度低于 k 的节点依次删除（同时有关的边也删除），节点入栈
- 重复第一步，直到所有节点都被删除，则为 k 可着色图



将度低于3的结点删除，同时有关的边也进行删除，加入栈中，这里删除了a



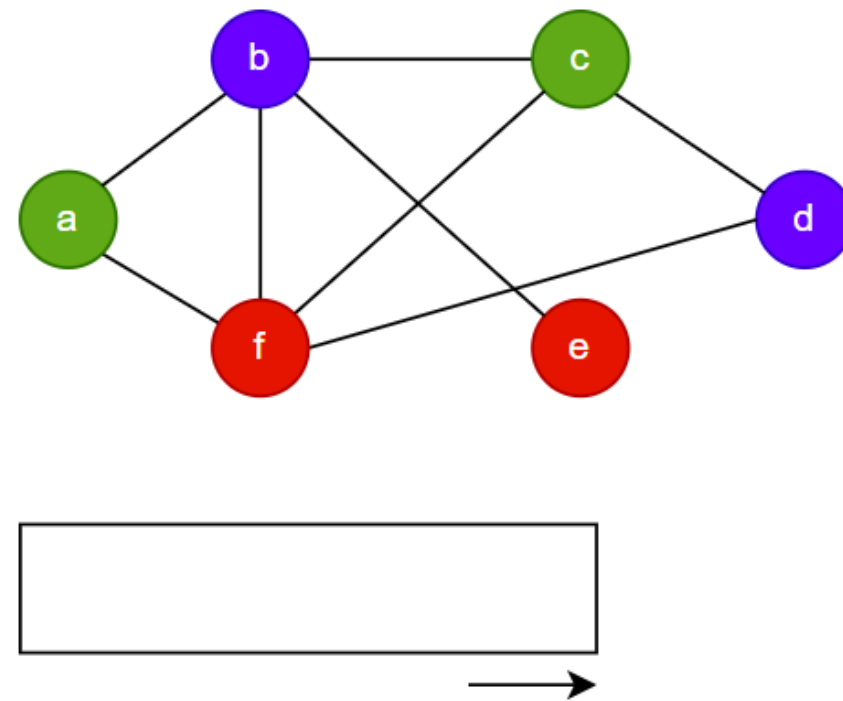
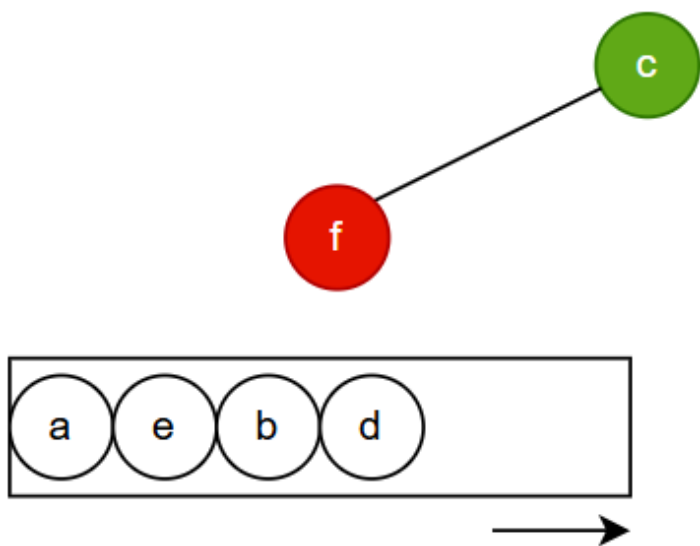
重复操作，直到所有结点放入了栈中
此时图已经为空

图着色问题



□图着色

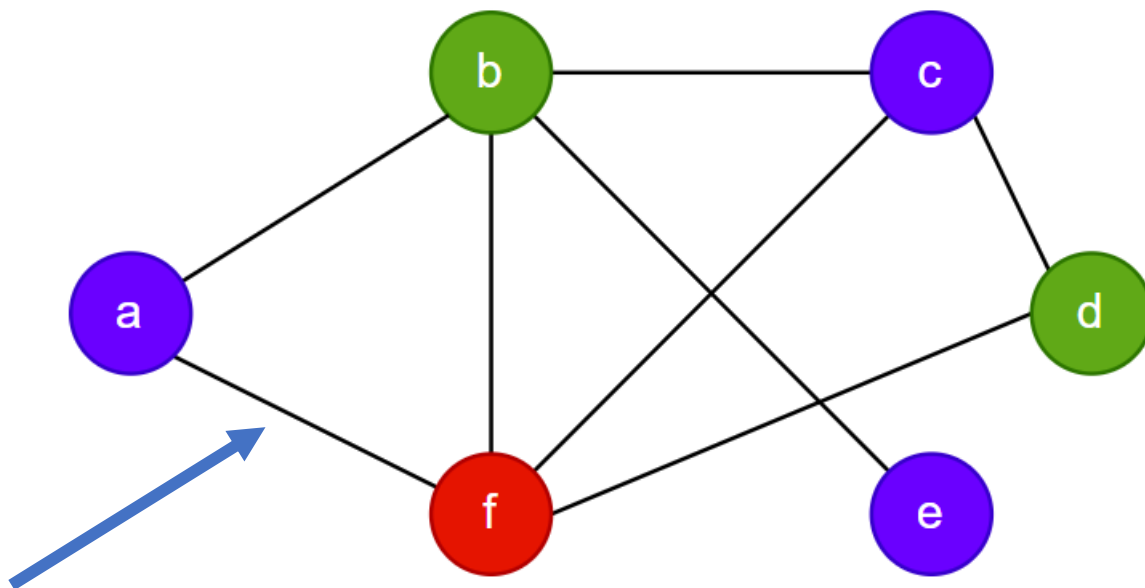
- 从栈中将结点取出，取出时进行着色
- 如果两者有连接关系，那么不可以染成同样的颜色



利用图着色进行寄存器分配的思路



- ❑ 寄存器分配也是类似思路
- ❑ 图中的节点可以被看作是中间代码中的虚拟寄存器/变量
- ❑ 颜色可以被看作是物理寄存器，每个物理寄存器对应一种颜色
- ❑ 分配物理寄存器就是对结点进行着色

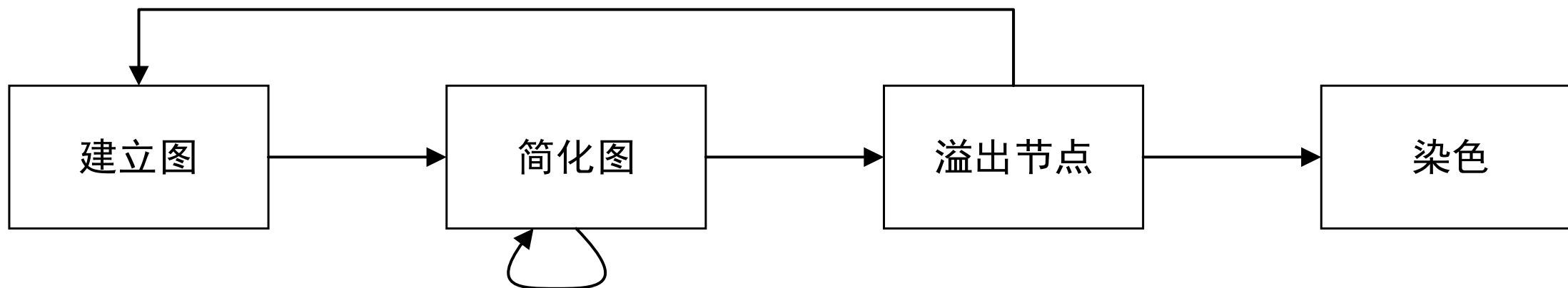


边代表了两个虚拟寄存器
不可以分配到同一个物理
寄存器

□ 算法步骤

■ 建立图

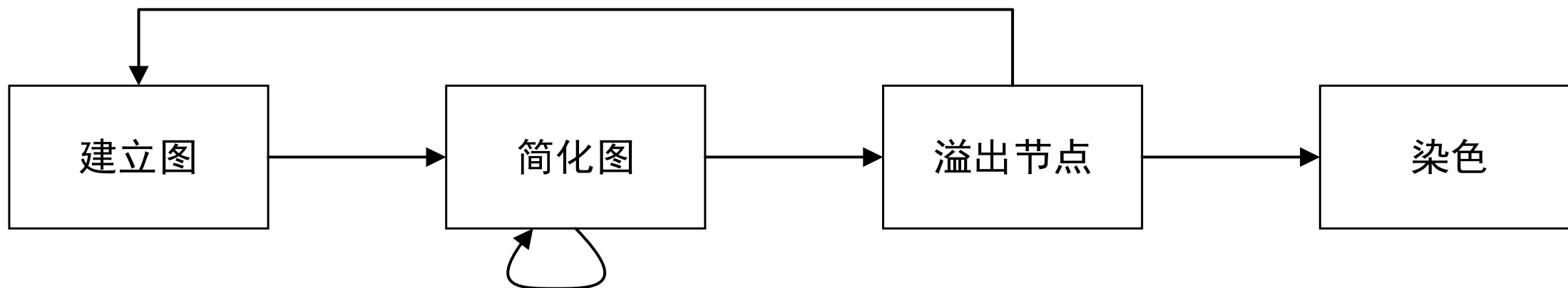
- 根据每个变量的活跃区间建立寄存器干涉图
- 图中节点对应其活跃变量，活跃区间互相冲突的节点之间建立边



□ 算法步骤

■ 简化图

- 图中的节点遍历
- 如节点度少于 k （可用的物理寄存器的数量），则将节点及邻边从图删除，并将节点压入栈
- 重复步骤直到图中所有节点度数大于等于 k ，或者得到空图为止



□ 算法步骤

■ 溢出节点

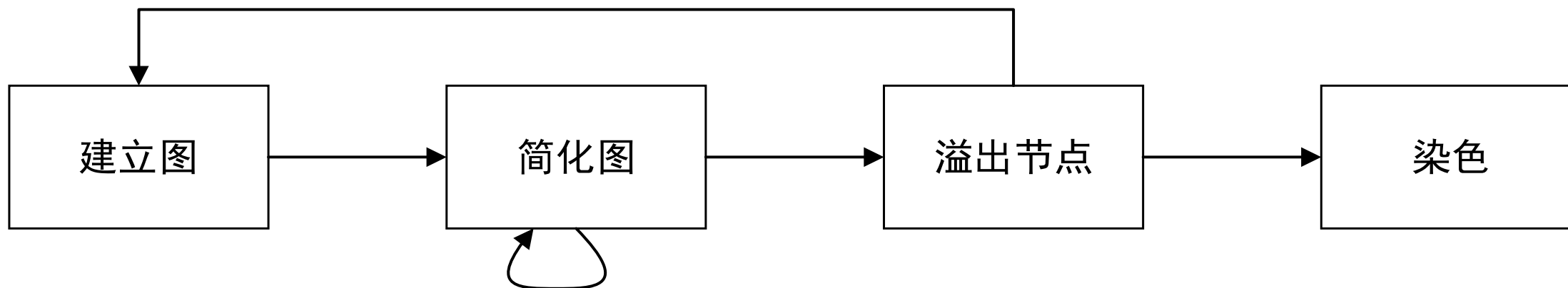
➤ 如简化最后图非空，需要选择一节点溢出（放入内存）

◆ 节点选择

✓ 比较简单的实现是选择删除在代码中出现次数较少的节点

✓ 根据综合代价选择

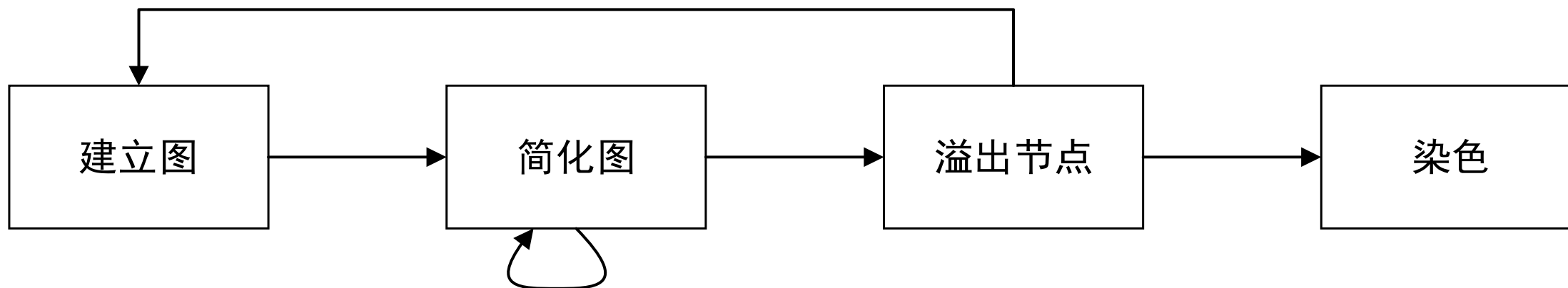
➤ 重新建立图和化简图



□ 算法步骤

■ 染色

- 算法按栈中节点出栈顺序进行图着色
- 图中的所有节点染上有且只有一种颜色



寄存器图着色示例



□ 假设系统存在5个寄存器

```
int main(void){  
    int a;    int b;  
    int c;    int d;  
    int e;    int f;  
    int g;    int h;  
    int k;  
    a = 1;  
    b = 2;  
    c = 3;  
    d = a+b;  
    e = 2*d;  
    g = d+e;  
    k = d+e+2;  
    f = e+d/2;  
    h = k+g+d+f;  
    return d;  
}
```

C代码

```
target triple = "x86_64-pc-linux-gnu"  
declare i32 @input()  
declare void @output(i32)  
declare void @outputFloat(float)  
declare void @neg_idx_except()  
define i32 @main() {  
label_entry:  
0 %op11 = add i32 1, 2  
2 %op13 = mul i32 2, %op11  
4 %op16 = add i32 %op11, %op13  
6 %op19 = add i32 %op11, %op13  
8 %op20 = add i32 %op19, 2  
10 %op23 = sdiv i32 %op11, 2  
12 %op24 = add i32 %op13, %op23  
14 %op27 = add i32 %op20, %op16  
16 %op29 = add i32 %op27, %op11  
18 %op31 = add i32 %op29, %op24  
20 ret i32 %op11  
}
```

IR

□ 变量活跃区间

变量	活跃区间
%op11	[1,21)
%op13	[3,13)
%op16	[5,15)
%op19	[7,9)
%op20	[9,15)
%op23	[11,13)
%op24	[13,19)
%op27	[15,17)
%op29	[17,19)

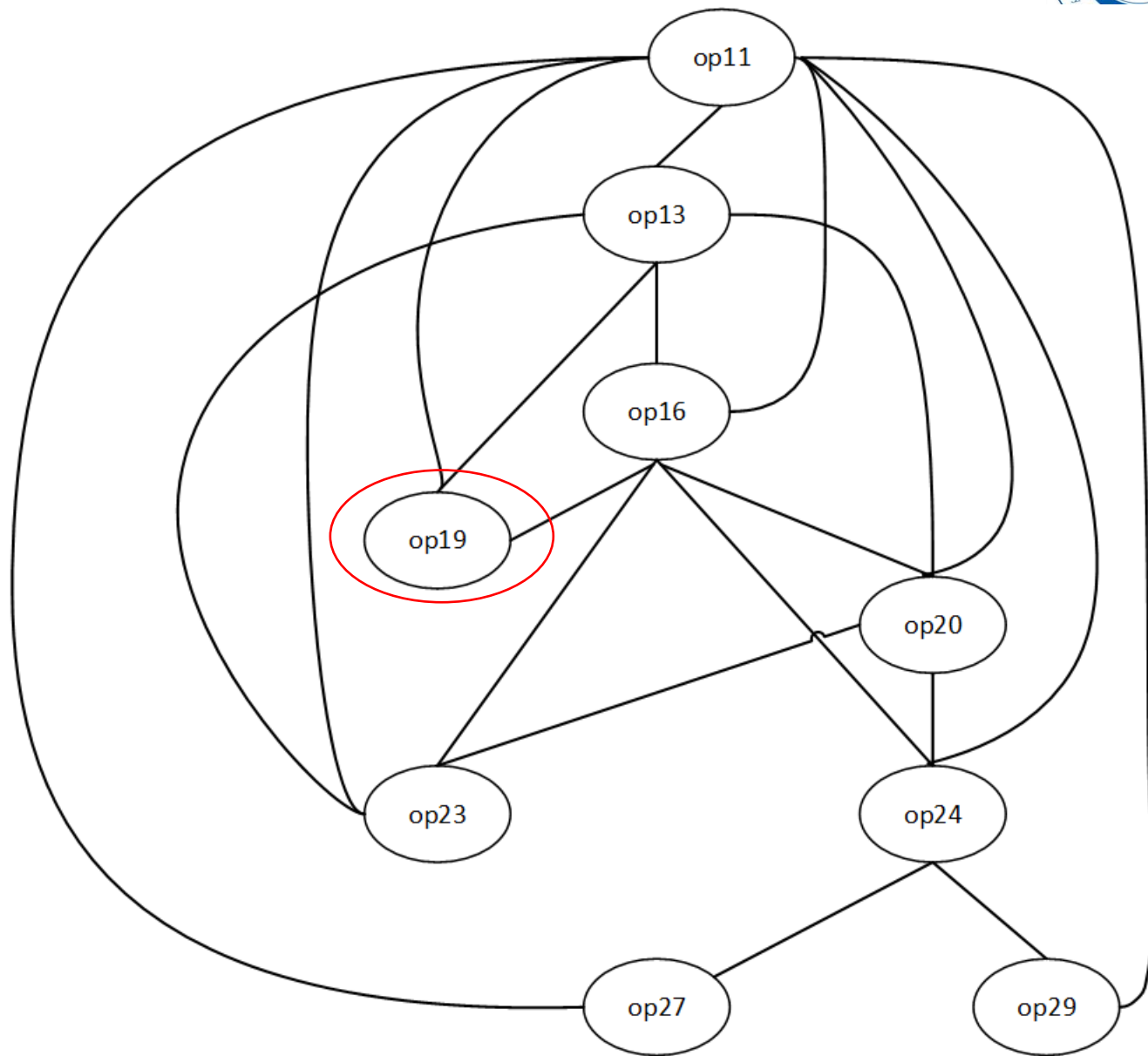
```
target triple = "x86_64-pc-linux-gnu"
declare i32 @input()
declare void @output(i32)
declare void @outputFloat(float)
declare void @neg_idx_except()
define i32 @main() {
label_entry:
0 %op11 = add i32 1, 2
2 %op13 = mul i32 2, %op11
4 %op16 = add i32 %op11, %op13
6 %op19 = add i32 %op11, %op13
8 %op20 = add i32 %op19, 2
10 %op23 = sdiv i32 %op11, 2
12 %op24 = add i32 %op13, %op23
14 %op27 = add i32 %op20, %op16
16 %op29 = add i32 %op27, %op11
18 %op31 = add i32 %op29, %op24
20 ret i32 %op11
}
```

寄存器图着色示例



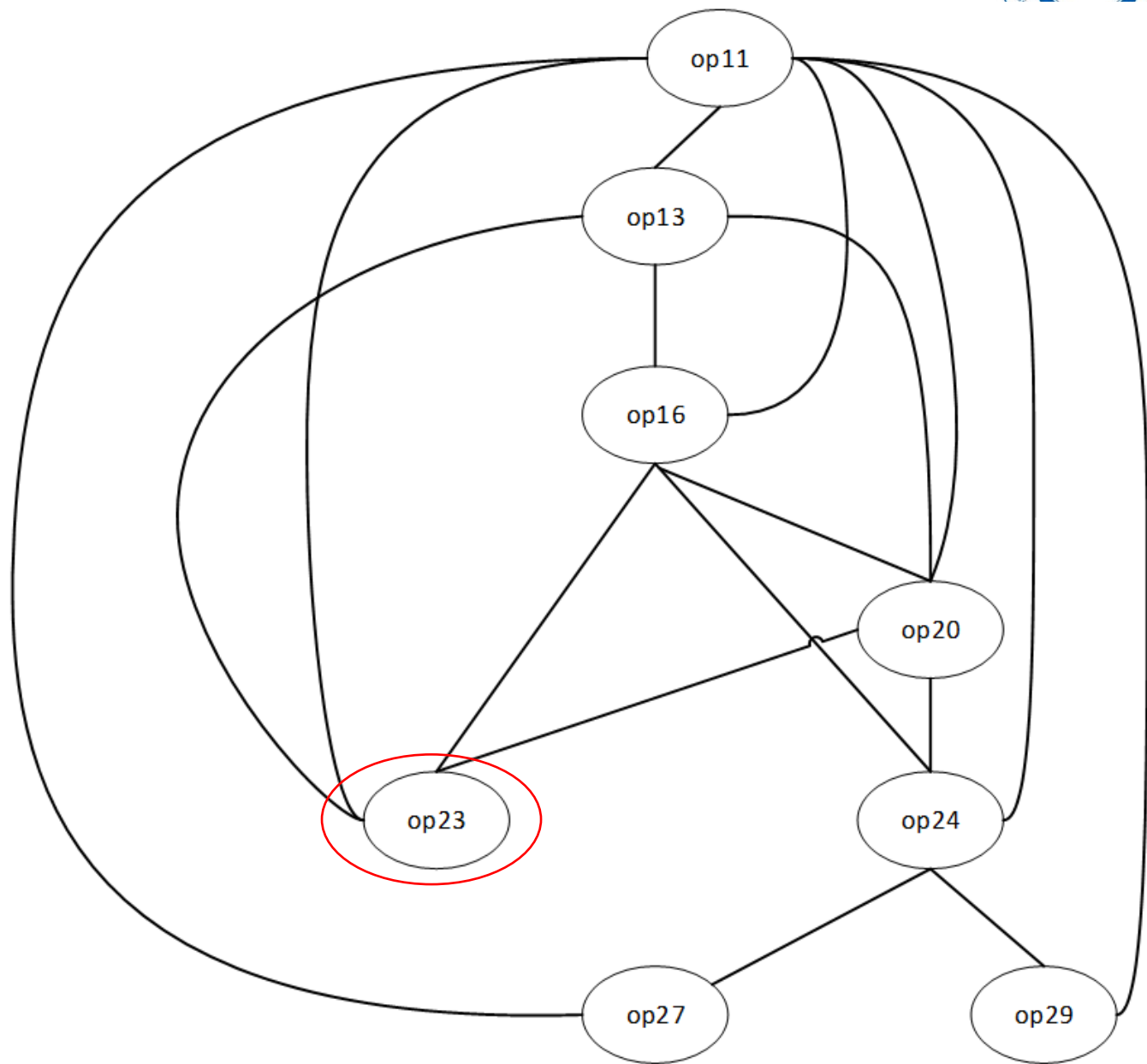
□ 建立干涉图

变量	活跃区间
%op11	[1,21)
%op13	[3,13)
%op16	[5,15)
%op19	[7,9)
%op20	[9,15)
%op23	[11,13)
%op24	[13,19)
%op27	[15,17)
%op29	[17,19)



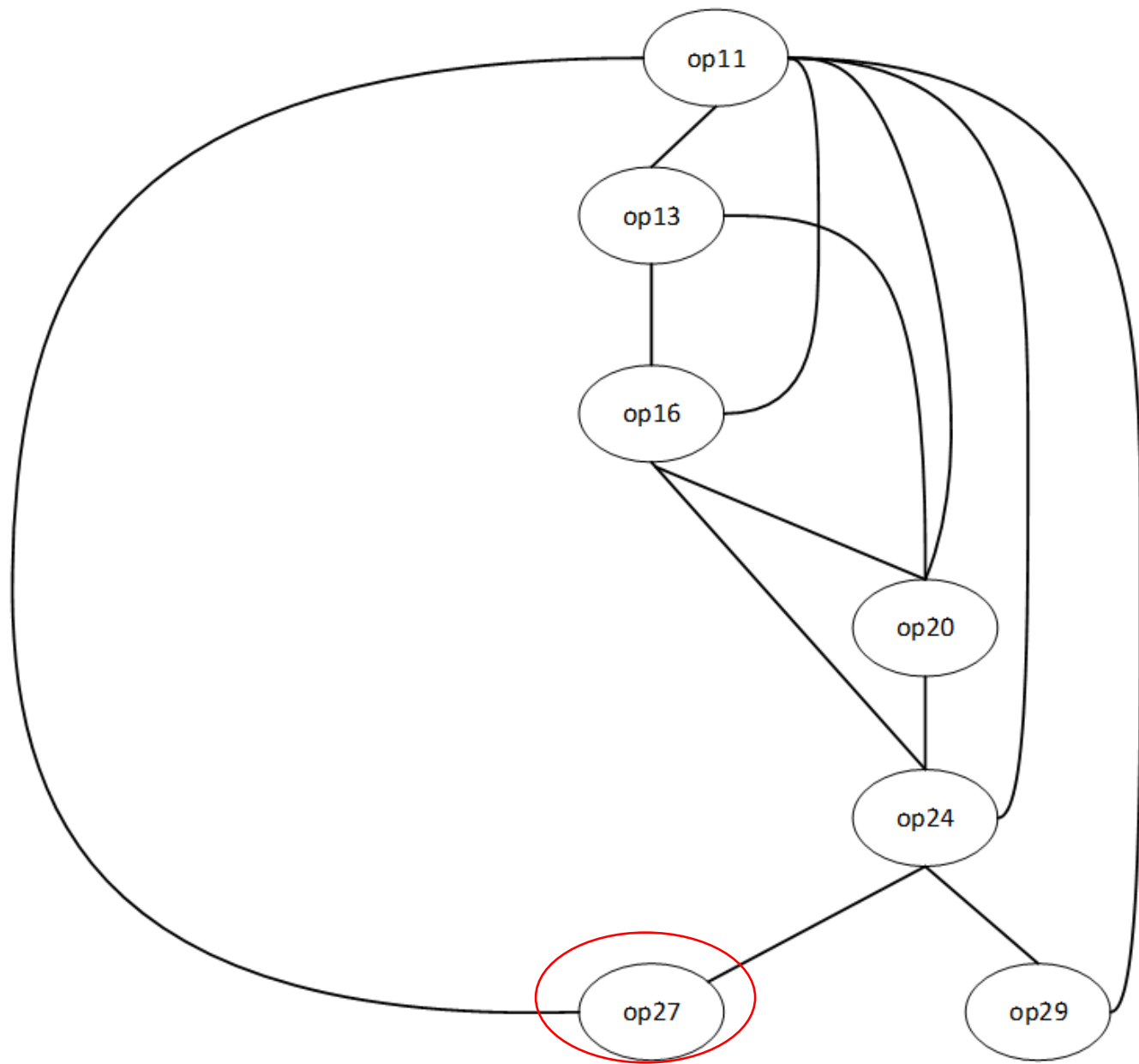
化简图

■ 删除%op19



化简图

- 删除%op19
- 删除%op23



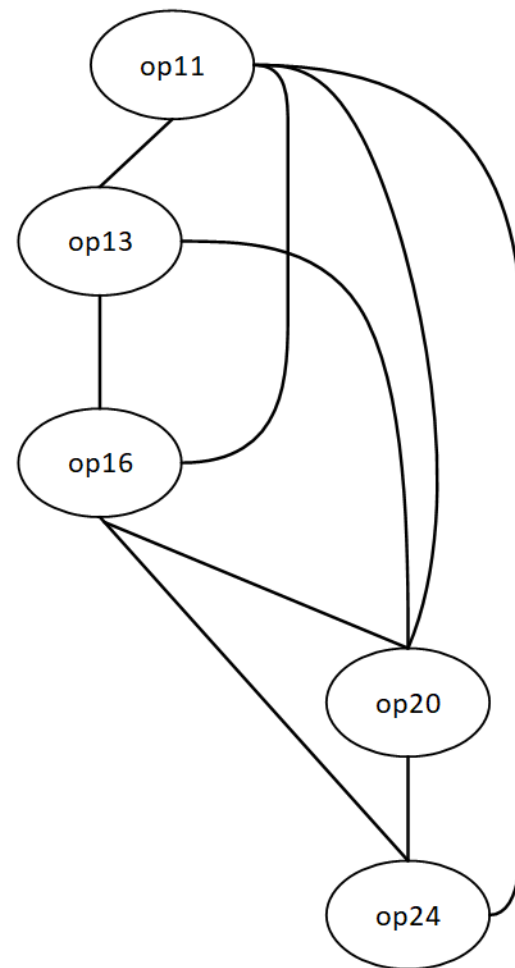
■ 化简图

- 删除%op19
- 删除%op23
- 删除%op27



化简图

- 删除%op19
- 删除%op23
- 删除%op27
- 删除%op29



□化简图

- 删除%op19
- 删除%op23
- 删除%op27
- 删除%op29
- 再将%op13, %op16, %op20, %op24, %op11删除, 并压入栈中
- 最后得到一个空图

□染色

■ 按照入栈的

%op23

%op27

%op29

%op13

%op16

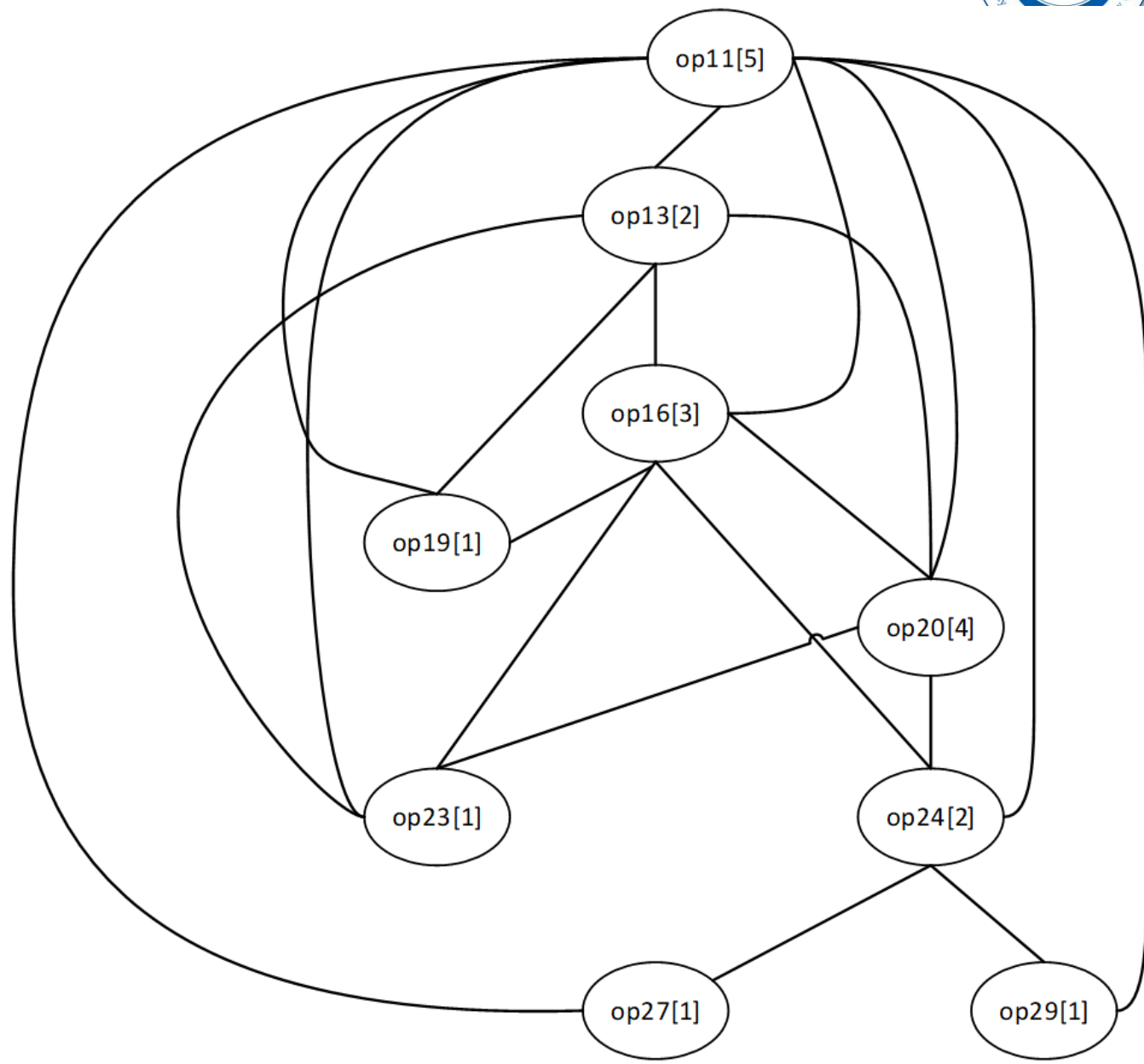
%op20

%op24

%op11的出栈顺序

相邻节点不赋予相同颜色的规则

进行着色，可以得到一张**5**-着色图



寄存器图着色示例2

□假设系统存在5个寄存器

```
int main(void){
    int a;    int b;
    int c;    int d;
    int e;    int f;
    int g;    int h;
    int k;    int i;
    a = 1;    b = 2;
    c = 3;
    d = a+b;
    e = 2*d;
    g = d+e;
    k = d+e+2;
    i = d+e/2;
    f = e+d/2;
    h = k+g+d+f+i;
    return d;
}
```

C代码

```
target triple = "x86_64-pc-linux-gnu"
declare i32 @input()
declare void @output(i32)
declare void @outputFloat(float)
declare void @neg_idx_except()
define i32 @main() {
label_entry:
0  %op12 = add i32 1, 2
2  %op14 = mul i32 2, %op12
4  %op17 = add i32 %op12, %op14
6  %op20 = add i32 %op12, %op14
8  %op21 = add i32 %op20, 2
10 %op24 = sdiv i32 %op14, 2
12 %op25 = add i32 %op12, %op24
14 %op28 = sdiv i32 %op12, 2
16 %op29 = add 732 %op14, %op28
18 %op32 = add i32 %op21, %op17
20 %op34 = add i32 %op32, %op12
22 %op36 = add i32 %op34, %op29
24 %op38 = add i32 %op36, %op25
26 ret i32 %op12
}
```

IR

寄存器图着色示例2

□ 变量活跃区间

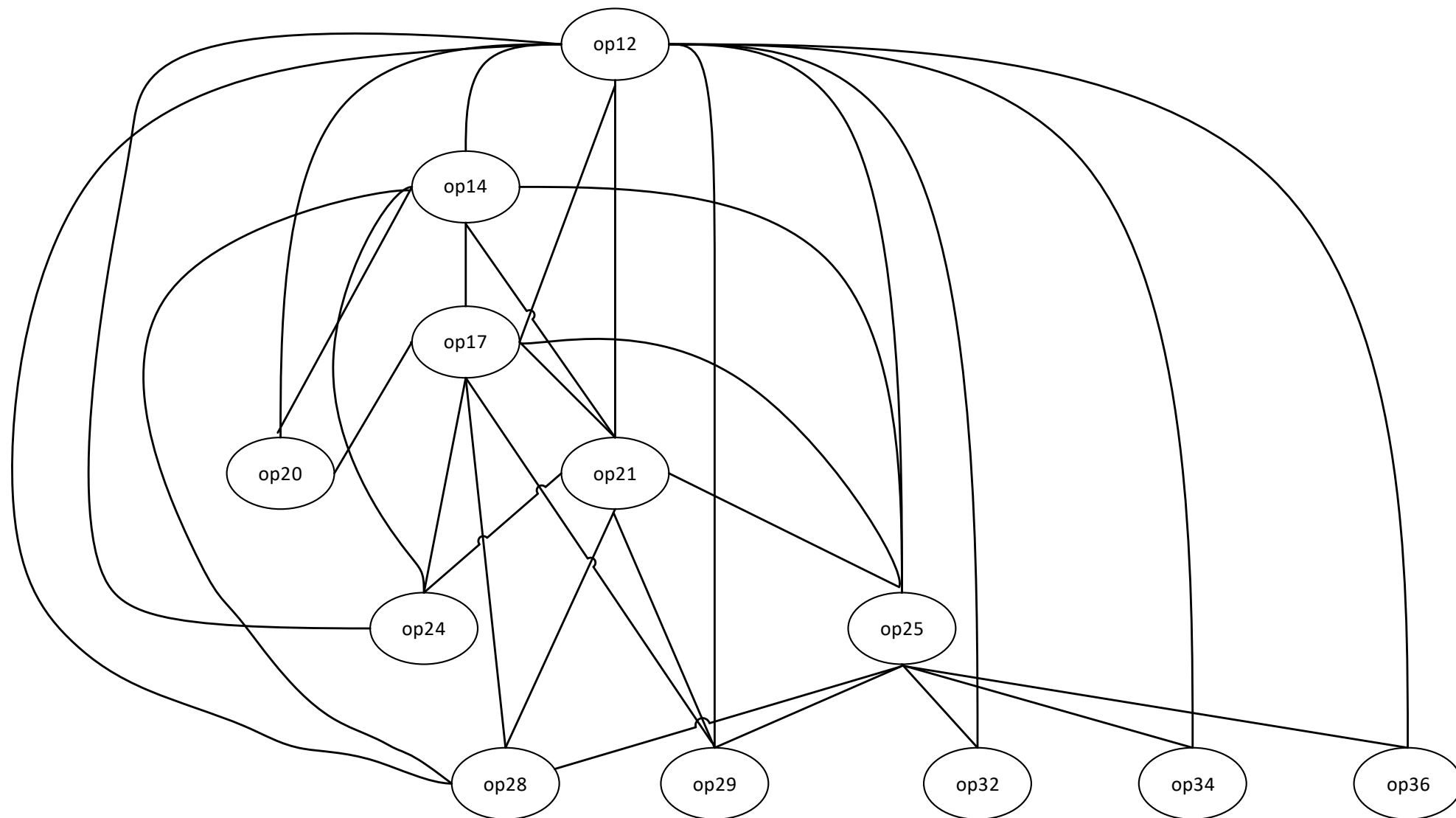
变量	活跃区间
%op12	[1,27)
%op14	[3,17)
%op17	[5,19)
%op20	[7,9)
%op21	[9,19)
%op24	[11,13)
%op25	[13,25)
%op28	[15,17)
%op29	[17,23)
%op32	[19,21)
%op34	[21,23)
%op36	[23,25)

```
target triple = "x86_64-pc-linux-gnu"
declare i32 @input()
declare void @output(i32)
declare void @outputFloat(float)
declare void @neg_idx_except()
define i32 @main() {
label_entry:
0 %op12 = add i32 1, 2
2 %op14 = mul i32 2, %op12
4 %op17 = add i32 %op12, %op14
6 %op20 = add i32 %op12, %op14
8 %op21 = add i32 %op20, 2
10 %op24 = sdiv i32 %op14, 2
12 %op25 = add i32 %op12, %op24
14 %op28 = sdiv i32 %op12, 2
16 %op29 = add i32 %op14, %op28
18 %op32 = add i32 %op21, %op17
20 %op34 = add i32 %op32, %op12
22 %op36 = add i32 %op34, %op29
24 %op38 = add i32 %op36, %op25
26 ret i32 %op12
}
```

寄存器图着色示例2



□建立干涉图



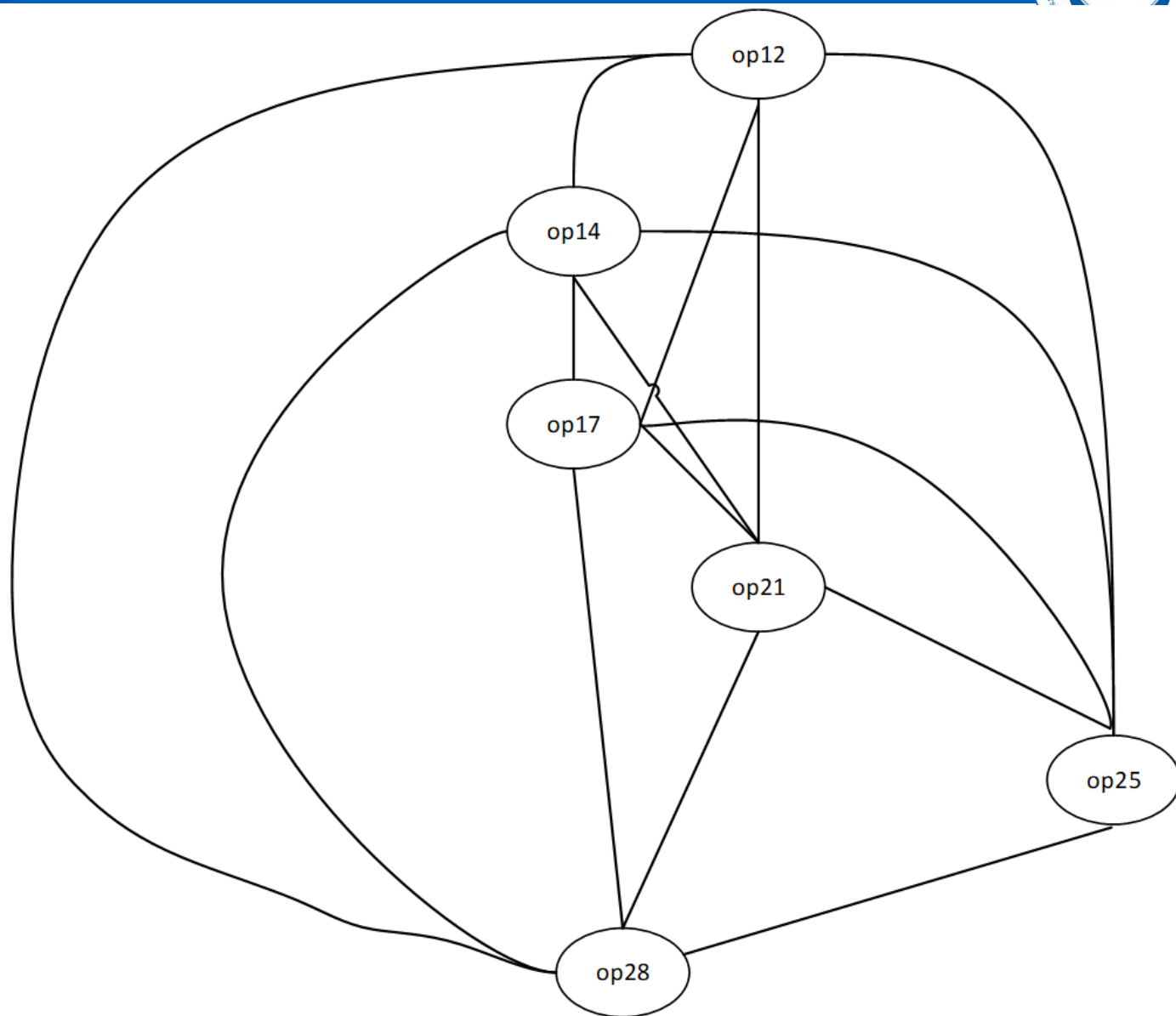
寄存器图着色示例2



化简图

■ 将度数小于5的节点

%op20, %op24, %op29,
%op32, %op34, %op36
依次从图中删除



寄存器图着色示例2

□ 溢出节点

■ 计算代价

■ 选择op25溢出

```
target triple = "x86_64-pc-linux-gnu"
declare i32 @input()
declare void @output(i32)
declare void @outputFloat(float)
declare void @neg_idx_except()
define i32 @main() {
label_entry:
    %op25 = alloca i32           ;记录变量在栈上位置
0  %op12 = add i32 1, 2
2  %op14 = mul i32 2, %op12
4  %op17 = add i32 %op12, %op14
6  %op20 = add i32 %op12, %op14
8  %op21 = add i32 %op20, 2
10 %op24 = sdiv i32 %op14, 2
12 %op39 = add i32 %op12, %op24
    store i32 %op39, i32* %op25 ;保存变量到栈上
14 %op28 = sdiv i32 %op12, 2
16 %op29 = add i32 %op14, %op28
18 %op32 = add i32 %op21, %op17
20 %op34 = add i32 %op32, %op12
22 %op36 = add i32 %op34, %op29
    %op40 = load i32, i32* %op25 ;从栈上读取数据
24 %op38 = add i32 %op36, %op40
26 ret i32 %op12
}
```

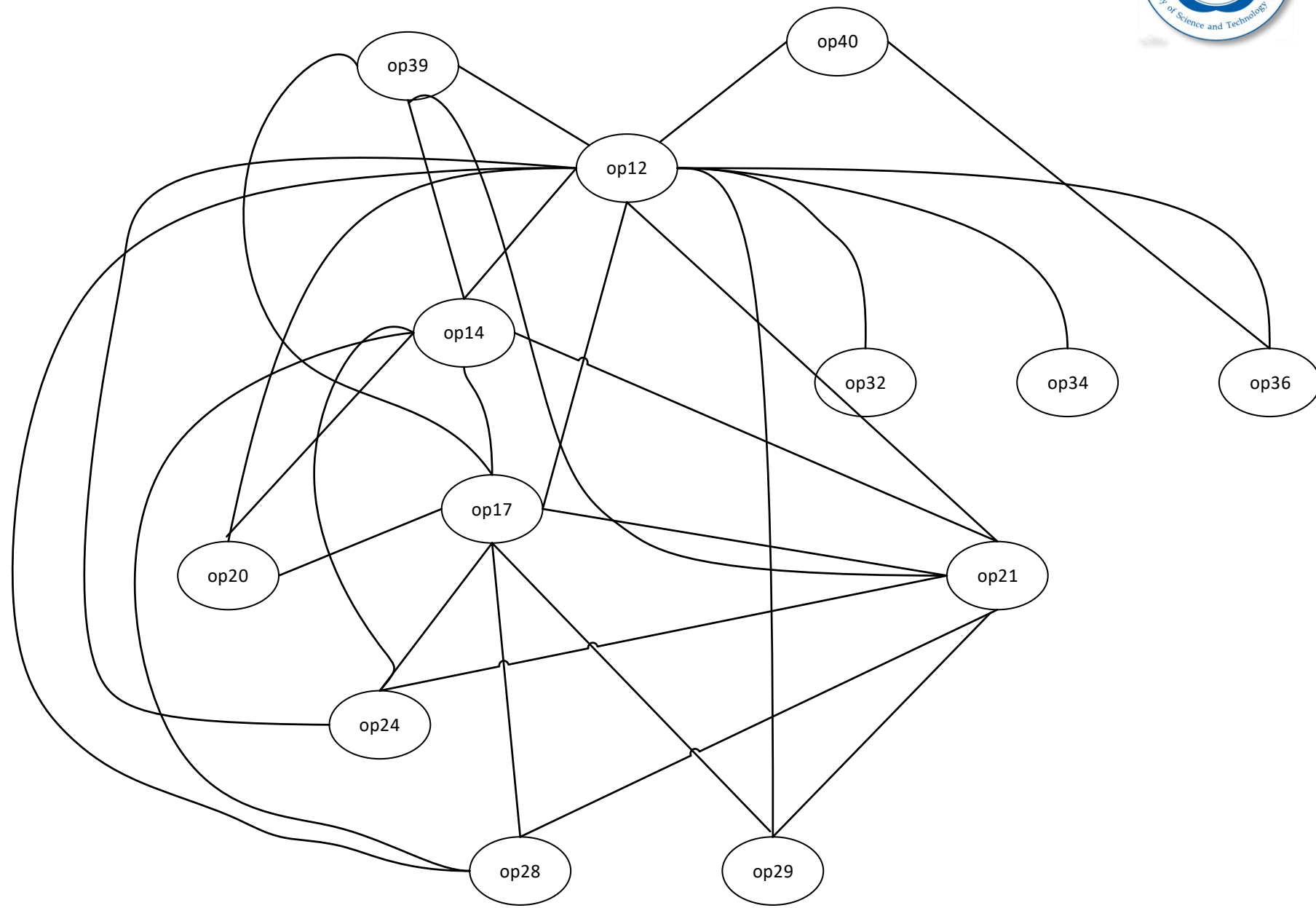


寄存器图着色示例2



□ 建立干涉图

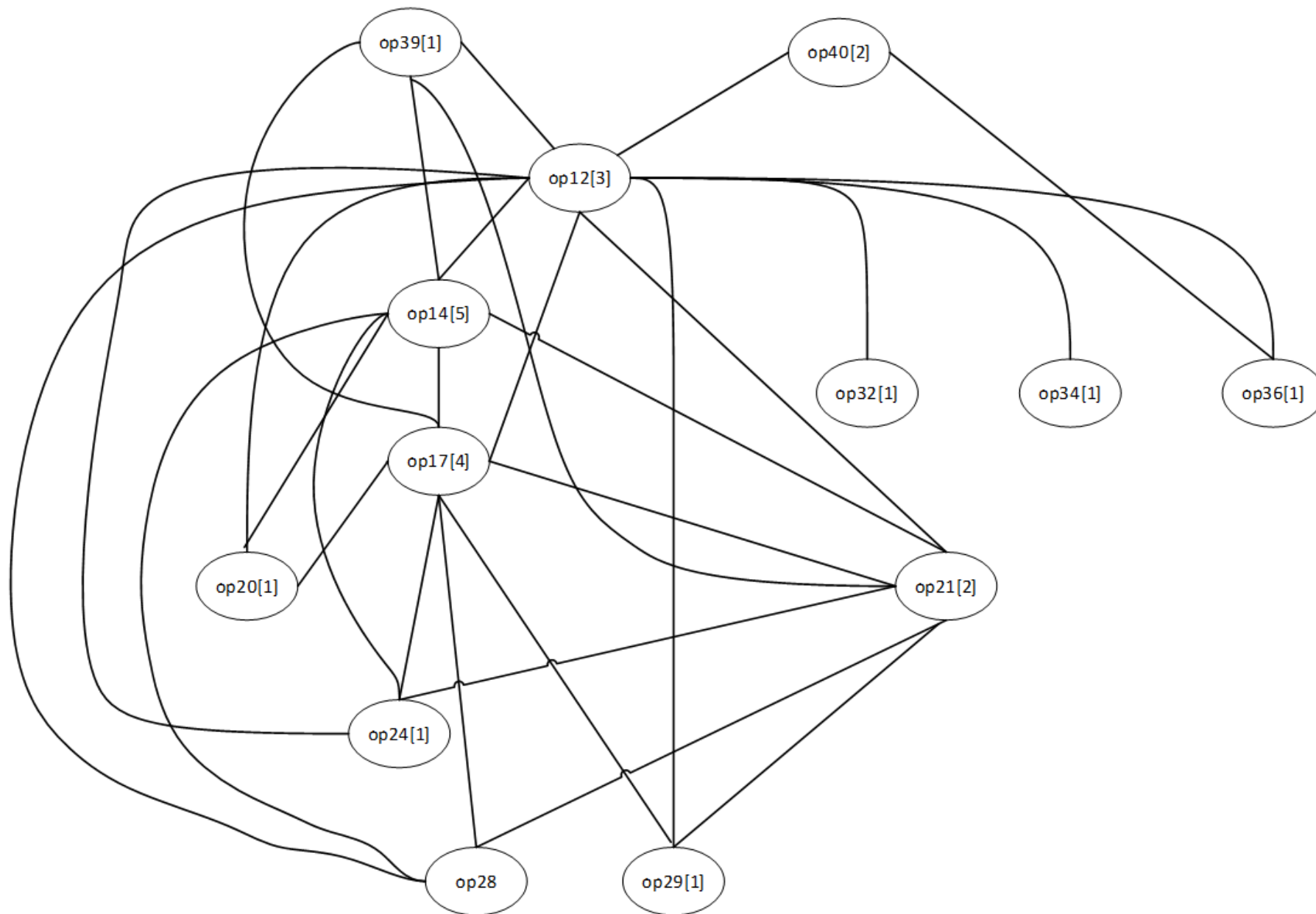
□ 化简图



寄存器图着色示例2



□染色





一起努力 打造国产基础系统软件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心
先进技术研究院、计算机科学与技术学院

2025年12月08日