编译原理笔记13：代码生成

一、流图

1. 基本块

基本块是满足下列条件的最大的连续三地址指令序列：

- 控制流只能从基本块的**第一个指令**进入该块。也就是说，没有跳转到基本块中间或末尾指令的转移指令
- 除了基本块的最后一个指令，控制流在离开基本块之前不会跳转或停机

基本块形成了流图的结点，而流图的边指明了哪些基本块可能紧跟一个基本块之后运行。

2. 基本块划分算法

算法 8.5 把三地址指令序列划分成为基本块。

输入：一个三地址指令序列。

输出：输入序列对应的一个基本块列表，其中每个指令恰好被分配给一个基本块。

方法：首先，我们确定中间代码序列中哪些指令是首指令(leader)，即某个基本块的第一个指令。跟在中间程序末端之后的指令的不包含在首指令集合中。选择首指令的规则如下：

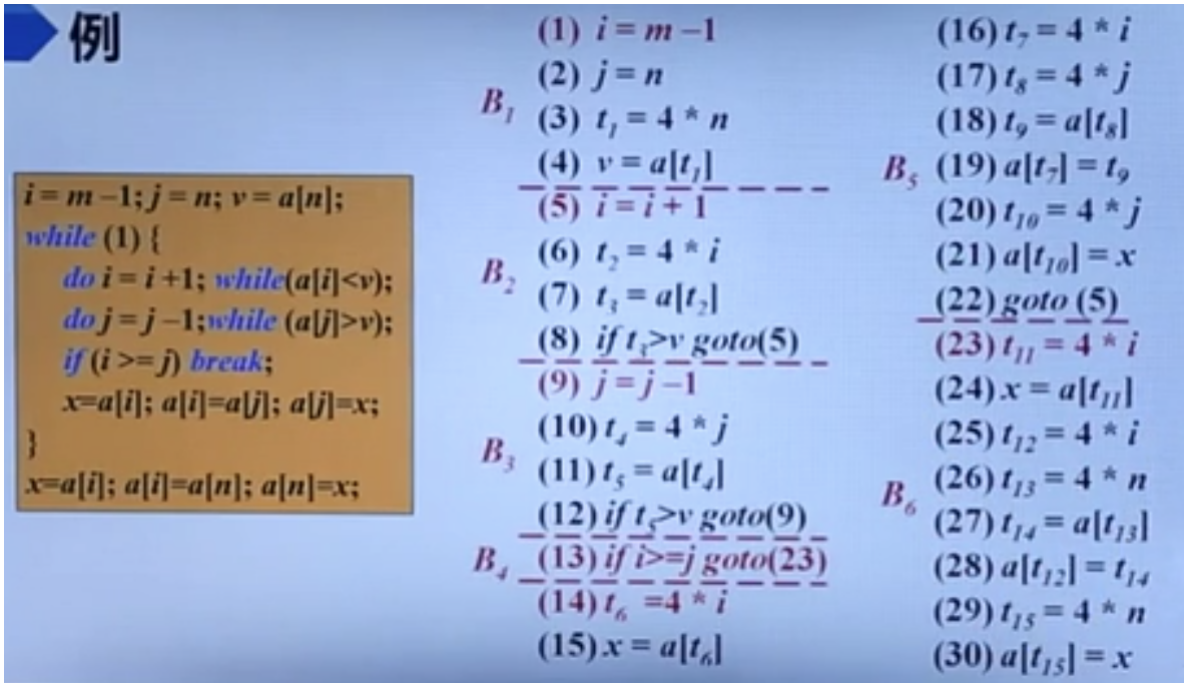
1）中间代码的第一个三地址指令是一个首指令。

2）任意一个条件或无条件转移指令的目标指令是一个首指令。

3）紧跟在一个条件或无条件转移指令之后的指令是一个首指令。

然后，每个首指令对应的基本块包括了从它自己开始，直到下一个首指令(不含)或者中间程序的结尾指令之间的所有指令。 □

举例说明：



划分的核心方法就是先找到所有的首指令，然后再进行划分。

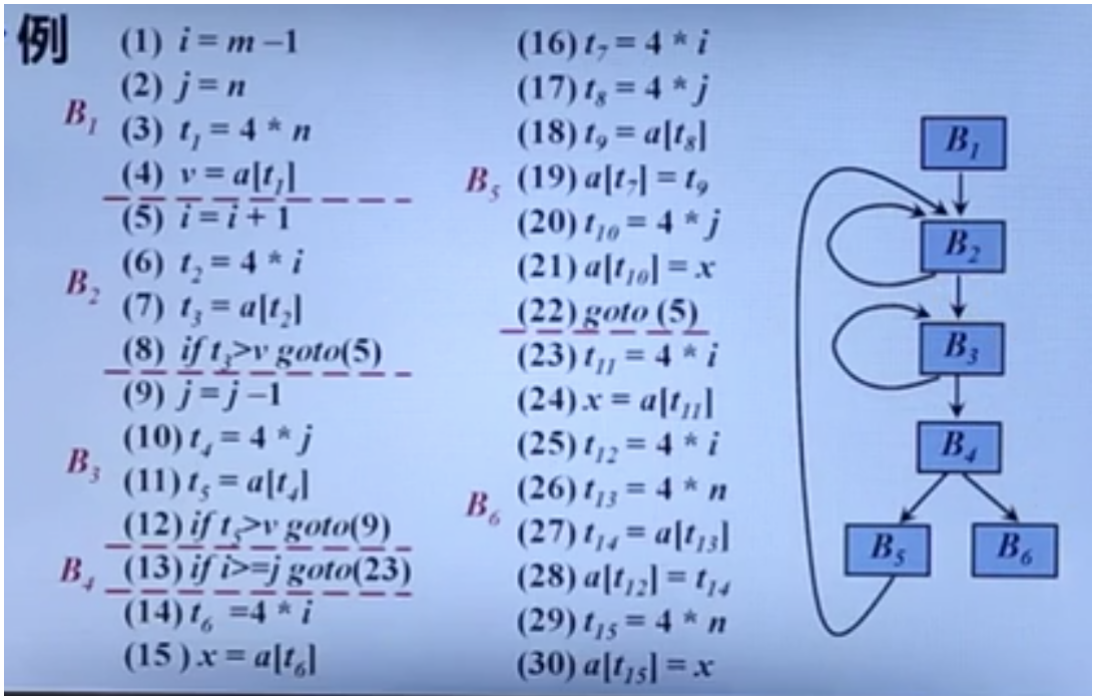
3. 流图

流图的结点使一些基本块。从基本块B到基本块C之间有一条边当且仅当基本块C的第一个指令可能紧跟在B的最后一条指令之后执行。

称B是C的前驱，C是B的后继。有两种方法可以确认这样的边：

1. 有一个**从B的结尾跳转到C的开头**的条件或无条件跳转语句
2. 按照原来三地址语句序列中的**顺序**，C紧跟在B之后，且B的结尾不存在**无条件跳转**语句

举例说明：



二、基本块的优化

1. 基本块的DAG表示

- 基本块中的每个语句 s 都对应着一个**内部结点N**，结点中的标号是 s 中的运算符
    - 同时还有一个**定值变量表**被关联到N，表示 s 是在此基本块内**最晚**对表中变量进行定值的语句
    - N的子节点是基本块中在 s 之前、最后一个对 s 所使用的某个运算分量进行定值的语句对应的结点。如果 s 的某个运算分量在基本块内没有在 s 之前被定值，则这个运算分量对应的子节点就是代表**该运算分量初始值的叶结点**（叶结点的定值变量表中的变量加上下脚标0）
    - 对于已经有的公共子表达式，不往DAG中添加公共结点，而是在已经存在的结点附加定值变量 x

2. 寻找局部公共子表达式

检查公共子表达式：当一个新的结点M被加入DAG中时，我们检查是否存在一个结点N，他和M具有相同的op和子节点，且子节点顺序相同。

例 8.10 下面的基本块的 DAG 见图 8-12。

```
a = b + c
b = a - d
c = b + c
d = a - d
```

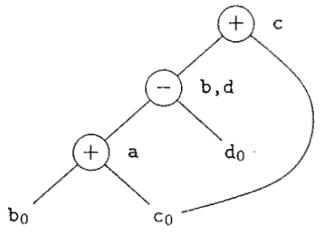


图 8-12 例 8.10 中的基本块的 DAG

当我们为第三个语句 c = b + c 构造结点的时候，我们知道 b + c 中 b 的使用指向图 8-12 中标号为 – 的结点。因为这个结点是 b 的最近的定值。因此，我们不会把语句 1 和语句 3 所计算的值混淆。

然而，对应于第四个语句 d = a – d 的结点的运算符是 –，且它的子结点是标记有变量 a 和 d_0 的结点。因为运算符和子结点都和语句 2 对应的结点相同，我们不需要创建这个结点，而是把 d 加到这个标记为 – 的结点的定值变量表中。 □

因为在图 8-12 的 DAG 中只有三个非叶子结点，看起来例 8.10 中的基本块可以替换为一个只有三个语句的基本块。实际上，假如 b 在这个基本块的出口点不活跃，我们不需要计算变量 b，可以使用 d 来存放图 8-12 中标号为 – 的结点所代表的值。这个基本块就变成了：

```
a = b + c
d = a - d
c = d + c
```

但是，如果 b 和 d 都在出口处活跃，我们就必须使用第四个语句把值从一个变量复制到另一个。[⊖]

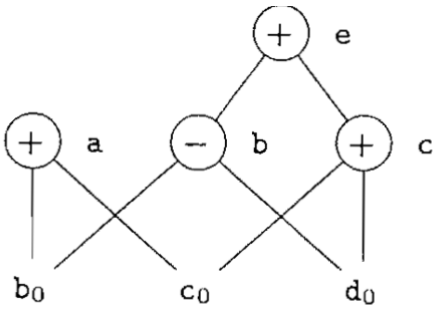
3. 消除死代码

死代码：即计算得到的值不会被使用的指令

活跃变量：值可能会在以后被使用的变量

在 DAG 上消除死代码的操作可以按照如下方式实现。我们从一个 DAG 上删除所有没有附加活跃变量的根结点（即没有父结点的结点）。重复应用这样的处理过程就可以从 DAG 中消除所有对应于死代码的结点。

举例说明：



在上图中，假设a和b是活跃变量，而e和c不是活跃变量。图中a和e是根结点，说明他们在基本块中没有被引用，而e不是活跃变量，因此在以后也不会被使用，因此把它删去，接着递归的，c 也满足上述条件可以删去。

4. 数组元素赋值指令的表示

在上图中，如果把a[i]作为一个公共的子表达式是不合理的，因为j=i是可能出现的。在DAG中，表示数组访问的正确方法如下：

1）从一个数组取值并赋给其他变量的运算（比如 x = a[i]）用一个新创建的运算符为 =[] 的结点表示。这个结点的左右子结点分别代表数组初始值（本例中是 a_0）和下标 i。变量 x 是这个结点的标号之一。

2）对数组的赋值（比如 a[j] = y）用一个新创建的运算符为[] = 的结点来表示。这个结点的三个子结点分别表示 a_0、j 和 y。没有变量用这个结点标号。不同之处在于此结点的创建杀死了所有当前已经建立的，其值依赖于 a_0 的结点。一个被杀死的结点不可能再获得任何标号。也就是说，它不可能成为一个公共子表达式。

例 8.13 基本块

```
x = a[i]
a[j] = y
z = a[i]
```

的 DAG 见图 8-14。对应于 x 的结点 *N* 首先被创建，但是当标号为[] = 的结点被创建时，N 就被杀死了。因此当 z 的结点被建立时，它不会被认为和 N 等同，而是必须创建一个具有同样的运算分量 a_0 和 i_0 的新结点。 □

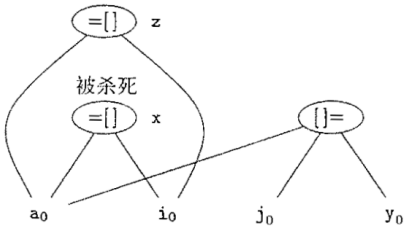
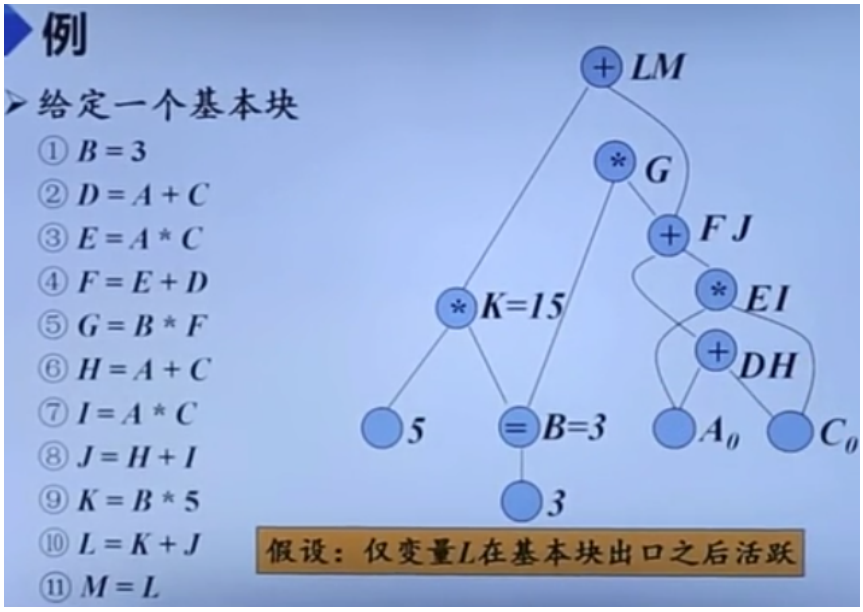


图 8-14 一个数组赋值序列的 DAG

举例说明：

对于下图左侧的基本块，我们可以构造出它的DAG如右图：



接着我们需要根据DAG重新生成基本块从而完成局部代码优化：

1. 首先删除没有活跃变量的根结点，如G
2. 对于局部公共子表达式，我们只需要生成一条三地址指令即可，因此我们可以删除M、J、I、H（我们倾向于将结果赋值给活跃变量）
3. 根据新的DAG图，重新生成基本块代码

当有常量的时候，我们可以直接用常量代替变量。因此第一条三地址表达式也不需要。在第九条表达式的时候，K也可以直接用15来代替，可以直接删除。