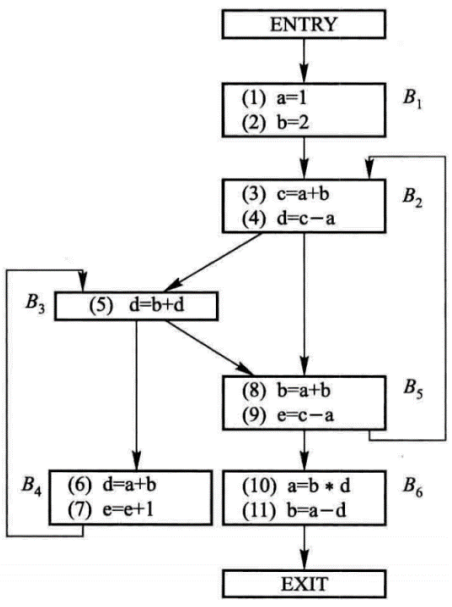
编译原理　第十五周作业　12 月 24 日　周四

PB18151866 龚小航

9.3 对给出的流图，计算：

(b) 为可用表达式分析，计算每个块的 e_gen, e_kill, IN, OUT 集合。



解：　参考课本 286 页的定义以及 287 页的算法：

（先算 IN 再算 OUT）

$$\mathrm{OUT}[ENTRY] = \emptyset$$

$$\mathrm{IN}[B] = \cap_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

$$\mathrm{OUT}[B] = e\_gen[B] \cup (\mathrm{IN}[B] - e\_kill[B])$$

列表如下，令初始时除了 $ENTRY$ 外其余块 $\mathrm{OUT} = \mathrm{U}$，从第一轮开始：

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

块	e_gen	e_kill	IN^1	OUT^1	IN^2	OUT^2
B_1	{1,2}	$a+b,\ c-a,\ b+d,\ b*d,\ a-d$	$\emptyset$	1, 2	$\emptyset$	1, 2
B_2	$\{a+b, c-a\}$	$b+d,\ b*d,\ a-d$	1, 2	1, 2, $a+b$, $c-a$	1, 2	1, 2, $a+b$, $c-a$
B_3	$\emptyset$	$b+d,\ b*d,\ a-d$	1, 2, $a+b$, $c-a$	1, 2, $a+b$, $c-a$	1, 2, $a+b$, $c-a$	1, 2, $a+b$, $c-a$
B_4	$\{a+b\}$	$b+d,\ e+1,\ b*d$, $a-d$	1, 2, $a+b$, $c-a$	1, 2, $a+b$, $c-a$	1, 2, $a+b$, $c-a$	1, 2, $a+b$, $c-a$
B_5	$\{c-a\}$	$a+b,\ b+d,\ e+1$, $b*d$	1, 2, $a+b$, $c-a$	1, 2, $c-a$	1, 2, $a+b$, $c-a$	1, 2, $c-a$
B_6	$\{a-d\}$	$a+b,\ c-a,\ b+d$, $b*d$	1, 2, $c-a$	1, 2, $a-d$	1, 2, $c-a$	1, 2, $a-d$

至此迭代算法已经在一轮中没有改变，得到了最终的结果，如上表所示。

9.15 对于本题中的流图，回答下列问题：

(a) 计算支配关系。

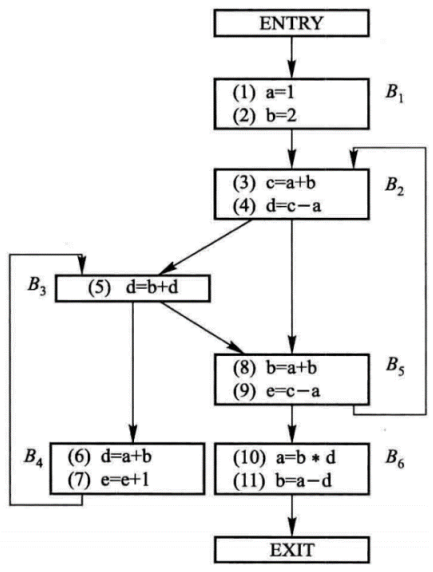
(b) 找出一种深度优先排序。

(c) 对上一小问的结果，标明前进边、后撤边和交叉边。

(d) 该流图是否可归约。

(e) 计算该流图的深度。

(f) 找出该流图的自然循环。

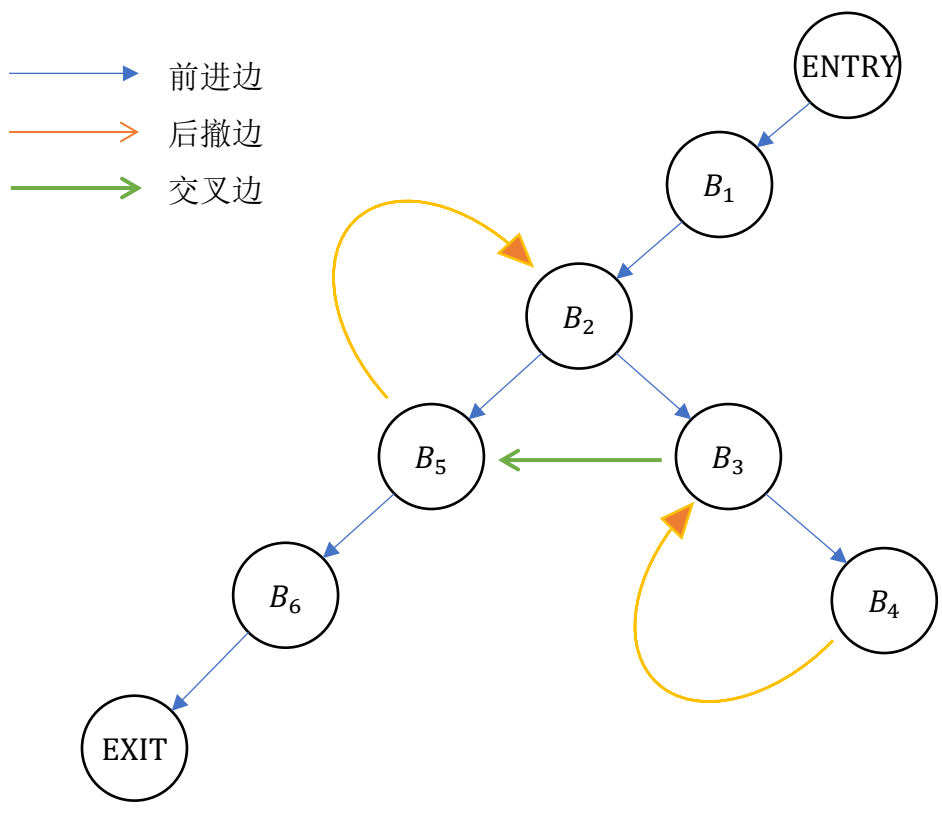


解：　对每一问分析如下

(a) 支配关系如下表所示：记 $U = \{\mathrm{ENTRY} \cup B_1 \cup B_2 \cup B_3 \cup B_4 \cup B_5 \cup B_6 \cup \mathrm{EXIT}\}$

块	ENTRY	B_1	B_2	B_3	B_4	B_5	B_6	EXIT
支配对象	U	U − EXIT	U − EXIT − B_1	B_3, B_4	B_4	B_5, B_6, EXIT	B_6, EXIT	EXIT

(b, c) 作图如下：深度优先系列为 $\mathrm{ENTRY} \to B_1 \to B_2 \to B_5 \to B_6 \to \mathrm{EXIT} \to B_3 \to B_4$



($\boldsymbol{d}$) 按照可归约性的定义【课本 313 页】，在任何深度的深度优先生成树上后撤边都是回边，因此这个流图是可归约的。

($\boldsymbol{e}$) $B_4 \to B_3 \to B_5 \to B_2$ 包含两条后撤边，因此该流图深度为 2

($\boldsymbol{f}$) 显然共两个自然循环：

后撤边$B_5 \to B_2$确定的自然循环：$\{B_2, B_3, B_4, B_5\}$

后撤边$B_4 \to B_3$确定的自然循环：$\{B_3, B_4\}$

9.22 请利用代码优化的思想（代码外提和强度削弱等），改写下面 C 语言程序中的循环，得到优化后的 C 语言程序。

```
main(){
    int i, j;
    int r[20][10];

    for(i=0; i<20; i++){
        for(j=0; j<10; j++){
            r[i][j] = 10 * i * j;
        }
    }
}
```

解：优化后代码如下：

```
main(){
    int i, j;
    int r[20][10];

    for(i=0; i<20; i++){
       muli = 10 * i;
       temp = 0;
       for(j=0; j<10; j++){
            r[i][j] = temp;
            temp = temp + muli;
       }
    }
}
```

9.24 某优化编译器对下面程序的局部变量 i 和 j 不分配空间，为什么？

```
main(){
    long i,j;
    i=5;
    j=i*2;
    printf("%d\n",i+j);
}
```

解：一般的编译器在做独立于机器的代码优化时，会进行常量传播、复写传播、死代码删除工作。本题给出的代码中 i,j 都可以进行复写传播、常量传播，完成这些工作后发现程序 2，3，4 行都是死代码。因此这一段程序优化后结果如下：

```
main(){
    printf("%d\n",5+5*2);
}
```

8.3 为下列 C 语句产生 8.2 节目标机器的代码，假定所有的变量都是静态的，并假定有 3

```
(a) x = a[i] + 1
(b) a[i] = b[c[i]]
(c) a[i] = a[i] + b[j]
```

个寄存器可用于保存计算结果。

解：记可用的三个寄存器为 R_1, R_2, R_3，以下为汇编表示的机器代码：

```
(a) x= a[i] + 1
    LODE  R1,i
    MUL   R1,R1,8
    LODE  R2,a[R1]
    ADD   R2,R2,1
    STORE x,R2
(b) a[i] = b[c[i]]
    LODE  R1,i
    MUL   R1,R1,8
    LODE  R2,c[R1]
    MUL   R2,R2,8
    LODE  R3,b[R2]
    STORE a[R1],R3
(c) a[i] = a[i] + b[j]
    LODE  R1,i
    MUL   R1,R1,8
    LODE  R2,j
    MUL   R2,R2,8
    LODE  R1,a[R1]
    LODE  R2,a[R2]
    ADD   R3,R1,R2
    LODE  R1,i
    MUL   R1,R1,8
    STORE a[R1],R3
```

【lecture14 Page50】

❑参考slides35-48页的例子，(a)为下面的三地址码序列生成对应的目标代码；(b)假设只有两个寄存器R0和R1，基本块出口处只有f活跃，请分析每一条三地址语句翻译后，寄存器描述符和变量的地址描述符的变化情况; (3) 计算目标代码的总代价，并请尽可能地优化以减少代价。

```
t1 = a + b
t2 = t1 – c
t3 = d + e
t3 = t2 * t3
t4 = t1 + t3
t5 = t3 – e
f = t4 * t5
```

解：生成的目标代码如下：

```
1   //t1=a+b
2   LODE  R0,a
3   LODE  R1,b
4   ADD   R0,R0,R1/*R0:t1 R1:b*/
5   //t2=t1-c
6   LODE  R1,c
7   SUB   R1,R0,R1  /*R0:t1 R1:t2*/
8   //t3=d+e
9   STORE t1,R0 /*存储 t1,第五行还需要 t1*/
10  LODE  R0,d
11  STORE t2,R1 /*存储 t2，第四行还需要 t2*/
12  LODE  R1,e
13  ADD   R0,R0,R1 /*R0:t3  R1:e*/
14  //t3 = t2*t3 = t2*(d+e) = t2*R0
15  LODE  R1,t2
16  MUL   R0,R0,R1  /*R0:t3 R1:t2*/
17     /*t3 在此后仅引用一次，无需 STORE，下一条语句直接从寄存器调用*/
18  //t4 = t1+t3
19  LODE  R1,t1
20  ADD   R1,R0,R1/*R0:t3  R1:t4*/
21  //t5 = t3-e
22  STORE t4,R1  /*存储 t4，第七行还要用到*/
23  LODE  R1,e
24  SUB   R1,R0,R1
25     /*t3 为临时变量，后续没有使用，无需 STORE*/
26  //f = t4*t5
27  LODE  R0,t4
28  MUL   R0,R0,R1
29  STORE f,R0
30     /*仅 f 在出口处活跃*/
```

按行号写出每一条三地址语句**调用后**寄存器与内存中的数据变化：

行号	R_0	R_1	a	b	c	d	e	f	e	t_2	t_3	t_4	t_5
1	t_1	b	a	b, R_1	c	d	e		R_0				
2	t_1	t_2	a	b	c	d	e		R_0	R_1			
3	t_3	e	a	b	c	d	e, R_1		t_1	t_2	R_0		
4	t_3	t_2	a	b	c	d	e		t_1	t_2	R_0		
5	t_3	t_4	a	b	c	d	e		t_1	t_2	R_0	R_1	
6	t_3	t_5	a	b	c	d	e		t_1	t_2	R_0	t_4	R_1
7	f	t_5	a	b	c	d	e	f	t_1	t_2		t_4	R_1

上述选择寄存器的策略已经是最优的，目标代码的总代价为 33