中国科学技术大学
University of Science and Technology of China

USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》习题课

任课老师：李　诚
主讲助教：邵新洋

25/12/2019

题号

- 第十次作业
  - 课后习题5.5和5.6
- 第十二次作业
  - 课后习题9.1和9.2
- 第十三次作业
  - 课后习题9.3 c, 9.15
- 第十四次作业
  - 课后6.6, 6.10, 6.13
  - lecture18_codegen.pdf中的第53页

习题5.5

- 假如有下列C的声明：

```
typedef struct {
        int a, b;
} CELL, * PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

习题5.5

- 假如有下列C的声明：

```
typedef struct {
        int a, b;
} CELL, * PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

foo: array(100, record((a × integer) × (b × integer)))

bar: (integer × record((a × integer) × (b × integer))) → pointer(record((a × integer) × (b × integer)))

习题5.6

- 下列文法定义字面常量表的表。符号的解释和图5.2文法的那些相同，增加了类型list，它表示类型 T 的元素表。

$$P \rightarrow D;E$$

$$D \rightarrow D;D \mid \mathbf{id} : T$$

$$T \rightarrow \mathbf{list\ of}\ T \mid \mathbf{char} \mid \mathbf{integer}$$

$$E \rightarrow (L) \mid \mathbf{literal} \mid \mathbf{num} \mid \mathbf{id}$$

$$L \rightarrow E,L \mid E$$

习题5.6

$P \rightarrow D;E$

$D \rightarrow D;D$

$D \rightarrow \textbf{id} : T \{ addtype(\text{id}.entry, T.type); \}$

$T \rightarrow \textbf{char} \{ T.type = char; \}$

$T \rightarrow \textbf{integer} \{ T.type = integer; \}$

$T \rightarrow \textbf{list of } T_1 \{ T.type = list(T_1.type); \}$

$E \rightarrow \textbf{literal} \{ E.type = char; \}$

$E \rightarrow \textbf{num} \{ E.type = integer; \}$

$E \rightarrow \textbf{id} \{ E.type = lookup(\textbf{id}.entry); \}$

$E \rightarrow (L) \{ E.type = list(L.type); \}$

$L \rightarrow E, L_1 \{ \textbf{if } (E.type == L_1.type)\ L.type = E.type; \textbf{ else } L.type = type\_error; \}$

$L \rightarrow E \{ L.type = E.type; \}$

习题9.1

❑对于图9.32流图：

(a) 识别该流图的循环。

(b) 块B_1中的语句(1)和(2)都是复写语句，并且它们给a和b赋的都是常量。可以对a和b的哪些引用实施复写传播并将这些引用替换成对常量的引用?

(c) 识别每个循环的全局公共子表达式。

(d) 识别每个循环的归纳变量，不要忘记把(b)的复写传播引入的常量考虑进去。

(e) 识别每个循环的循环不变计算。

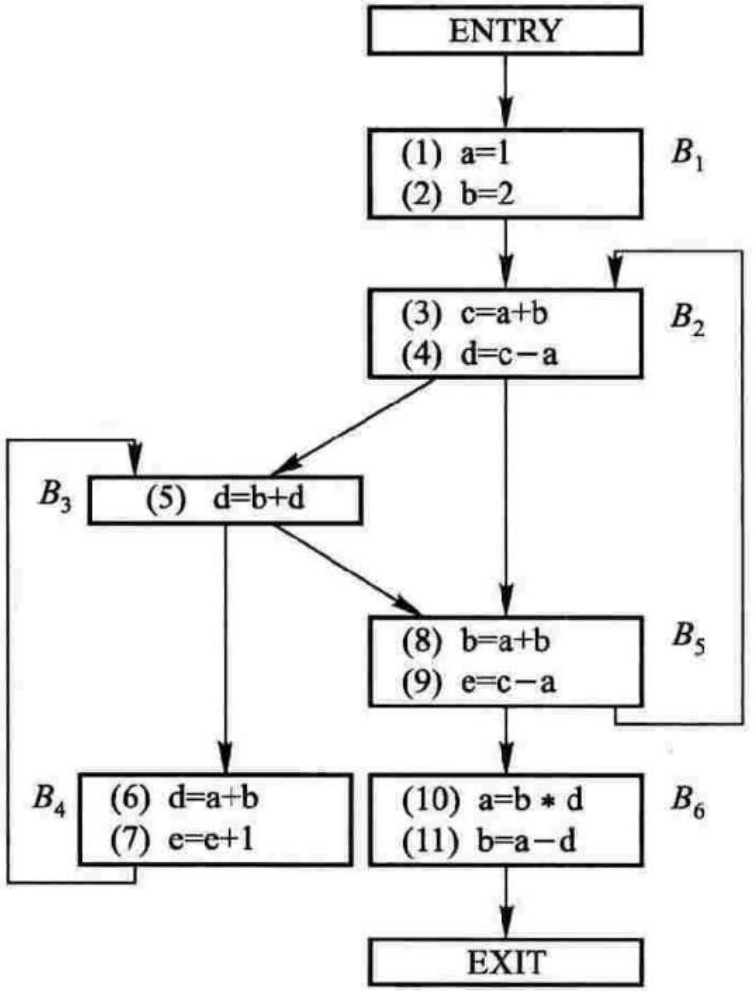


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(a) 识别该流图的循环。

B_2, B_3, B_5

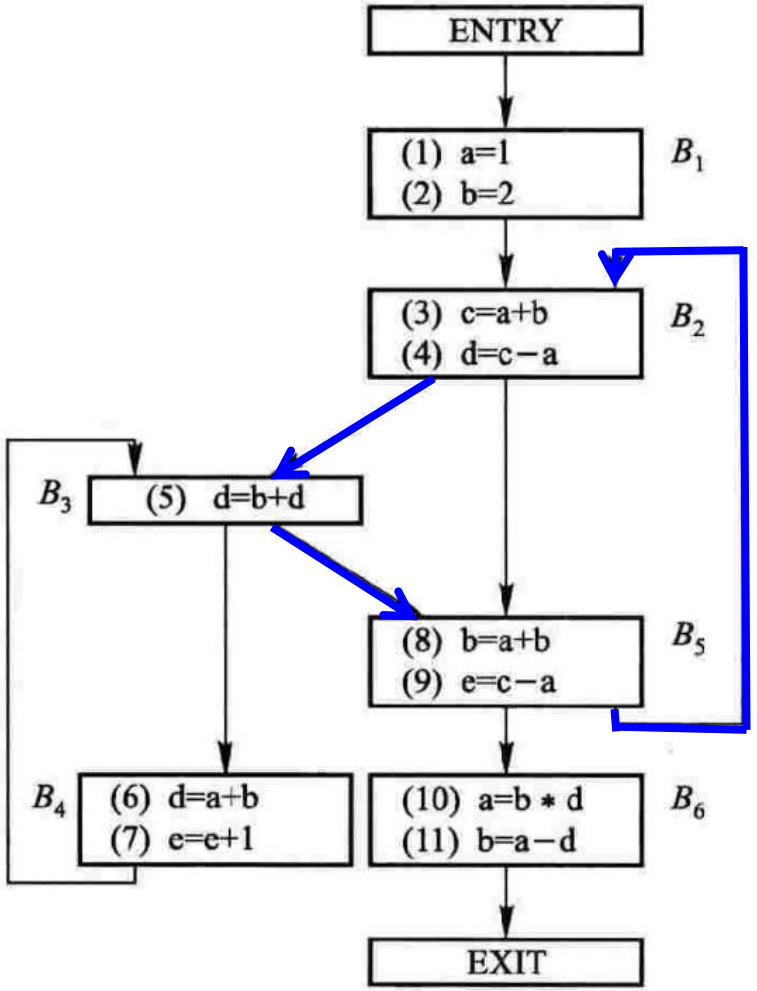


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(a) 识别该流图的循环。

B_2, B_3, B_5

B_2, B_5

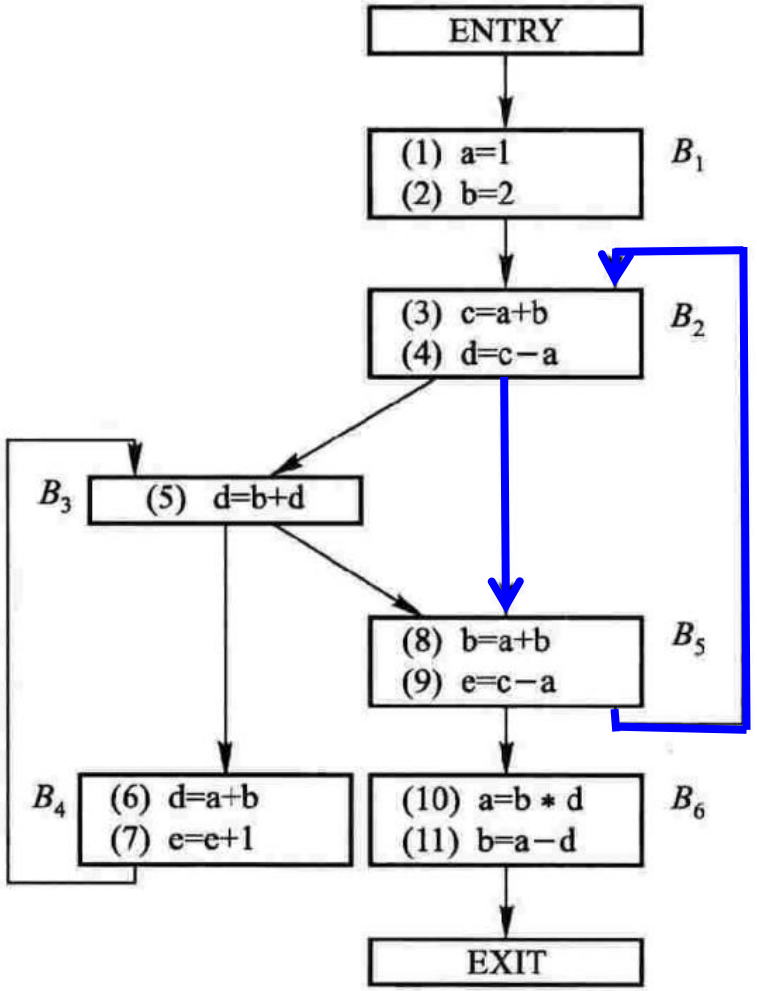


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(a) 识别该流图的循环。

B_2, B_3, B_5

B_2, B_5

B_3, B_4

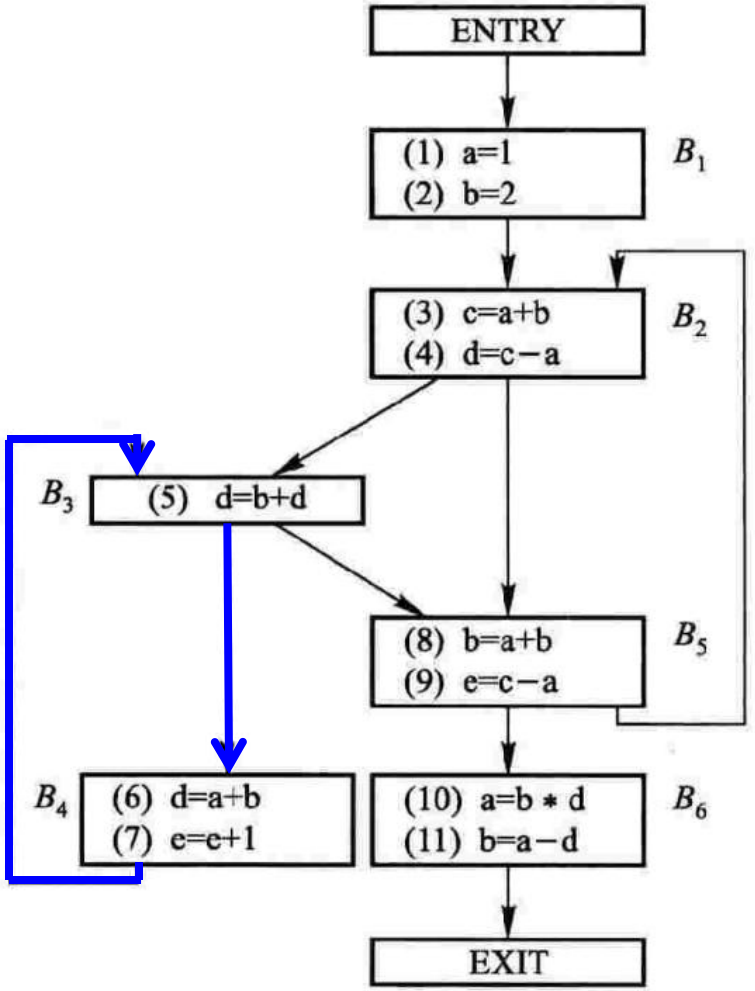


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(a) 识别该流图的循环。

B_2, B_3, B_5

B_2, B_5

B_3, B_4

B_2, B_3, B_4, B_5

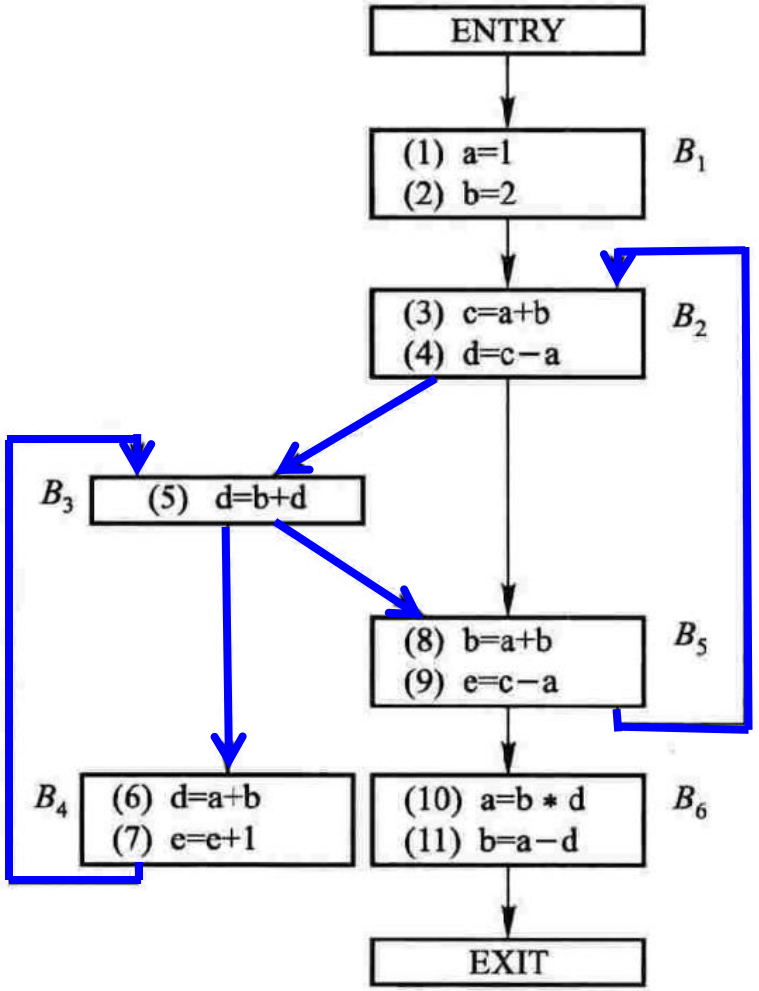


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(b) 块B_1中的语句(1)和(2)都是复写语句，并且它们给a和b赋的都是常量。可以对a和b的哪些引用实施复写传播并将这些引用替换成对常量的引用?

a的值：(3) (4) (6) (8) 可替换为 1

b的值：无（(8) 中存在修改）

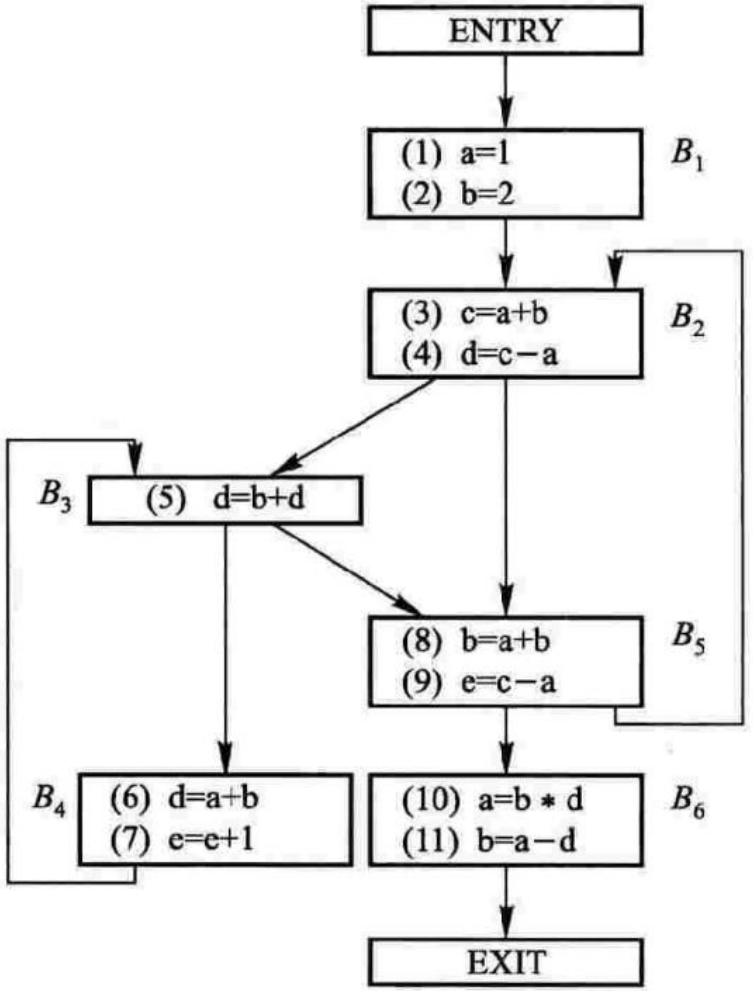


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(c) 识别每个循环的全局公共子表达式。

(3) (6) (8) 中的a+b和
(4) (9) 中的c−a

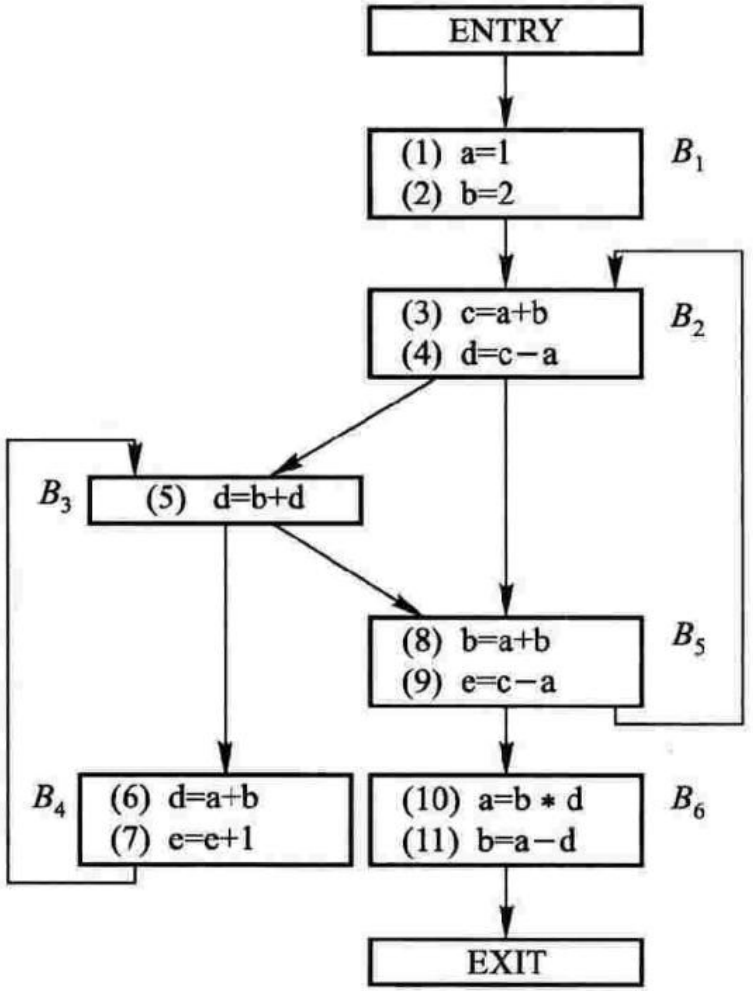


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(d) 识别每个循环的归纳变量，不要忘记把(b)的复写传播引入的常量考虑进去。

$\{B_2, B_5\}$： b, c, d

$\{B_2, B_3, B_5\}$： b, c

$\{B_3, B_4\}$： e

$\{B_2, B_3, B_4, B_5\}$： b, c

注：c在循环中自增2

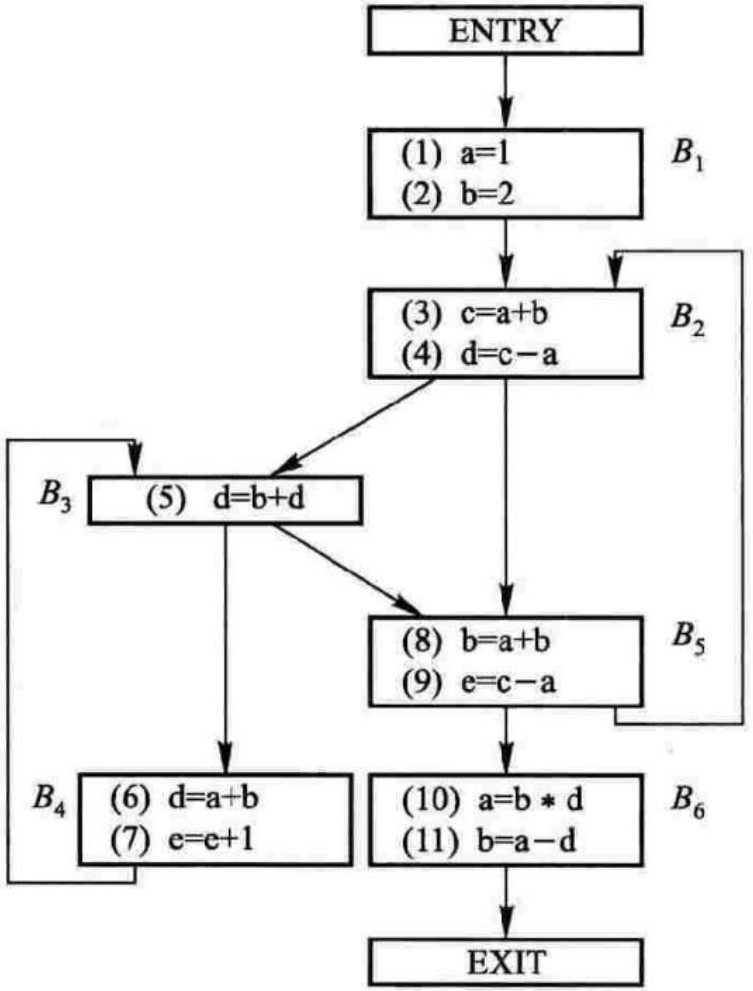


图 9.32 一个流图

习题9.1

❑对于图9.32流图：

(e) 识别每个循环的循环不变计算。

$\{B_3, B_4\}$: d=a+b 其他没有

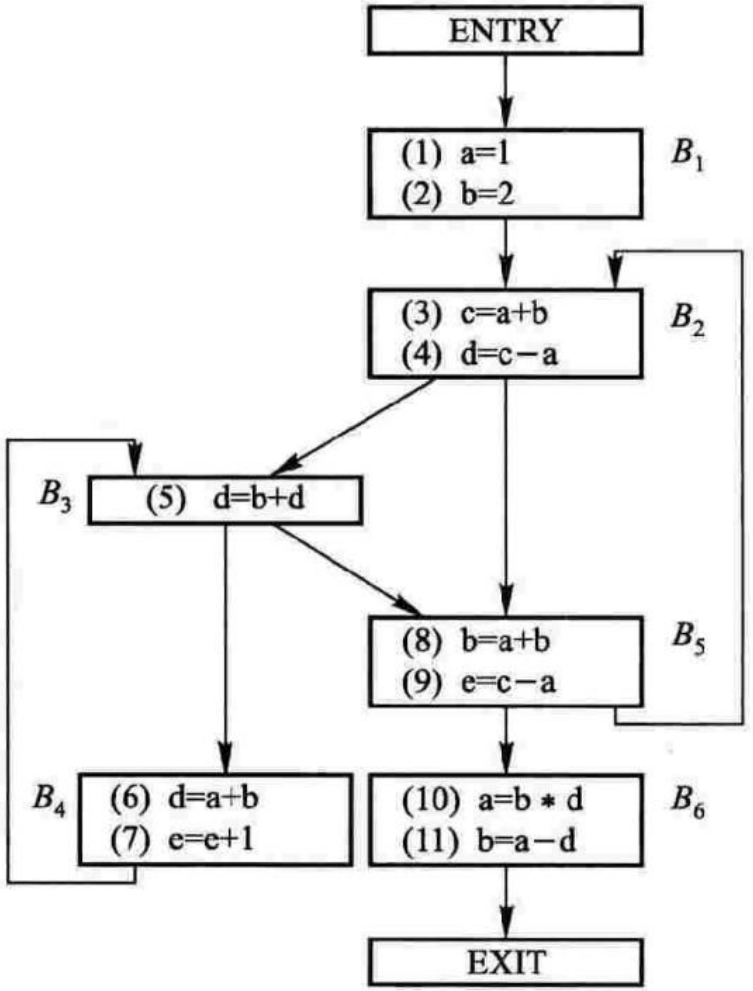


图 9.32 一个流图

习题9.2

- 为图9.33计算向量A和B点积的中间代码完成下列优化：删除公共子表达式、归纳变量上的强度削弱和尽量删除归纳变量。

（答案不唯一）

```
    dp = 0
    i = 0
L:  t1 = i * 8
    t2 = A[t1]
    t3 = i * 8
    t4 = B[t3]
    t5 = t2 * t4
    dp = dp+t5
    i = i+1
    if i<n goto L
```

图 9.33　计算点积的中间代码

习题9.2

❑删除公共子表达式：

第三行和第五行中的i*8为局部公共子表达式。

```
        dp = 0
        i = 0
L:      t1 = i * 8
        t2 = A[t1]
        t3 = i * 8
        t4 = B[t3]
        t5 = t2 * t4
        dp = dp + t5
        i = i + 1
        if i < n goto L
```

习题9.2

❑删除公共子表达式：

第三行和第五行中的i*8为局部公共子表达式。

```
        dp = 0
        i = 0
L:      t1 = i * 8
        t2 = A[t1]
        t4 = B[t1]
        t5 = t2 * t4
        dp = dp + t5
        i = i + 1
        if i < n goto L
```

习题9.2

❑归纳变量上的强度削弱：

变量t1总是每次增加8，因此成为归纳变量。我们将它的乘法削弱为加法。

```
        dp = 0
        i = 0
L:      t1 = i * 8
        t2 = A[t1]
        t4 = B[t1]
        t5 = t2 * t4
        dp = dp + t5
        i = i + 1
        if i < n goto L
```

习题9.2

❑归纳变量上的强度削弱：

变量t1总是每次增加8，因此成为归纳变量。我们将它的乘法削弱为加法。

```
        dp = 0
        i = 0
        t1 = 0
L:      t2 = A[t1]
        t4 = B[t1]
        t5 = t2 * t4
        dp = dp + t5
        t1 = t1 + 8
        i = i + 1
        if i < n goto L
```

习题9.2

❑尽量删除归纳变量：

变量t1和i的值的变化保持步调一致，因此可以只保留一个。

考虑到t2和t4的赋值语句中用到了t1，而循环退出条件中虽然用到i，但是n是个常数。因此将这里我们保留t1，并将退出条件中的i和n变为t1和8*n。

```
    dp = 0
    i = 0
    t1 = 0
L:  t2 = A[t1]
    t4 = B[t1]
    t5 = t2 * t4
    dp = dp + t5
    t1 = t1 + 8
    i = i + 1
    if i < n goto L
```

习题9.2

❑尽量删除归纳变量：

变量t1和i的值的变化保持步调一致，因此可以只保留一个。

考虑到t2和t4的赋值语句中用到了t1，而循环退出条件中虽然用到i，但是n是个常数。因此将这里我们保留t1，并将退出条件中的i和n变为t1和8*n。

```
        dp = 0
        i = 0
        t1 = 0
L:      t2 = A[t1]
        t4 = B[t1]
        t5 = t2 * t4
        dp = dp + t5
        t1 = t1 + 8
        i = i + 1
        if t1 < 8*n goto L
```

循环不变量

习题9.2

❑尽量删除归纳变量：

变量t1和i的值的变化保持步调一致，因此可以只保留一个。

考虑到t2和t4的赋值语句中用到了t1，而循环退出条件中虽然用到i，但是n是个常数。因此将这里我们保留t1，并将退出条件中的i和n变为t1和8*n。

```
        dp = 0
        t1 = 0
        t3 = 8 * n
L:      t2 = A[t1]
        t4 = B[t1]
        t5 = t2 * t4
        dp = dp + t5
        t1 = t1 + 8
        if t1 < t3 goto L
```

习题9.3 (c)

- 对图9.32的流图，计算：

(c) 为活跃变量分析，计算每个块的def、use、IN和OUT集合。

（过程见Lecture15活跃变量相关部分）

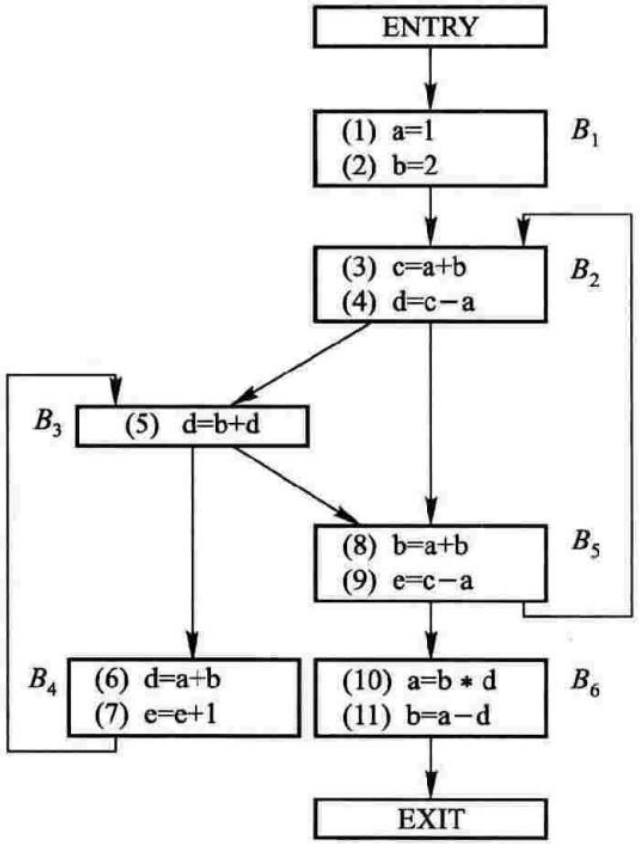


图 9.32 一个流图

习题9.3 (c)

- $use[B_1] = \{\}, def[B_1] = \{a, b\}$
- $use[B_2] = \{a, b\}, def[B_2] = \{c, d\}$
- $use[B_3] = \{b, d\}, def[B_3] = \{\}$
- $use[B_4] = \{a, b, e\}, def[B_4] = \{d\}$
- $use[B_5] = \{a, b, c\}, def[B_5] = \{e\}$
- $use[B_6] = \{b, d\}, def[B_6] = \{a\}$

	IN[B]	OUT[B]
B_1	$\{e\}$	$\{a, b, e\}$
B_2	$\{a, b, e\}$	$\{a, b, c, d, e\}$
B_3	$\{a, b, c, d, e\}$	$\{a, b, c, d, e\}$
B_4	$\{a, b, c, e\}$	$\{a, b, c, d, e\}$
B_5	$\{a, b, c, d\}$	$\{a, b, d, e\}$
B_6	$\{b, d\}$	$\{\}$

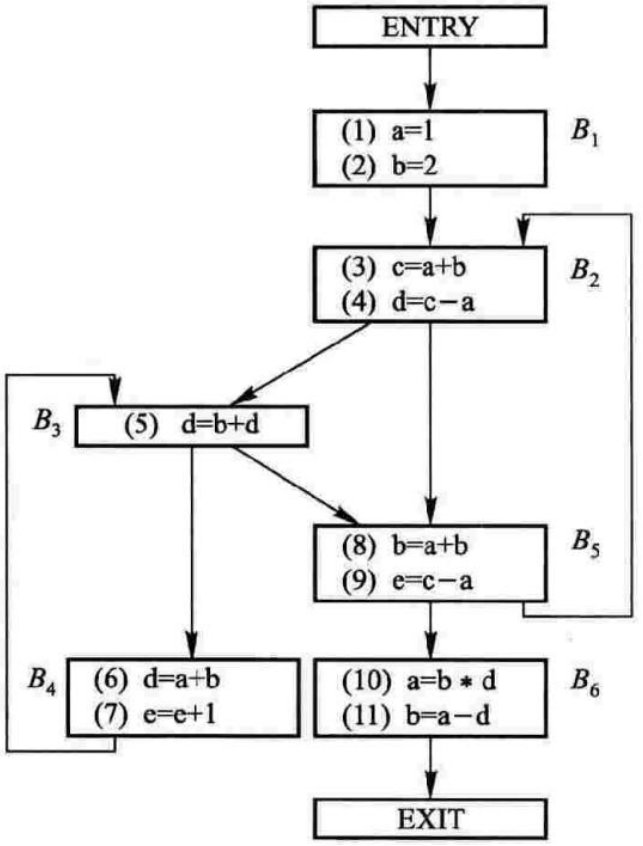


图 9.32 一个流图

习题9.15

❑对图9.32的流图：

(a) 计算支配关系。

(b) 找出一种深度优先排序。

(c) 对(b)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。

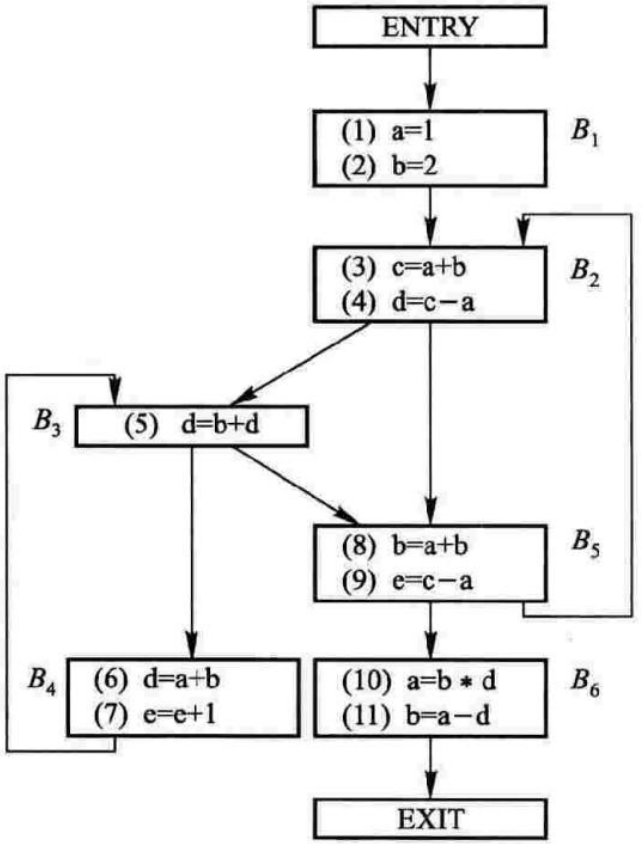


图 9.32 一个流图

习题9.15

(a) 支配关系如下：

B_1是B_1到B_6的支配节点

B_2是B_2到B_6的支配节点

B_3是B_3和B_4的支配节点

B_4是B_4的支配节点

B_5是B_5和B_6的支配节点

B_6是B_6的支配节点

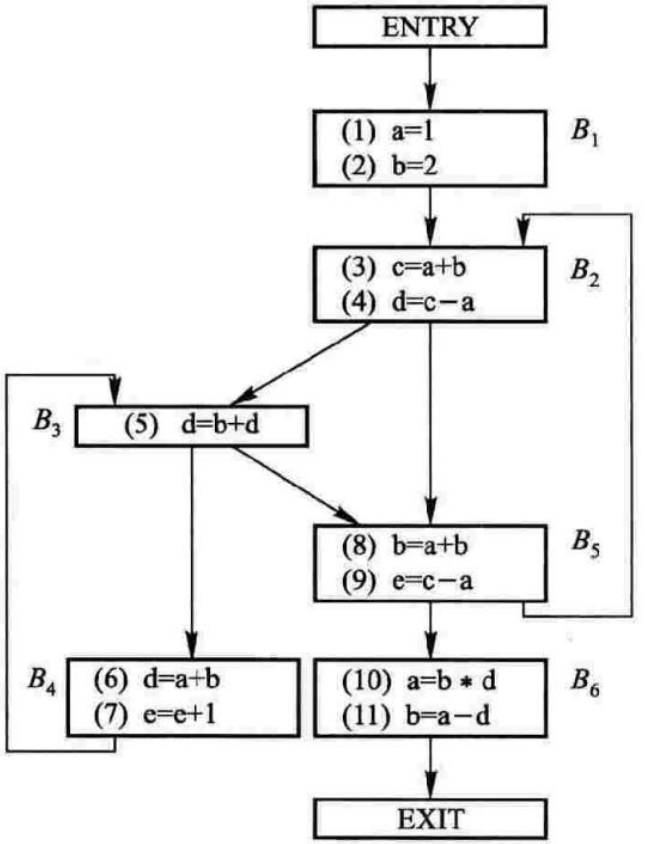


图 9.32 一个流图

习题9.15

(b) 找出一种深度优先排序。

最简单的是 $B_1, B_2, B_3, B_4, B_5, B_6$。（如有其他答案，需要确保是深度优先排序。）

~~其他答案示例：~~

- ~~$B_1, B_2, B_3, B_5, B_6, B_4$~~
- ~~$B_1, B_2, B_5, B_6, B_3, B_4$~~
- 注：深度优先排序是深度优先后序遍历的逆序，后序遍历书上讲了是先右后左，因此只有一种可能。

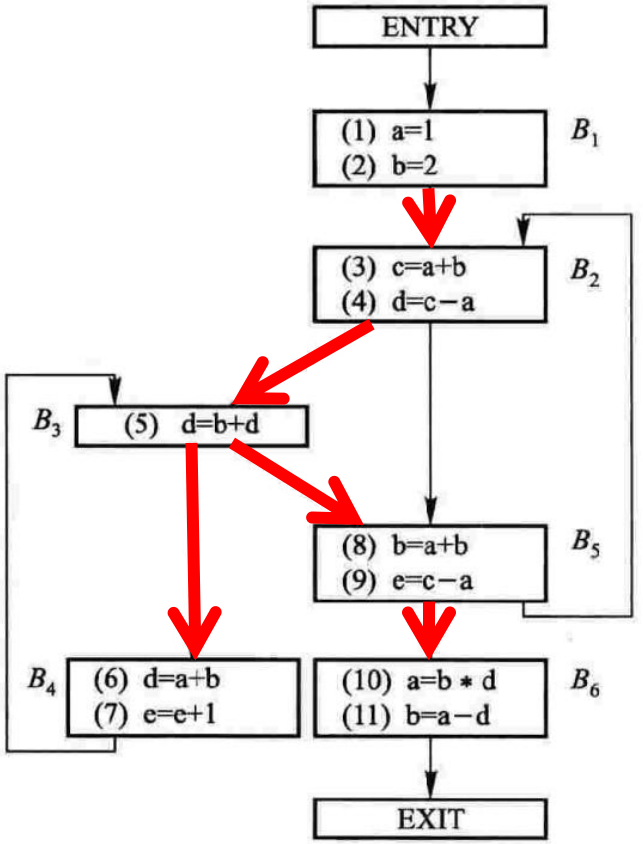


图 9.32 一个流图

习题9.15

(c) 对(b)的结果，标明前进边、后撤边和交叉边。

按$B_1, B_2, B_3, B_4, B_5, B_6$的顺序，前进边为：

$$B_1 \rightarrow B_2$$
$$B_2 \rightarrow B_3$$
$$B_3 \rightarrow B_4$$
$$B_3 \rightarrow B_5$$
$$B_5 \rightarrow B_6$$

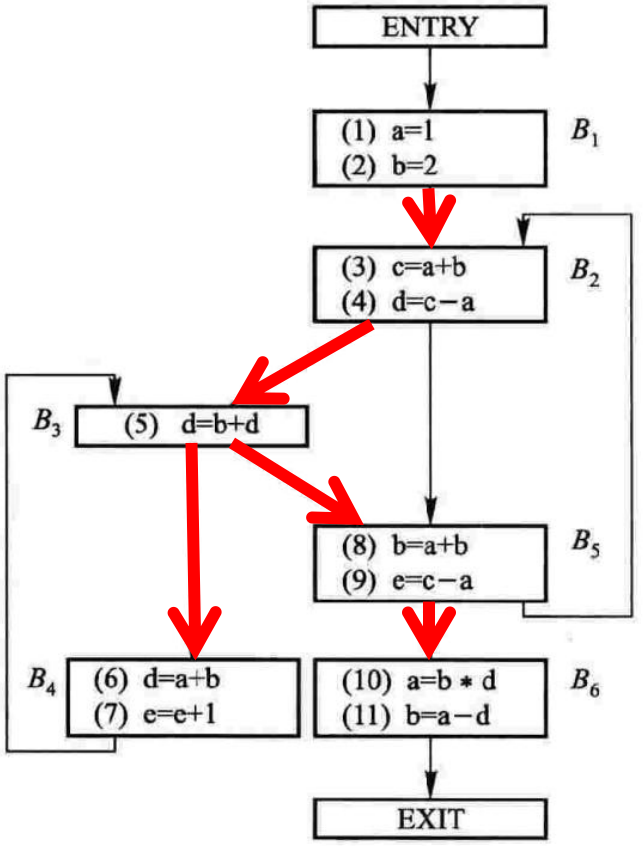


图 9.32 一个流图

(c) 对(b)的结果，标明前进边、后撤边和交叉边。

按$B_1, B_2, B_3, B_4, B_5, B_6$的顺序，后撤边为：

$$B_4 \rightarrow B_3$$
$$B_5 \rightarrow B_2$$

交叉边：无

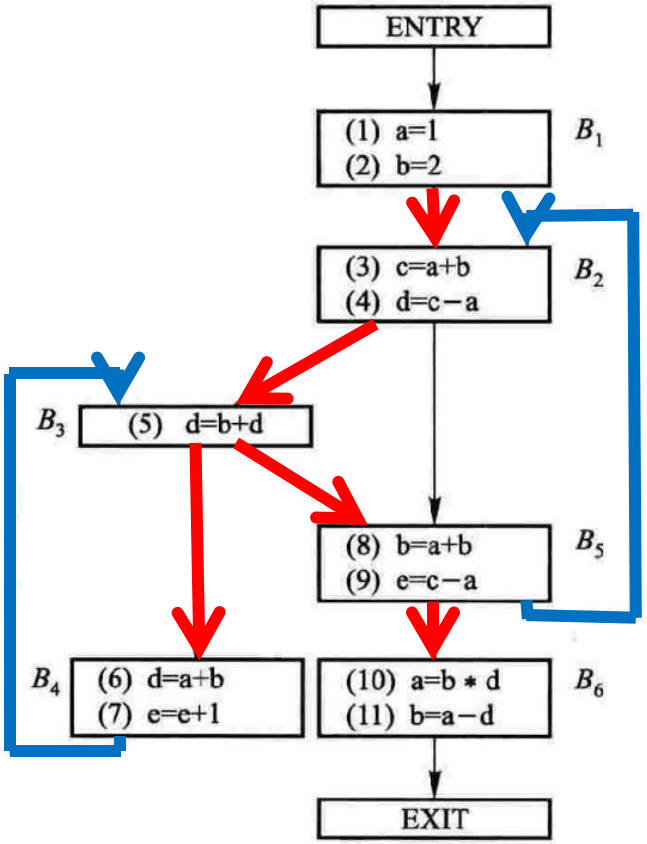


图 9.32 一个流图

9.15 (b)(c)一种错误答案 *

(b) 找出一种深度优先排序。

(c) 对(b)的结果，标明前进边、后撤边和交叉边。

按如图顺序为$B_1, B_2, B_3, B_4, B_5, B_6$，前进边为：$B_1 \to B_2$、$B_2 \to B_3$、$B_3 \to B_4$、$B_2 \to B_5$和$B_5 \to B_6$。

后撤边为：$B_4 \to B_3$和$B_5 \to B_2$。

交叉边为$B_3 \to B_5$。

错误：不是深度优先排序。

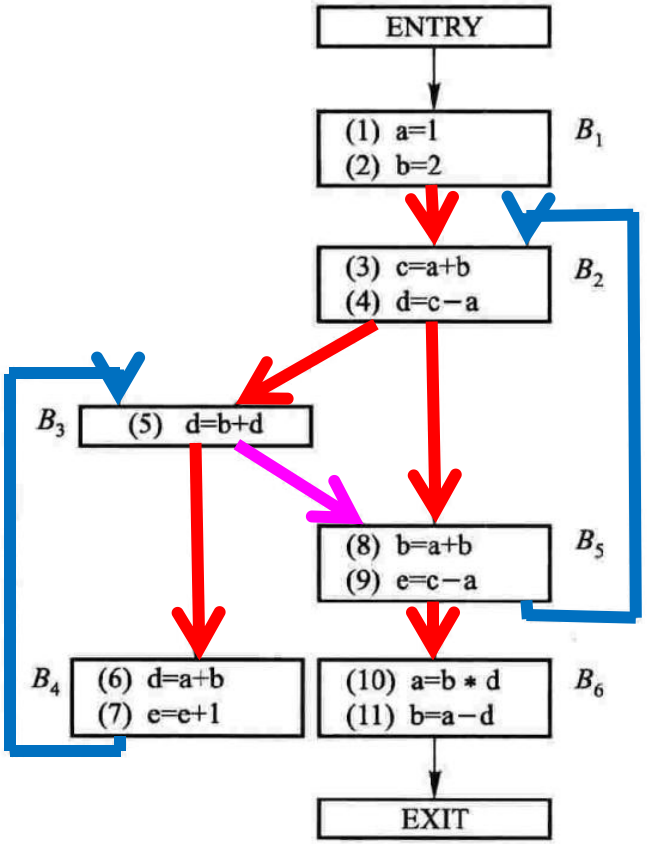


图 9.32 一个流图

习题9.15

(d) 该流图是否可归约。

(c)中提到后撤边为$B_4 \to B_3$和$B_5 \to B_2$。由于B_3是B_4的支配节点，因此$B_4 \to B_3$是一条回边，同理$B_5 \to B_2$也是一条回边。

流图删除所有回边后，可以验证剩下的图是有向无环图，因此该流图可归约。

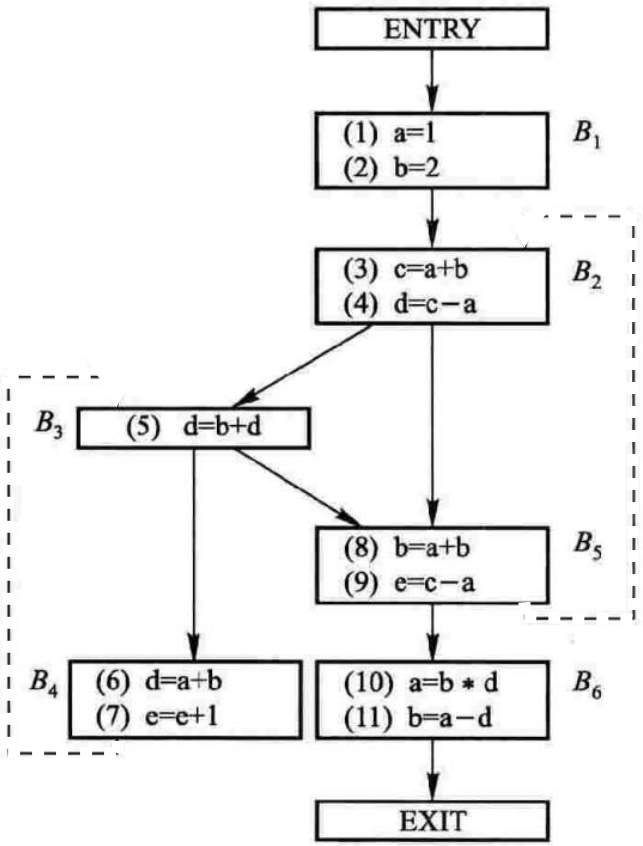


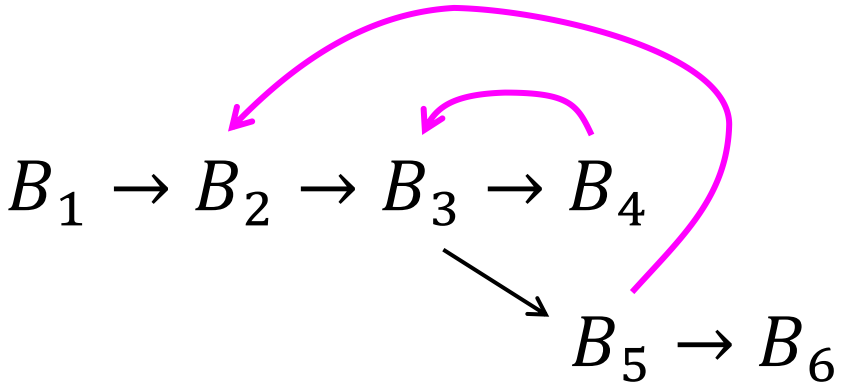
图 9.32 一个流图

习题9.15

(e) 计算该流图的深度。

流图的深度，是无环路径上的最大后撤边数，所以该流图的~~深度为2~~。

该流图的两条回边嵌套，深度为1.



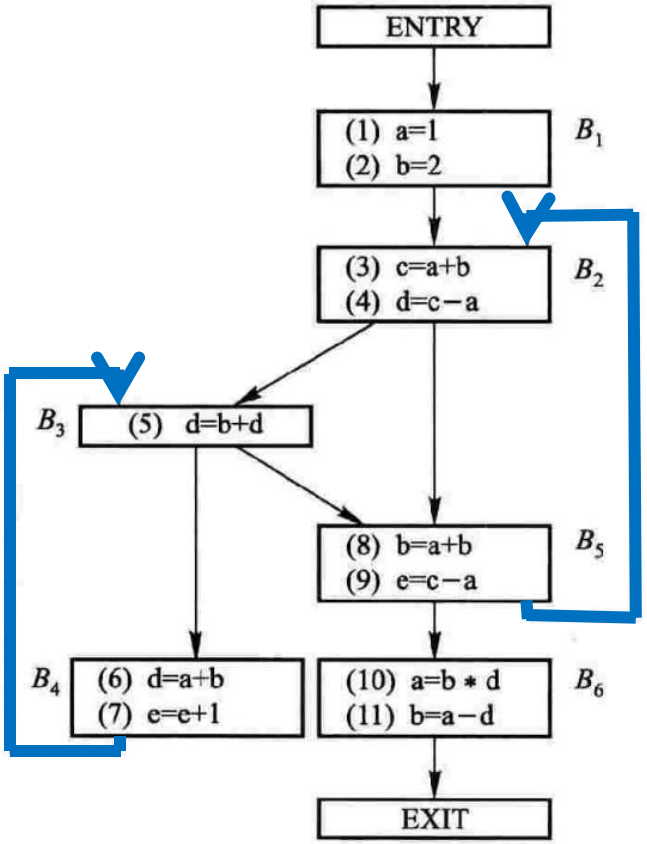
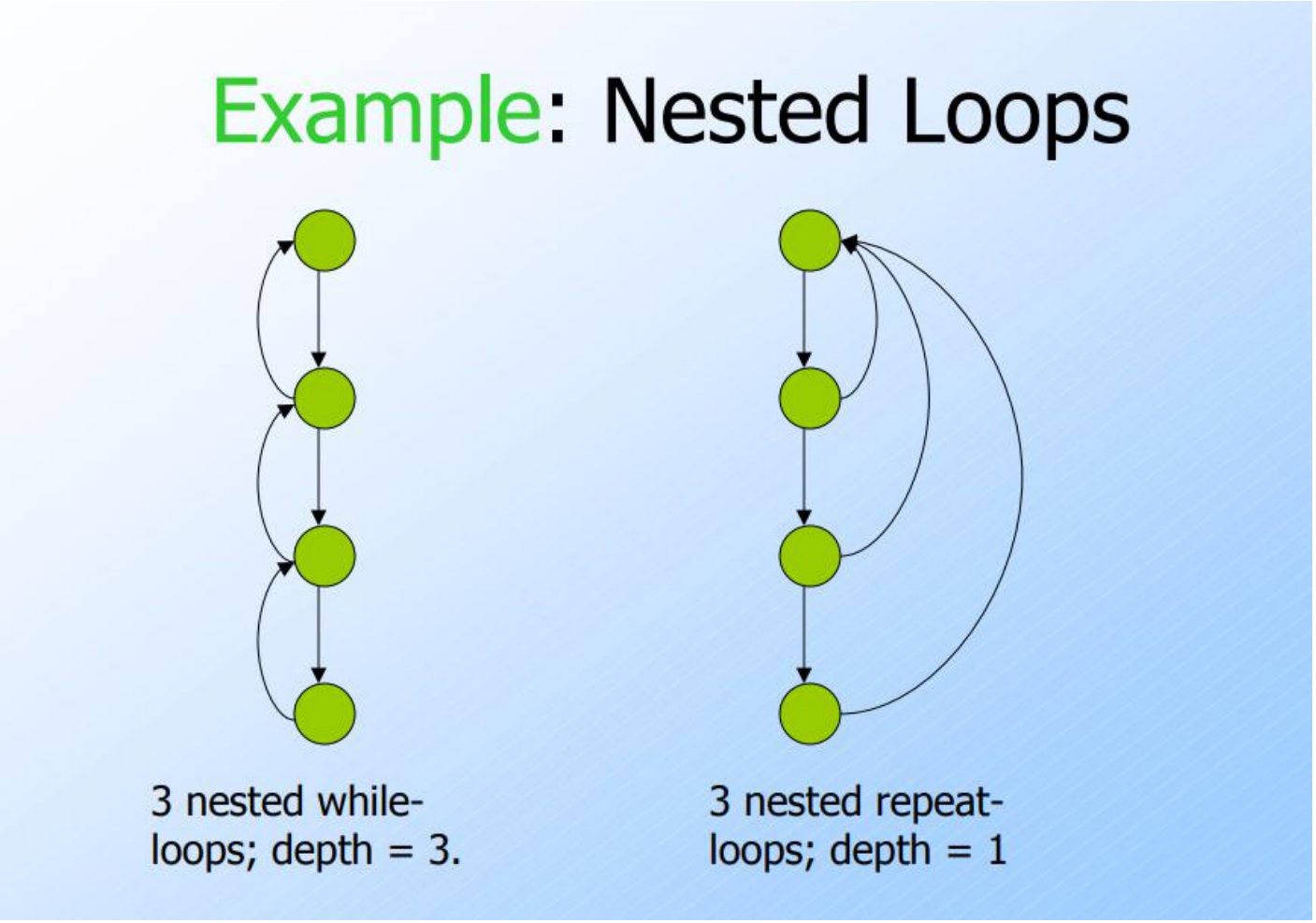


图 9.32 一个流图

附：流图深度举例：


Example: Nested Loops
3 nested while-loops; depth = 3.
3 nested repeat-loops; depth = 1

习题9.15

(f) 找出该流图的自然循环。

回边$B_4 \to B_3$确定的自然循环为$\{B_3, B_4\}$;

回边$B_5 \to B_2$确定的自然循环为$\{B_2, B_3, B_4, B_5\}$。

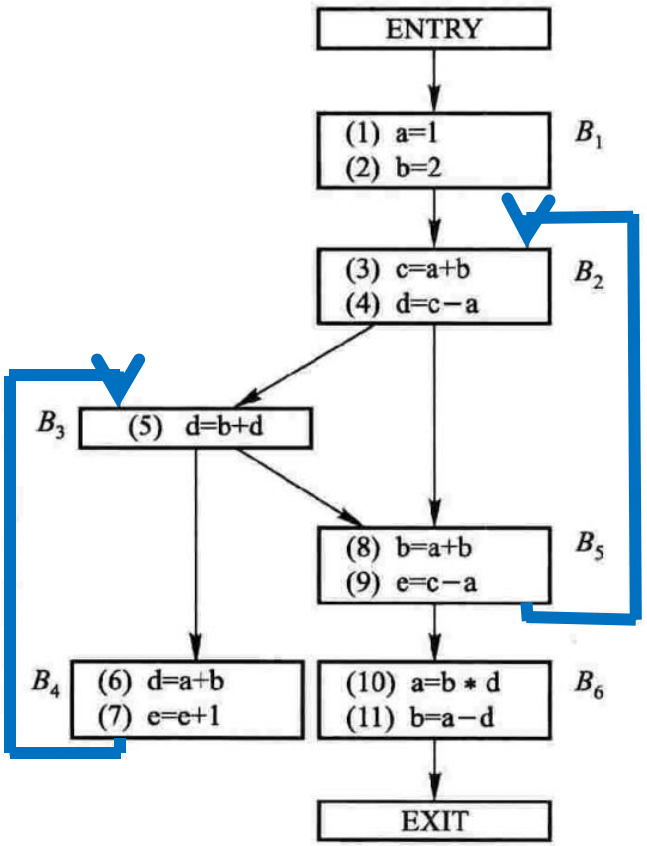


图 9.32 一个流图

习题6.6

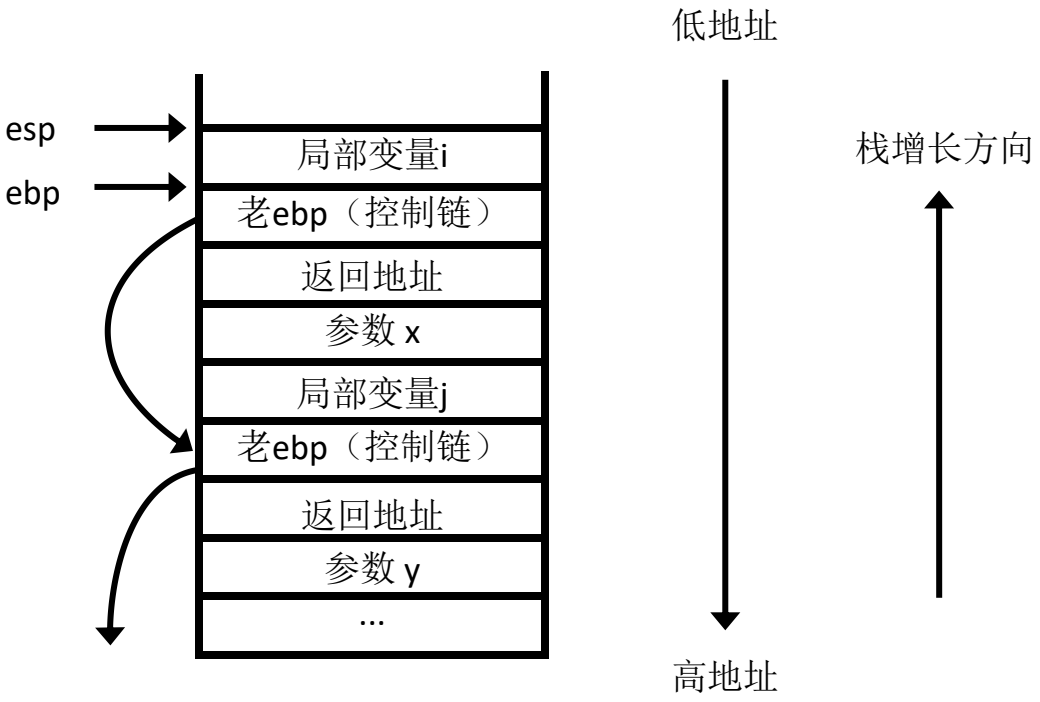
- 下面是C语言的两个函数f和g的概略（它们不再有其他的局部变量）：

```
int f(int x) { int i; ... return i +1; ...}

int g(int y) {int j; ... f(j+1); ...}
```

请按照图6.11的形式，画出函数g调用f，f的函数体正在执行时，活动记录栈的内容及相关信息，并按图6.10左侧箭头方式画出控制链。假定函数返回值是通过寄存器传递的。

习题6.6



注：题目已标注返回值通过寄存器传递，因此不需要为返回值分配空间，否则为错误！

习题6.10

❑一个C语言程序如下：

```
func(i1, i2, i3) long i1, i2, i3; {
        long j1, j2, j3;
        printf("Addresses of i1,i2,i3=%o,%o,%o\n", &i1, &i2, &i3);
        printf("Addresses of j1,j2,j3=%o,%o,%o\n", &j1, &j2, &j3);
}
main() {
        long i1, i2, i3;
        func(i1, i2, i3);
}
```

习题6.10

该程序在x86/Linux 系统上，经某编译器编译后的运行结果如下：

Addresses of il ,i2 ,i3 =27777775460 ,27777775464 ,27777775470

Addresses of jl ,j2 ,j3 =27777775444 ,27777775440 ,27777775434

从上面的结果可以看出 ,func 函数的三个形式参数的地址依次升高，而三个局部变量的地址依次降低。试说明为什么会有这个区别。注意，输出的数据是八进制的。

答：实参逆序进栈，因此地址依次增加；而局部变量按声明顺序分配空间，因此地址依次减小。

习题6.13

❑ 一个C语言程序如下：

```
int fact( i ) int i;  {
        if ( i = = 0 )
                return 1 ;
        else
                return i* fact ( i-1 ) ;
}
main ( ) {
        printf(“%d\n”, fact(5));
        printf(“%d\n", fact(5, 10, 15));
        printf(“%d\n", fact(5.0));
        printf("%d\n", fact() ) ;
}
```

该程序在x86/Linux 系统上，用编译器 GCC:（GNU）egcs-2. 91. 66 19990314/Linux (egcs-1 . 1. 2release) 编译后，运行结果如下：

```
120
120
1
Segmentation fault(core dumped)
```

请解释下面问题:

(a) 第二个 fact 调用：结果为什么没有受参数过多的影响?

(b) 第三个 fact 调用：为什么用浮点数 5.0 作为参数时结果变成 1？

(c) 第四个 fact 调用：为什么没有提供参数时会出现 Segmentation fault?

(a) 三个参数逆序入栈，第一个参数5在栈中所有参数里地址最小；而函数fact只取了一个参数，因此不会受到影响。

(b) 5.0按浮点数格式入栈，但函数fact用int类型来读取参数。由于小端（little endian）机器将低地址存储低位，因此读入int类型的值为0，fact结果变成了1。

(c) 由于没有参数，main函数没有在栈中分配实参位置，且main函数也没有局部变量，因此调用fact之后，fact取到的“参数”值为main函数控制链（老ebp / 虚拟基地址）。这个值可能很大，递归调用后栈溢出产生段错误。

作业

中国科学技术大学 University of Science and Technology of China

- 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
```

三地址代码：

代价：0

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

R1	R2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f					

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

代价：0

```
LD R1, a
LD R2, b
ADD R1, R1, R2
```

R1	R2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f					

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+4

代价： **4**

```
LD R1, a
LD R2, b
ADD R1, R1, R2
```

R1	R2
a	b

a	b	c	d	e	f	t1	t2	t3	t4	t5
a, R1	b, R2	c	d	e	f					

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+1

代价： **5**

```
LD R1, a
LD R2, b
ADD R1, R1, R2
```

R1	R2
t1	b

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b, R2	c	d	e	f	R1				

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

代价：5

LD R2, c
SUB R2, R1, R2

R1	R2
t1	b

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b, R2	c	d	e	f	R1				

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+2

代价： 7

LD R2, c
SUB R2, R1, R2

R1	R2
t1	c

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c, R2	d	e	f	R1				

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+1

代价：8

LD R2, c

SUB R2, R1, R2

R1	R2
t1	t2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	R1	R2			

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

代价：8

```
ST t1, R1
ST t2, R2
LD R1, d
LD R2, e
ADD R1, R1, R2
```

R1	R2
t1	t2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	R1	R2			

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+4

代价：12

```
ST t1, R1
ST t2, R2
LD R1, d
LD R2, e
ADD R1, R1, R2
```

R1	R2
t1	t2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1,R1	t2,R2			

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+4

代价：16

```
ST t1, R1
ST t2, R2
LD R1, d
LD R2, e
ADD R1, R1, R2
```

R1	R2
d	e

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d, R1	e, R2	f	t1	t2			

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+1

代价：17

```
ST t1, R1
ST t2, R2
LD R1, d
LD R2, e
ADD R1, R1, R2
```

R1	R2
t3	e

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e, R2	f	t1	t2	R1		

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

代价：17

LD R2, t2
MUL R1, R1, R2

R1	R2
t3	e

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e, R2	f	t1	t2	R1		

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+2

代价： 19

LD R2, t2

MUL R1, R1, R2

R1	R2
t3	t2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2,R2	R1		

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+1

代价：**20**

LD R2, t2

MUL R1, R1, R2

R1	R2
t3	t2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2,R2	R1		

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

代价：20

```
LD R2, t1
ADD R2, R1, R2
```

R1	R2
t3	t2

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2,R2	R1		

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+2

代价： 22

LD R2, t1

ADD R2, R1, R2

R1	R2
t3	t1

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1,R2	t2	R1		

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+1

代价：23

LD R2, t1

ADD R2, R1, R2

R1	R2
t3	t4

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2	R1	R2	

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

代价：23

```
ST t4, R2
LD R2, e
SUB R2, R1, R2
```

R1	R2
t3	t4

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2	R1	R2	

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+2

代价： 25

ST t4, R2
LD R2, e
SUB R2, R1, R2

R1	R2
t3	t4

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2	R1	t4,R2	

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+2

代价： 27

ST t4, R2
LD R2, e
SUB R2, R1, R2

R1	R2
t3	e

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e, R2	f	t1	t2	R1	t4	

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+1

代价： 28

ST t4, R2
LD R2, e
SUB R2, R1, R2

R1	R2
t3	t5

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2	R1	t4	R2

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

代价：28

```
LD R1, t4
MUL R1, R1, R2
ST f, R1
```

R1	R2
t3	t5

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2	R1	t4	R2

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+2

代价：30

```
LD R1, t4
MUL R1, R1, R2
ST f, R1
```

R1	R2
t4	t5

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f	t1	t2		t4,R1	R2

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+1

代价：31

```
LD R1, t4
MUL R1, R1, R2
ST f, R1
```

R1	R2
f	t5

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	R1	t1	t2		t4	R2

三地址代码：

t1 = a + b

t2 = t1 - c

t3 = d + e

t3 = t2 * t3

t4 = t1 + t3

t5 = t3 - e

f = t4 * t5

+2

代价：33

```
LD R1, t4
MUL R1, R1, R2
ST f, R1
```

R1	R2
f	t5

a	b	c	d	e	f	t1	t2	t3	t4	t5
a	b	c	d	e	f, R1	t1	t2		t4	R2

(a) 目标代码：

```
LD R1, a
LD R2, b
ADD R1, R1, R2
LD R2, c
SUB R2, R1, R2
ST t1, R1
ST t2, R2
LD R1, d
LD R2, e
ADD R1, R1, R2
LD R2, t2
MUL R1, R1, R2
LD R2, t1
ADD R2, R1, R2
ST t4, R2
LD R2, e
SUB R2, R1, R2
LD R1, t4
MUL R1, R1, R2
ST f, R1
```

(b) 寄存器描述符和变量的地址描述符见前。

(c) 目标代码的总代价为33，且为最小代价。

《编译原理与技术》习题课

The end!