

编译原理

习题课 (4)

第七章 语义分析和中间代码产生

- 翻译成四元式、构造翻译模式
 - 布尔表达式
 - 赋值语句
 - 控制语句

(1) $E \rightarrow E_1 \text{ or } M E_2$
 { backpatch(E_1 .falselist, M.quad);
 E .truelist:=merge(E_1 .truelist, E_2 .truelist);
 E .falselist:= E_2 .falselist }

(2) $E \rightarrow E_1 \text{ and } M E_2$
 { backpatch(E_1 .truelist, M.quad);
 E .truelist:= E_2 .truelist;
 E .falselist:=merge(E_1 .falselist, E_2 .falselist) }

(5) $E \rightarrow id_1 \text{ relop } id_2$
 { E .truelist:=makelist(nextquad);
 E .falselist:=makelist(nextquad+1);
 emit('j' relop.op ',' id_1 .place ',' id_2 .place ',' '0');
 emit('j, - , - , 0') }

(6) $E \rightarrow id$
 { E .truelist:=makelist(nextquad);
 E .falselist:=makelist(nextquad+1);
 emit('jnz' ',' id.place ',' '-' ',' '0') ;
 emit('j - , - , 0') }

(3) $E \rightarrow \text{not } E_1$
 { E .truelist:= E_1 .falselist;
 E .falselist:= E_1 .truelist }

(4) $E \rightarrow (E_1)$
 { E .truelist:= E_1 .truelist;
 E .falselist:= E_1 .falselist }

~~(7) $M \rightarrow \epsilon$ { M .quad:=nextquad }~~

(1) $E \rightarrow E_1$ or $M E_2$

```
{ backpatch( $E_1$ .falselist, M.quad);  
   $E$ .truelist:=merge( $E_1$ .truelist,  $E_2$ .truelist);  
   $E$ .falselist:= $E_2$ .falselist }
```

(2) $E \rightarrow E_1$ and $M E_2$

```
{ backpatch( $E_1$ .truelist, M.quad);  
   $E$ .truelist:= $E_2$ .truelist;  
   $E$ .falselist:=merge( $E_1$ .falselist,  $E_2$ .falselist) }
```

(5) $E \rightarrow id_1 \text{ relop } id_2$

```
{  $E$ .truelist:=makelist(nextquad);  
   $E$ .falselist:=makelist(nextquad+1);  
  emit('j' relop.op ','  $id_1$ .place ','  $id_2$ .place ',' '0');  
  emit('j, - , - , 0') }
```

(6) $E \rightarrow id$

```
{  $E$ .truelist:=makelist(nextquad);  
   $E$ .falselist:=makelist(nextquad+1);  
  emit('jnz' ',' id.place ',' ' - ' ',' ' 0') ;
```

(3) $E \rightarrow \text{not } E_1$

```
{  $E$ .truelist:= $E_1$ .falselist;  
   $E$ .falselist:= $E_1$ .truelist }
```

(4) $E \rightarrow (E_1)$

```
{  $E$ .truelist:= $E_1$ .truelist;  
   $E$ .falselist:= $E_1$ .falselist }
```

~~(7) $M \rightarrow \epsilon$ { M .quad:=nextquad }~~

P218-6. 按 7.4.2 节的办法，写出布尔式 $A \text{ or } (B \text{ and not } (C \text{ or } D))$ 的四元式序列。

四元式序列

100. (jnz, A, -, 0)

101. (j, -, -, 102)

102. (jnz, B, -, 104)

103. (j, -, -, 0)

104. (jnz, C, -, 103)

105. (j, -, -, 106)

106. (jnz, D, -, 104)

107. (j, -, -, 100)



falselist



trueelist

$S \rightarrow \text{if } E \text{ then } M \ S_1$
 { backpatch(E.truelist, M.quad);
 $S.\text{nextlist} := \text{merge}(E.\text{falselist}, S_1.\text{nextlist})$ }

$S \rightarrow \text{if } E \text{ then } M_1 \ S_1 \ N \ \text{else } M_2 \ S_2$
 { backpatch(E.truelist, $M_1.\text{quad}$);
 backpatch(E.falselist, $M_2.\text{quad}$);
 $S.\text{nextlist} := \text{merge}(S_1.\text{nextlist}, N.\text{nextlist}, S_2.\text{nextlist})$ }

$M \rightarrow \varepsilon$ { $M.\text{quad} := \text{nextquad}$ }

$N \rightarrow \varepsilon$ { $N.\text{nextlist} := \text{makelist}(\text{nextquad}, \dots)$ }

□ $\text{emit}(j, S, -, -')$

□ 标号与 goto 语句

□ CASE 语句

■ 过程调用的翻译

$S \rightarrow \text{while } M_1 \ E \ \text{do } M_2 \ S_1$
 { backpatch(E.truelist, $M_2.\text{quad}$);
 backpatch($S_1.\text{nextlist}$, $M_1.\text{quad}$);
 $S.\text{nextlist} := E.\text{falselist}$
 $\text{emit}(j, -, -, 'M_1.\text{quad})$ }

面的语句翻

do A:=A+2

```
S → if E then M S1
{ backpatch(E.truelist, M.quad);
  S.nextlist := merge(E.falselist, S1.nextlist) }
```

```
S → if E then M1 S1 N else M2 S2
{ backpatch(E.truelist, M1.quad);
  backpatch(E.falselist, M2.quad);
  S.nextlist := merge(S1.nextlist, N.nextlist, S2.nextlist) }
```

```
M → ε { M.quad := nextquad }
```

```
N → ε { N.nextlist := makelist(nextquad, ...
  emit('j, - , - , - ' ) }
```

```
S → while M1 E do M2 S1
{ backpatch(E.truelist, M2.quad);
  backpatch(S1.nextlist, M1.quad);
  S.nextlist := E.falselist
  emit('j, - , - , ' M1.quad) }
```

P218-7. 用 7.5.1 节的办法，把下面的语句翻译成四元式序列：

while $A < C$ and $B < D$ do

if $A=1$ then $C:=C+1$

else while $A \leq D$ do $A:=A+2$

100. (j<, A, C, 102)

101. (j, -, -, 0)

102. (j<, B, D, 104)

103. (j, -, -, 101)

104. (j=, A, '1', 106)

105. (j, -, -, 109)

106. (+, C, '1', T1)

107. (:=, T1, -, C)

108. (j, -, -, 100)

109. (j≤, A, D, 111)

110. (j, -, -, 100)

111. (+, A, '2', T2)

112. (:=, T2, -, A)

113. (j, -, -, 109)

114. (j, -, -, 100)

12. Pascal 语言中 for 语句的一般形式为

for v:=initial to final do S

其意义如下:

begin

t1:=initial; t2:=final;

if $t1 \leq t2$ then begin

v:=t1;

S;

while $v \neq t2$ do begin

v:=succ(v);

S;

end

end

end

(1) 设有下列 Pascal 程序:

program

forloop(input,output);

var i, initial, final: integer;

begin

read (initial, final);

for i:=initial to final do


writeln (i)

end

当 initial=MAXINT-

5、 final=MAXINT 时, 该程序的执行结果是什么? 其中 MAXINT 是目标机上能表示的最大整数。

(2) 试构造一个翻译模式, 把 Pascal 语言的 for 语句翻译成四元式。



```
program forloop (input,  
    output);  
var i, initial, final:integer;  
begin  
    read (initial, final);  
    for i:=initial to final do  
        writeln (i)  
    end
```

```
MAXINT – 5  
MAXINT – 4  
MAXINT – 3  
MAXINT – 2  
MAXINT – 1  
MAXINT
```

构造翻译模式

for $v := \text{initial}$ to final do S

其意义如下：

begin

$t1 := \text{initial}; t2 := \text{final};$

 if $t1 \leq t2$ then begin

$v := t1;$

$S;$

 while $v \neq t2$ do begin

$v := \text{succ}(v);$

$S;$

 end

 end

end

$S \rightarrow \text{for } id := E_1 \text{ to } E_2 \text{ do } S_1$

改造为：

$S \rightarrow F S_1$

$F \rightarrow \text{for } id := E_1 \text{ to } E_2 \text{ do}$

for v:=initial to final do S

其意义如下:

begin

t1:=initial; t2:=final;

if t1 ≤ t2 then begin

v:=t1;

S;

while v ≠ t2 do begin

v:=succ(v);

S;

end

end

end

$F \rightarrow$ for id:= E_1 to E_2 do

{

INITIAL=NEWTEMP;

emit(':=,' E_1 .PLACE ', -,' INITIAL);

FINAL=NEWTEMP;

emit(':=,' E_2 .PLACE ', -,' FINAL);

p:= nextquad+2;

emit('j≤,' INITIAL ', FINAL ', p);

F.nextlist:=makelist(nextquad);

emit('j, - , - , - ');

F.place:=lookup(id.name);

if F.place≠nil then

emit(':=', F.place, '-', INITIAL);

F.quad:=nextquad;

F.final:=FINAL;

}

for v:=initial to final do S

其意义如下:

begin

t1:=initial; t2:=final;

if t1 ≤ t2 then begin

v:=t1;

S;

while v ≠ t2 do begin

v:=succ(v);

S;

end

end

end

$S \rightarrow F S_1$

{

backpatch(S_1 .nextlist, nextquad)

$p := \text{nextquad} + 2;$

emit('j≠,' F.place', ' F.final ', ' p);

$S.\text{nextlist} := \text{merge}(F.\text{nextlist}, \text{makelist}(\text{nextquad}));$

emit('j, - , - , - ');

emit('succ,' F.place ', -, ' F.place);

$\text{emit}('j, - , - , ' F.\text{quad});$

}

```

F → for id:=E1 to E2 do
{
    INITIAL=NEWTEMP;
    emit(':=,' E1.PLACE ', -,' INITIAL);
    FINAL=NEWTEMP;
    emit(':=,' E2.PLACE ', -,' FINAL);
    p:= nextquad+2;
    emit('j≤,' INITIAL ', FINAL ', p);
    F.nextlist:=makelist(nextquad);
    emit('j, - , - , - ');
    F.place:=lookup(id.name);
    if F.place≠nil then
        emit( ':=', F.place, '-', INITIAL);
    F.quad:=nextquad;
    F.final:=FINAL;
}

```

```

S → F S1
{
    backpatch(S1.nextlist, nextquad)
    p:=nextquad+2;
    emit('j≠,' F.place ', F.final ', p );
    S.nextlist := merge(F.nextlist, makelist(nextquad));
    emit('j, - , - , - ');

    emit('succ, ' F.place ', -,' F.place);
    emit('j, - , - , ' F.quad);
}

```

构造翻译模式

for $v := \text{initial}$ to final do S

其意义如下:

begin

$t1 := \text{initial}; t2 := \text{final};$

 if $t1 \leq t2$ then begin

$v := t1;$

$S;$

 while $v \neq t2$ do begin

$v := \text{succ}(v);$

$S;$

 end

 end

end

$S \rightarrow \text{for } id := E_1 \text{ to } E_2 \text{ do } S_1$

改造为:

$S \rightarrow F \text{ do } M S_1$

$F \rightarrow \text{for } l := E_1 \text{ to } E_2$

$l \rightarrow id$

$M \rightarrow \varepsilon$

构造翻译模式

for $v := \text{initial}$ to final do S

其意义如下:

```
begin
  t1:=initial; t2:=final;
  if t1 ≤ t2 then begin
    v:=t1;
    S;
    while v ≠ t2 do begin
      v:=succ(v);
      S;
    end
  end
end
end
```

$S \rightarrow \text{for } id := E_1 \text{ to } E_2 \text{ do } S_1$

改造为:

$S \rightarrow F \text{ do } M S_1$

$F \rightarrow \text{for } l := E_1 \text{ to } E_2$

$l \rightarrow id$

```
l → id
{
  p:=lookup(id.name);
  if p <> nil then
    l.place := p
  else error
}

M → ε
{
  M.quad := nextquad
}
```


for v:=initial to final do S

其意义如下:

begin

t1:=initial; t2:=final;

if $t1 \leq t2$ then begin

v:=t1;

S;

while $v \neq t2$ do begin

v:=succ(v);

S;

end

end

end

$F \rightarrow \text{for } I := E_1 \text{ to } E_2$

{

F.falselist:= makelist(nextquad);

emit('j>,' E₁.place ',' E₂.place ',0');

emit(I.Place ':='E₁.place);

F.truelist := makelist(nextquad);

emit('j,-,-,0');

F.place := I.place;

F.end := E₂.place;

}

$S \rightarrow F \text{ do } M S_1$

{

backpatch(S₁.nextlist, nextquad);

backpatch(F.truelist, M.quad);

emit(F.place ':='F.place '+1');

emit('j<=,' F.place ',' F.end ',' M.quad);

S.nextlist := F.falselist;

}

```

S → F do M S1
{
    backpatch(S1.nextlist, nextquad);
    backpatch(F.truelist, M.quad);
    emit(F.place ':=' F.place '+' 1);
    emit('j', F.place ',', F.end ',', M.quad);
    S.nextlist := F.falselist;
}

```

```

F → for I := E1 to E2
{
    F.falselist := makelist(nextquad);
    emit('j>', E1.place ',', E2.place ',0');
    emit(I.Place ':=' E1.place);
    F.truelist := makelist(nextquad);
    emit('j,-,-,-');
    F.place := I.place;
    F.end := E2.place;
}

```

```

I → id
{
    p := lookup(id.name);
    if p <> nil then
        I.place := p
    else error
}

M → ε
{
    M.quad := nextquad
}

```

第九章 运行时存储空间组织

- 目标程序运行时的活动
- 运行时存储器的划分
- 静态存储管理
- 一个简单栈式存储分配
- 嵌套过程语言的栈式实现

P270-9. 对于下面的程序：

```
procedure P(X,Y,Z);  
begin  
    Y:=Y+1;  
    Z:=Z+X;  
end P;  
begin  
    A:=2;  
    B:=3;  
    P(A+B,A,A);  
    print A  
end
```

若参数传递的办法分别为（1）传名，（2）传地址，（3）得结果，以及（4）传值，试问，程序执行时所输出的 A 分别是什么？

（1）传名 $A = 9$

（2）传地址
 $A = 8$

（3）得结果
 $A = 7$


（4）传值 $A = 2$
20

第十章 优化

- 优化概述
- 局部优化
- 循环优化

P306-1. 试把以下程序划分为基本块并作出其程序流图。

```
    read C
    A:=0
    B:=1
L1:  A:=A+B
    if  $B \geq C$  goto L2
    B:=B+1
    goto L1
L2:  write A
    halt
```



```
read C
A: =0
B:=1
L1:A:=A+B
  if  $B \geq C$  goto L2
  B:=B+1
  goto L1
L2:write A
halt
```

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句

read C

A:=0

B:=1

L1:A:=A+B

if $B \geq C$ goto L2

B:=B+1

goto L1

L2:write A

halt

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句

read C

A:=0

B:=1

L1:A:=A+B

if $B \geq C$ goto L2

B:=B+1

goto L1

L2:write A

halt

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句

read C

A:=0

B:=1

L1:A:=A+B

if $B \geq C$ goto L2

B:=B+1

goto L1

L2:write A

halt

1. 求出四元式程序中各个基本块的入口语句：

- 1) 程序第一个语句，或
- 2) 能由条件转移语句或无条件转移语句转移到的语句，或
- 3) 紧跟在条件转移语句后面的语句

read C

A:=0

B:=1

L1:A:=A+B

if $B \geq C$ goto L2

B:=B+1

goto L1

L2:write A

halt

read C

A: =0

B:=1

L1:A:=A+B

if $B \geq C$ goto L2

B:=B+1

goto L1

L2:write A

halt

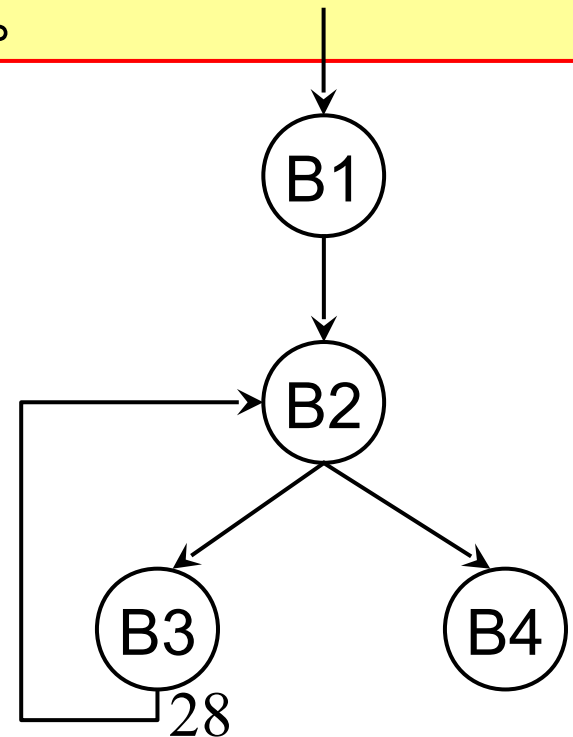
2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句（不包括该入口语句）、或到一转移语句（包括该转移语句）、或一停语句（包括该停语句）之间的语句序列组成的。

B1

B2

B3

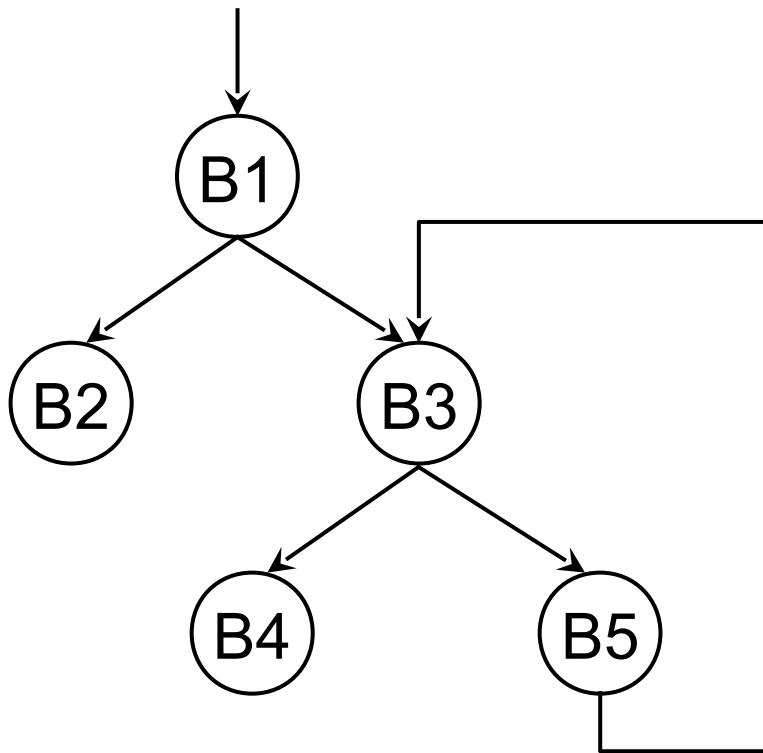
B4



P306-2. ☐ ☐ ☐ ☐ ☐ ☐

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

☐ ☐ ☐ ☐



read A, B

F:=1

C:=A*A

D:=B*B

if C<D goto L1

B1

E:=A*A

F:=F+1

E:=E+F

write E

halt

B2

L1: E:=B*B

F:=F+2

E:=E+F

write E

if E>100 goto L2

B3

halt

B4

L2: F:=F-1

goto L1

B5

P306-3. 试对以下基本块 B1 和 B2 :

B1: A:=B*C

D:=B/C

E:=A+D

F:=2*E

G:=B*C

H:=G*G

F:=H*G

L:=F

M:=L

B2: B:=3

D:=A+C

E:=A*C

G:=B*F

H:=A+C

I:=A*C

J:=H+I

K:=B*5

L:=K+J

M:=L

分别应用 DAG 对它们进行优化，并就以下两种情况分别写出优化后的四元式序列：

(1) 假设只有 G、L、M 在基本块后面还要被引用；

(2) 假设只有 L 在基本块后面还要被³⁰引用。

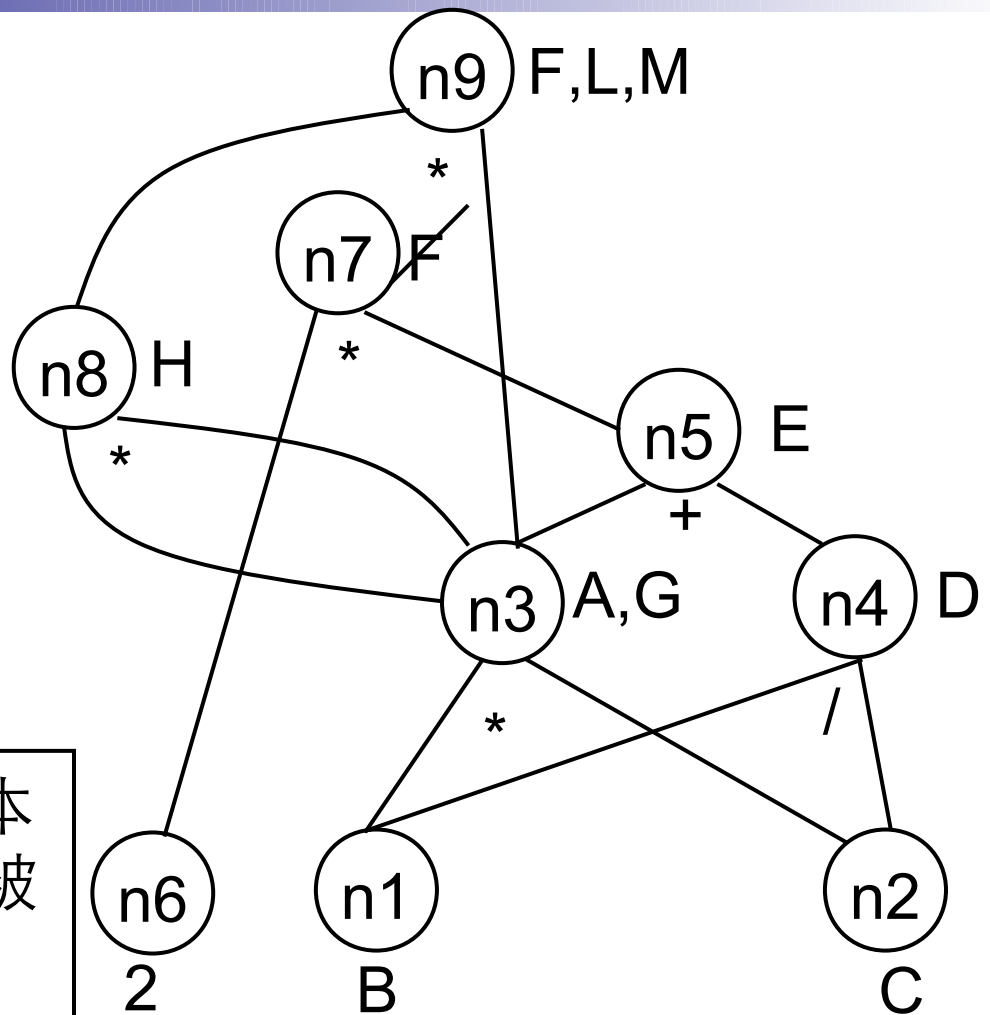
B1: $A := B * C$
 $D := B / C$
 $E := A + D$
 $F := 2 * E$
 $G := B * C$
 $H := G * G$
 $F := H * G$
 $L := F$
 $M := L$

只有 G,L,M
 在基本块后面
 还要被引用:

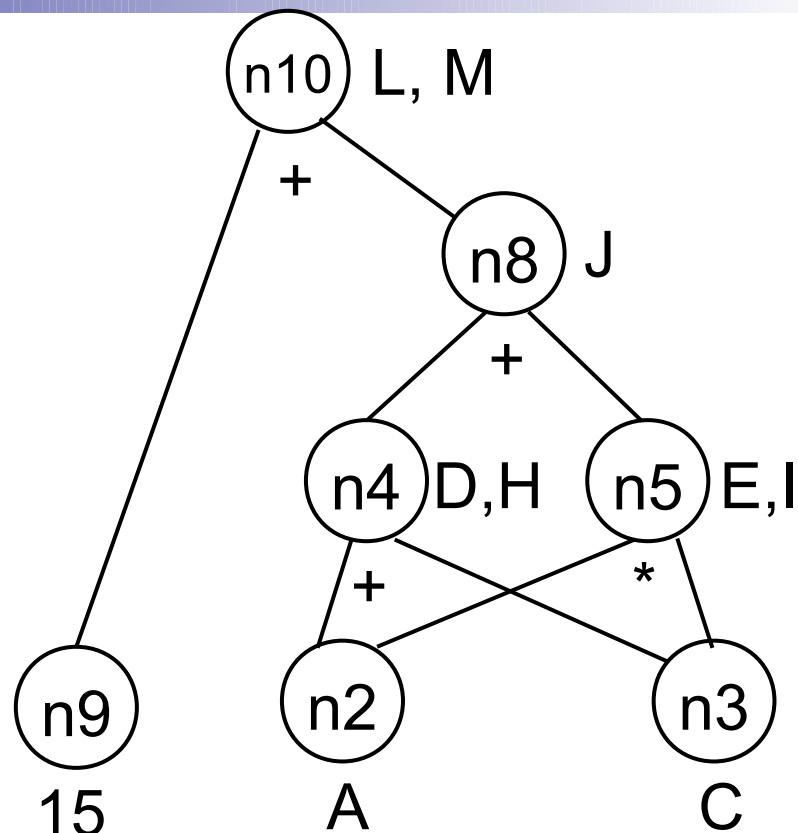
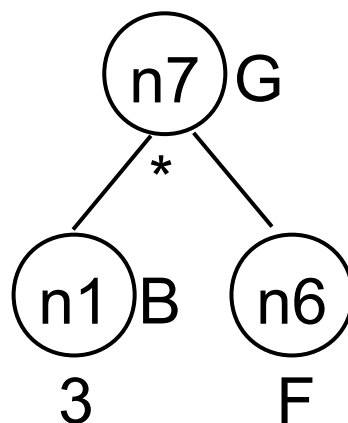
$G := B * C$
 $H := G * G$
 $L := H * G$
 $M := L$

只有 L 在基本
 块后面还要被
 引用:

$G := B * C$
 $H := G * G$
 $L := H * G$



B2: B:=3
 D:=A+C
 E:=A*C
 G:=B*F
 H:=A+C
 I:=A*C
 J:=H+I
 K:=B*5
 L:=K+J
 M:=L



只有 G,L,M 在基本块后面还要被引用

:

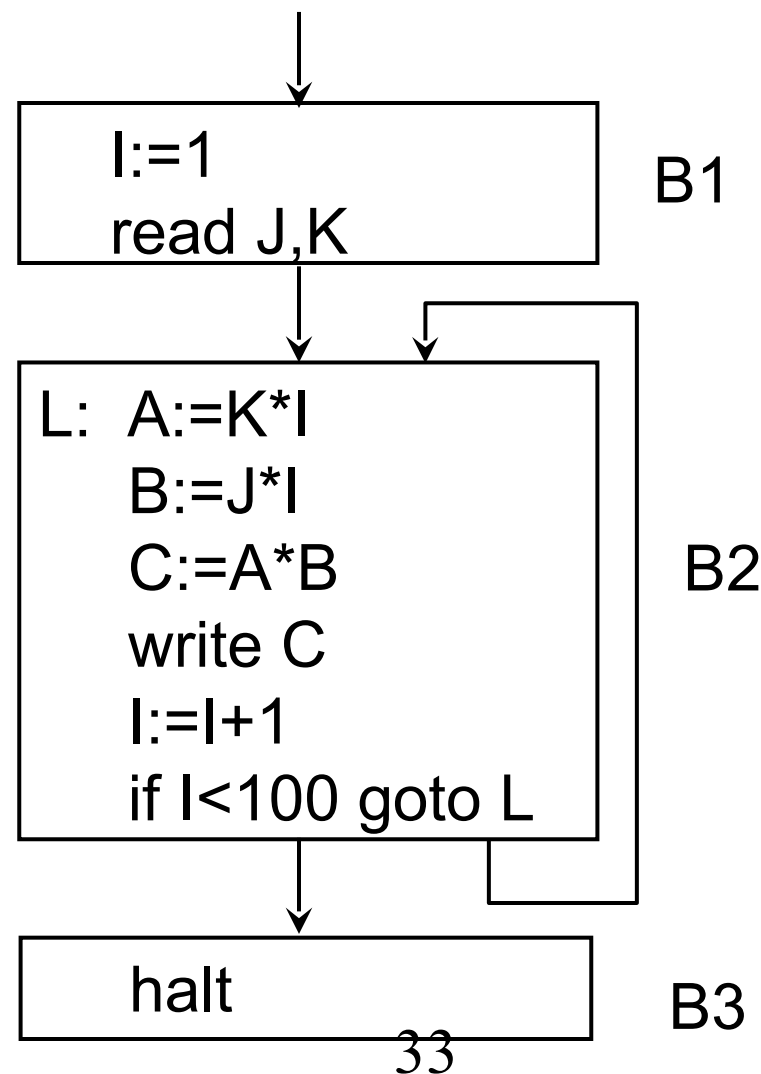
G:=3*F
 D:=A+C
 E:=A*C
 J:=D+E
 L:=15+J

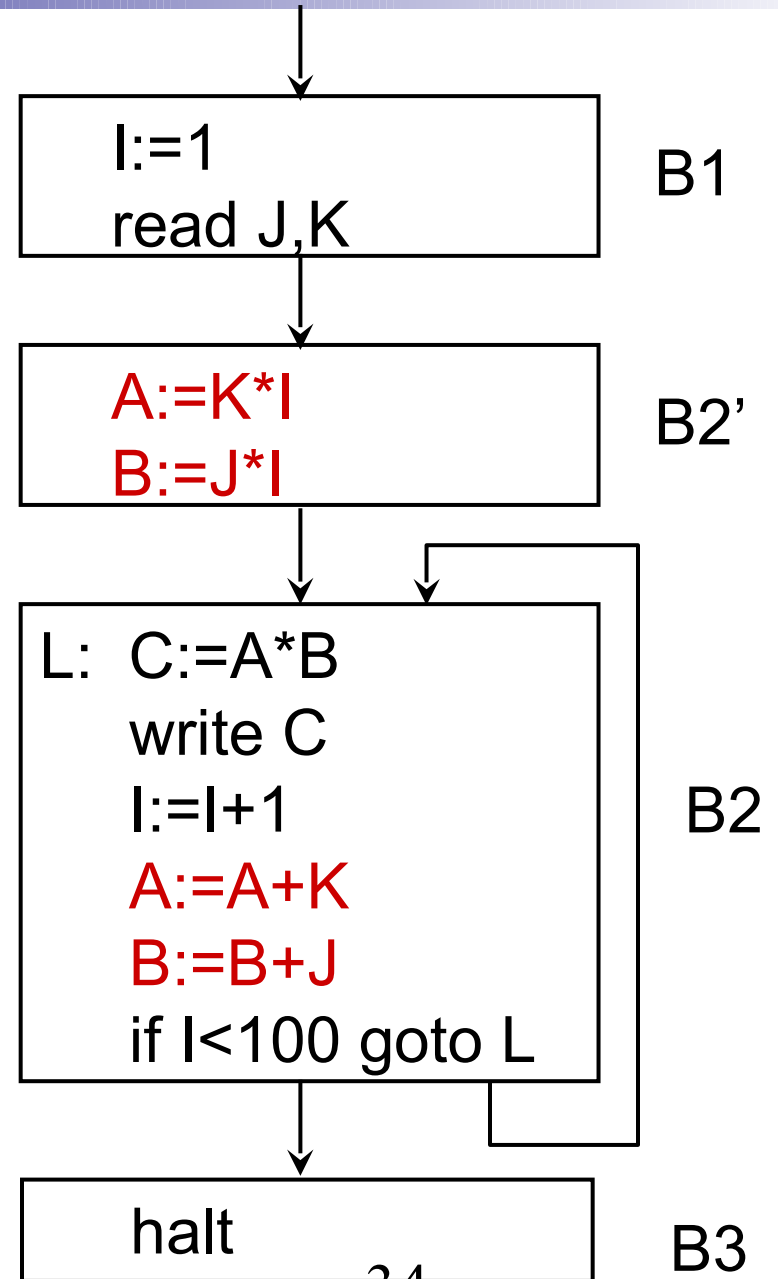
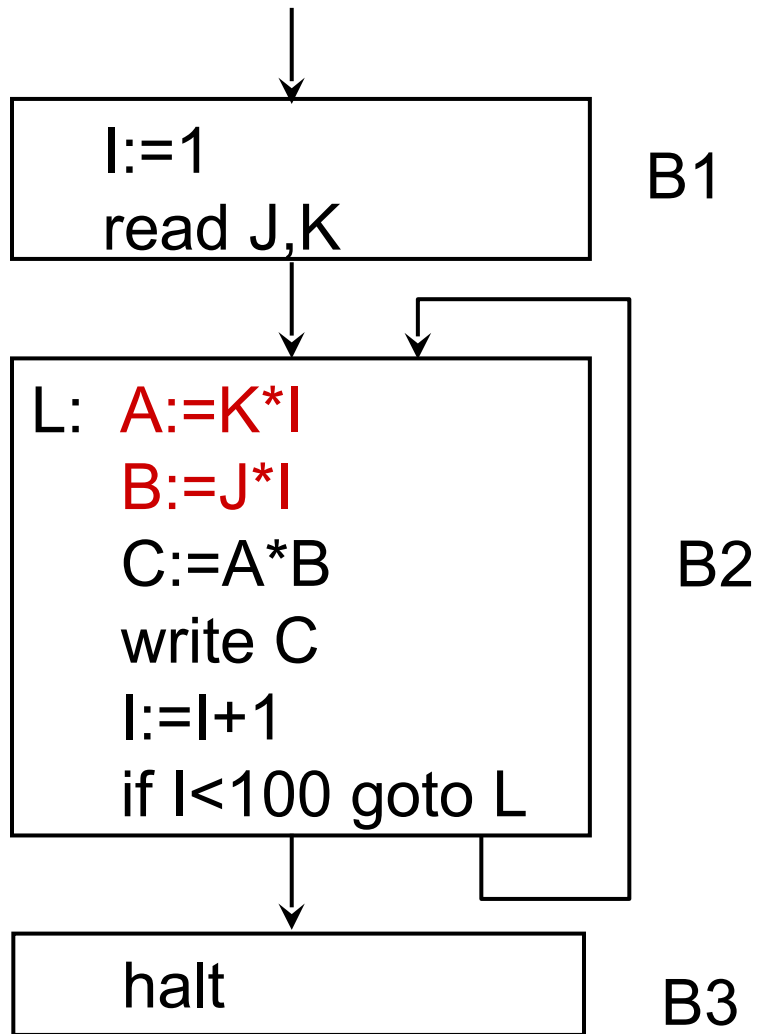
只有 L 在基本块后面还要被引用:

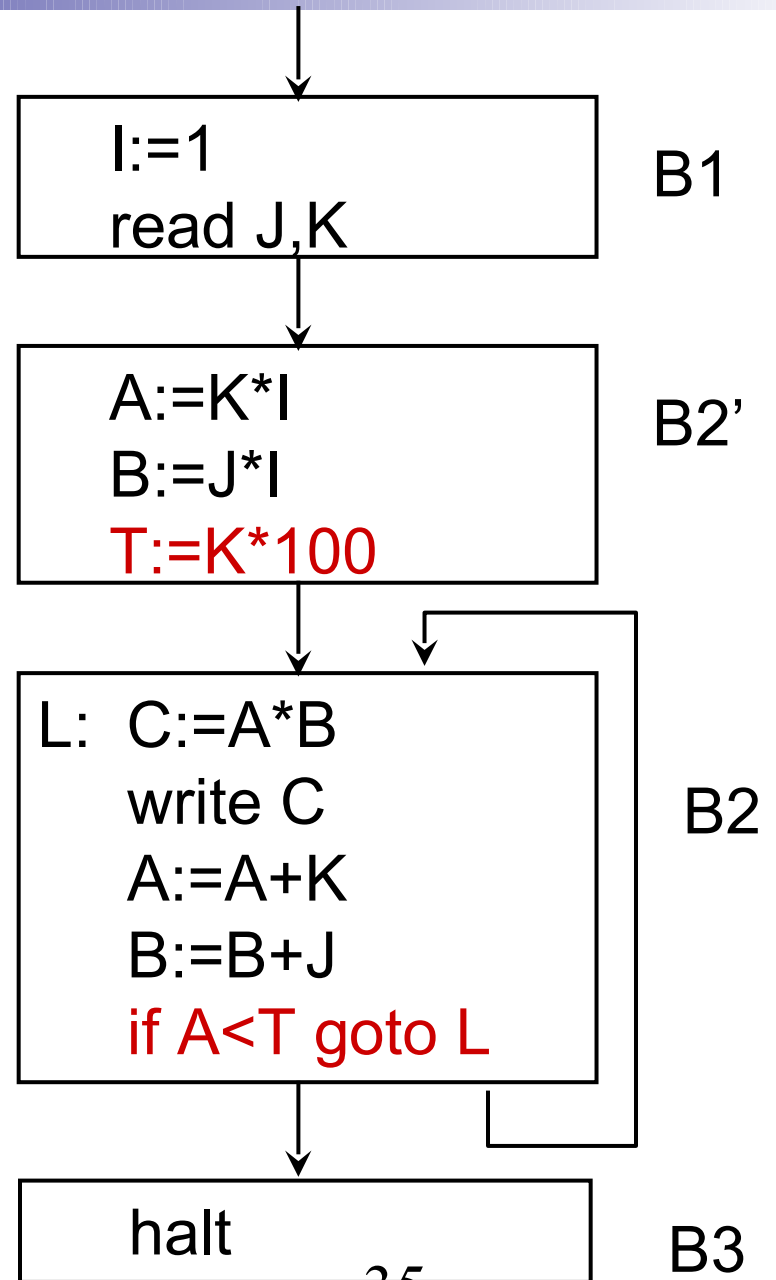
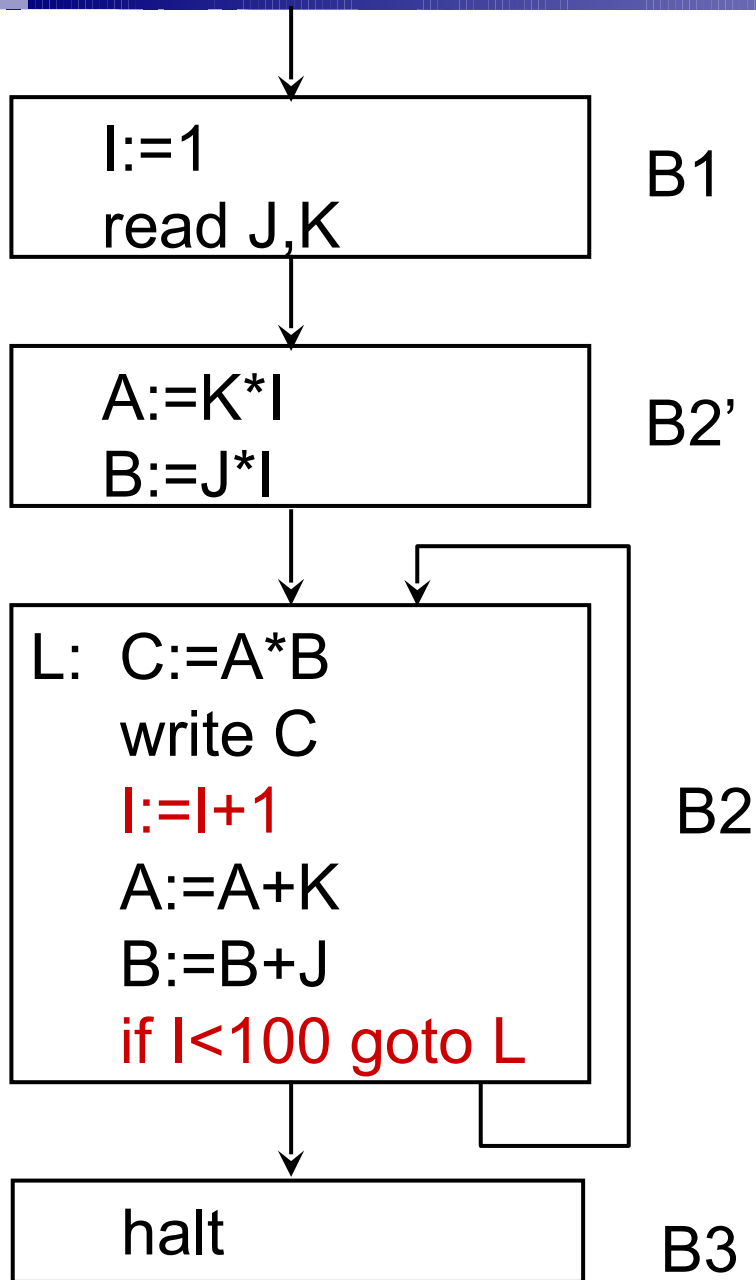
D:=A+C
 E:=A*C
 J:=D+E
 L:=15+J

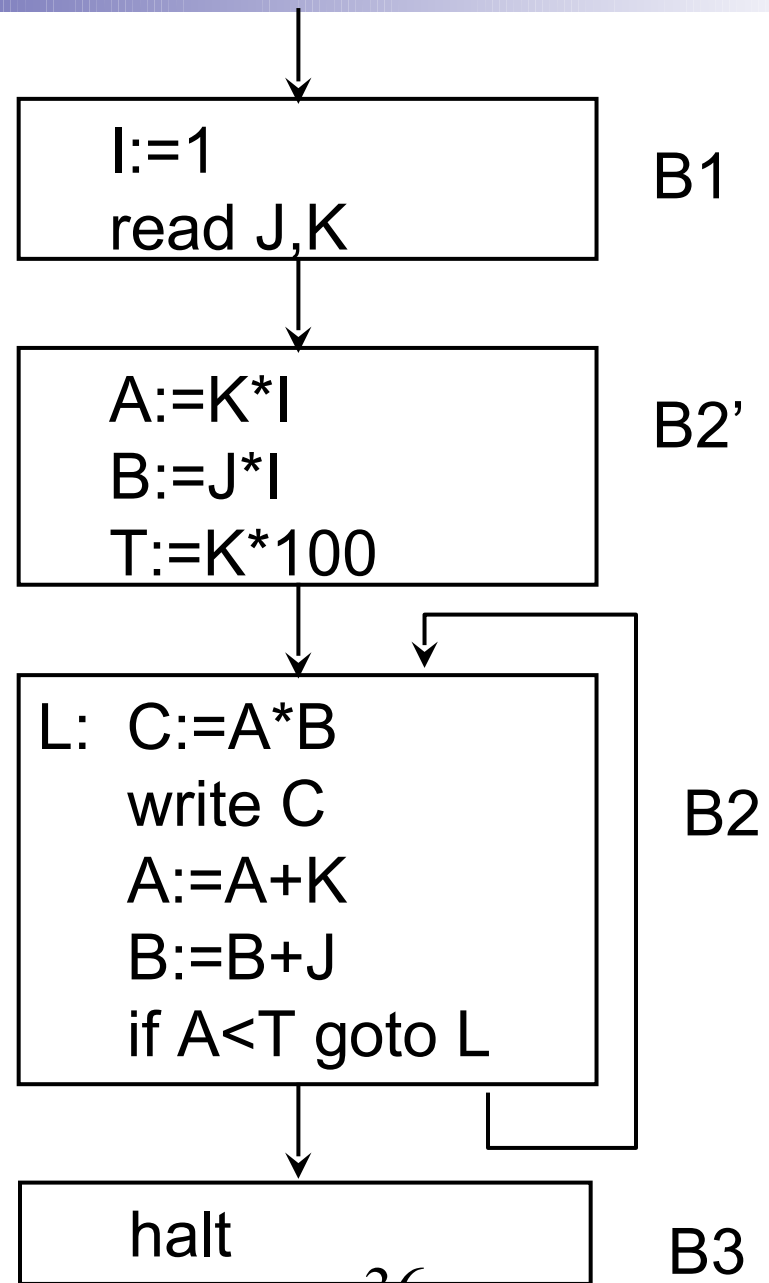
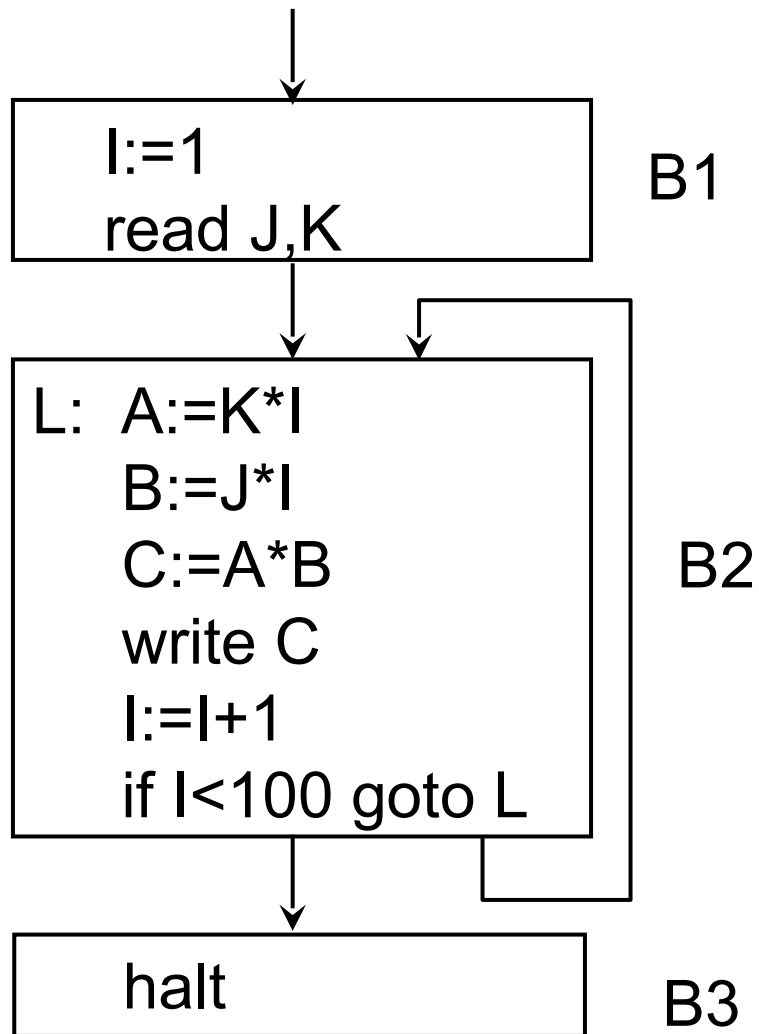
P306-4. 对以下四元式程序，对其中循环进行循环优化。

```
      I:=1
      read J, K
L:    A:=K*I
      B:=J*I
      C:=A*B
      write C
      I:=I+1
      if I<100 goto L
      halt
```



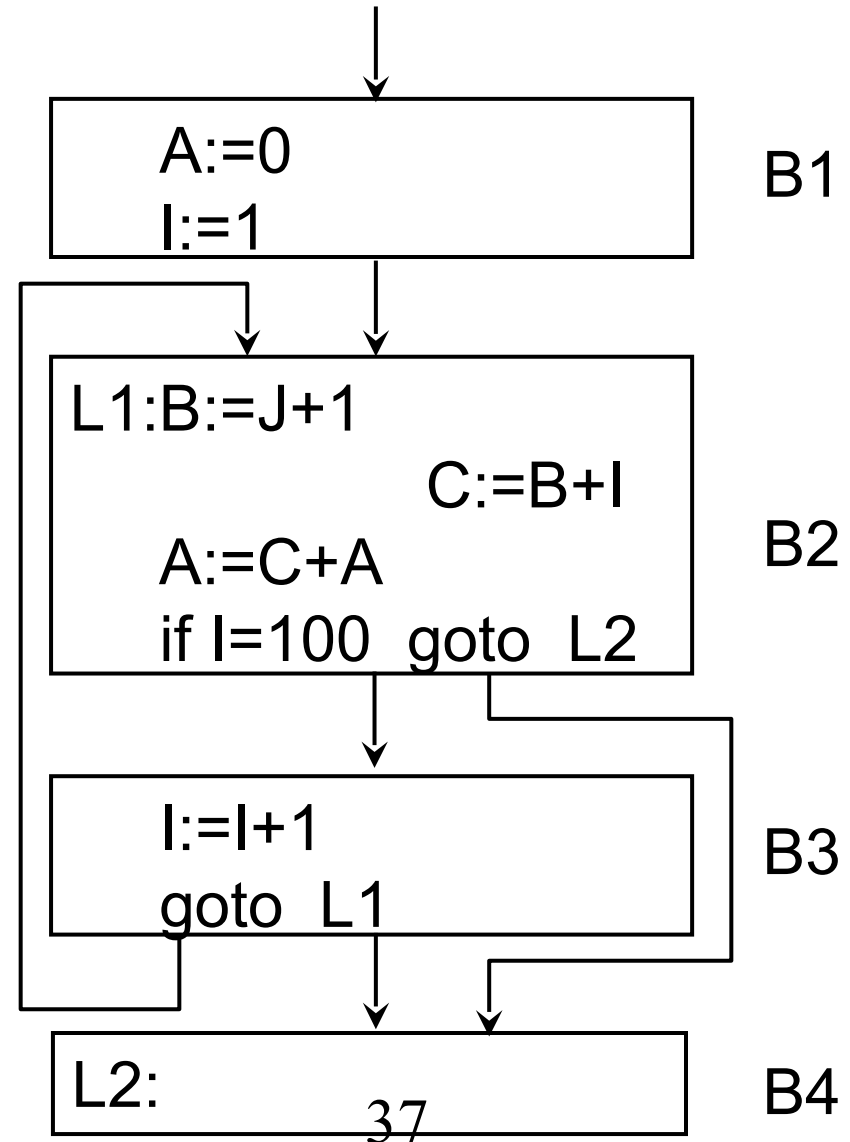


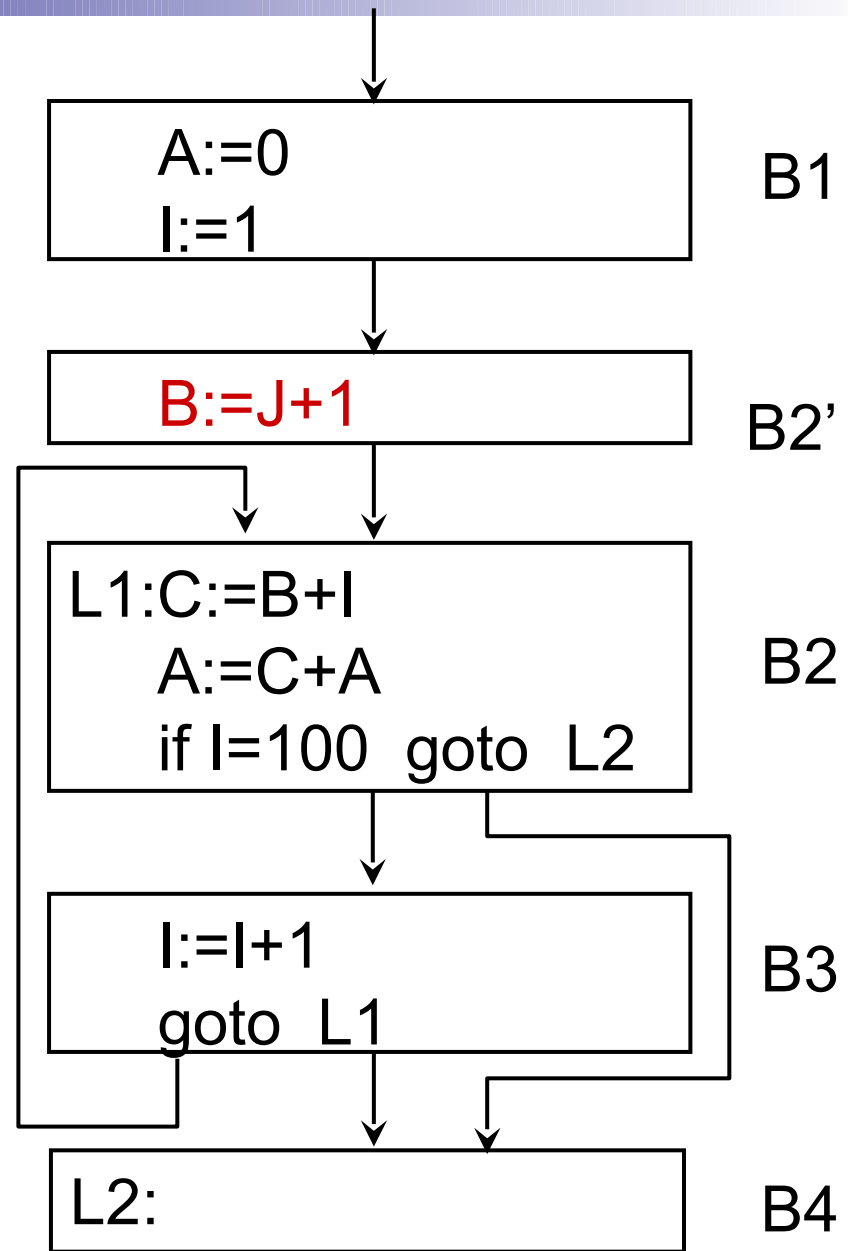
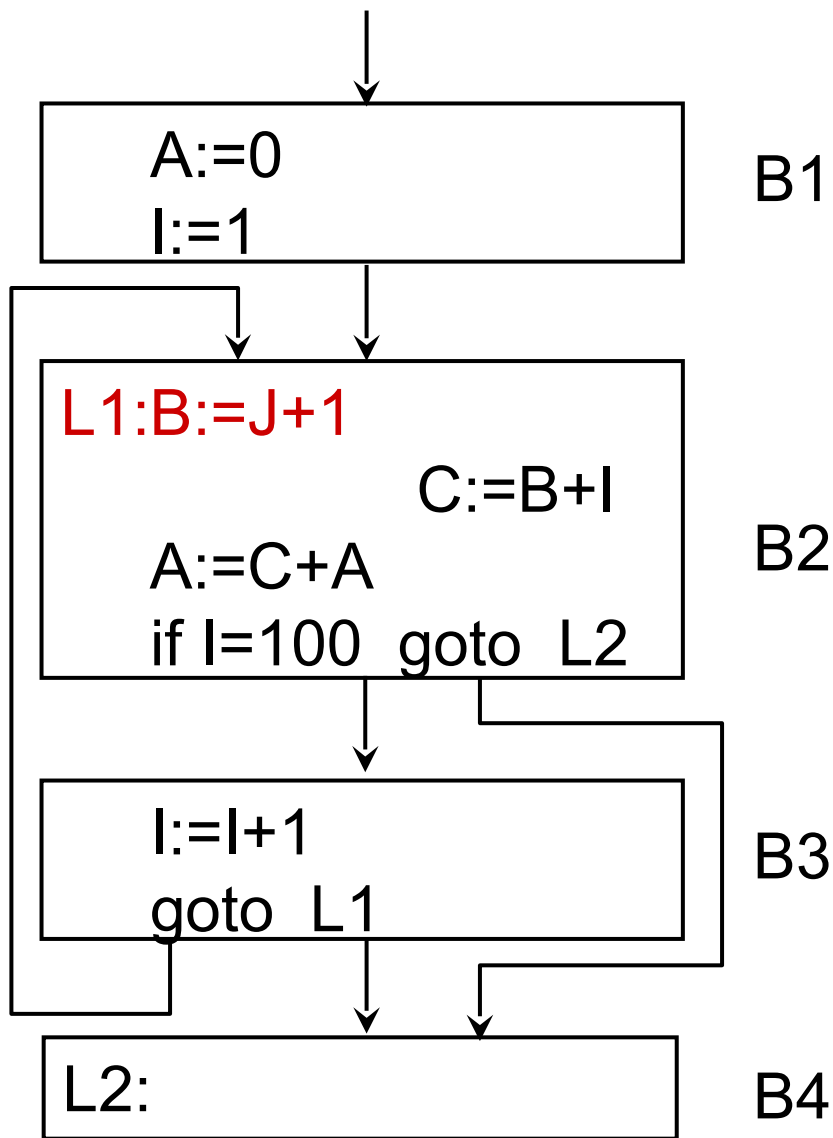


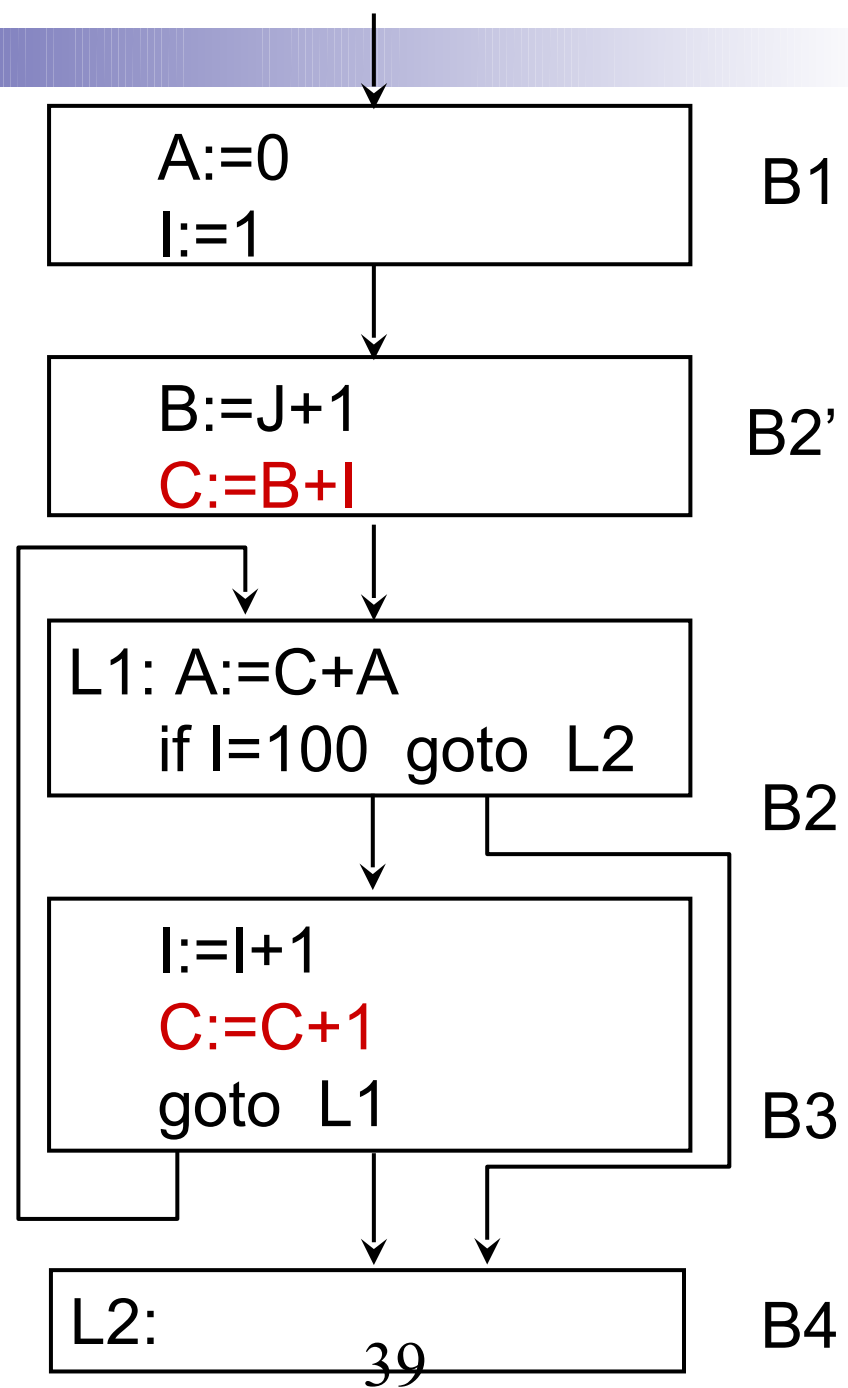
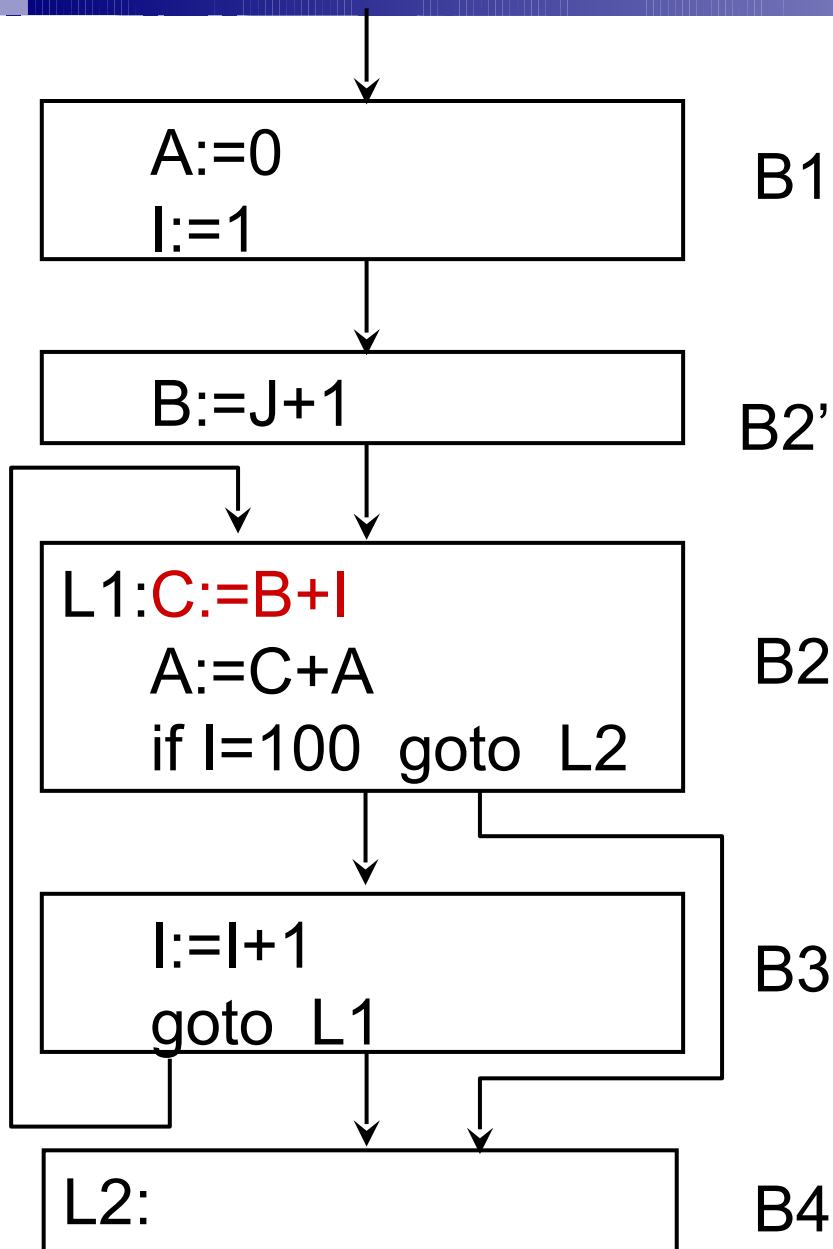


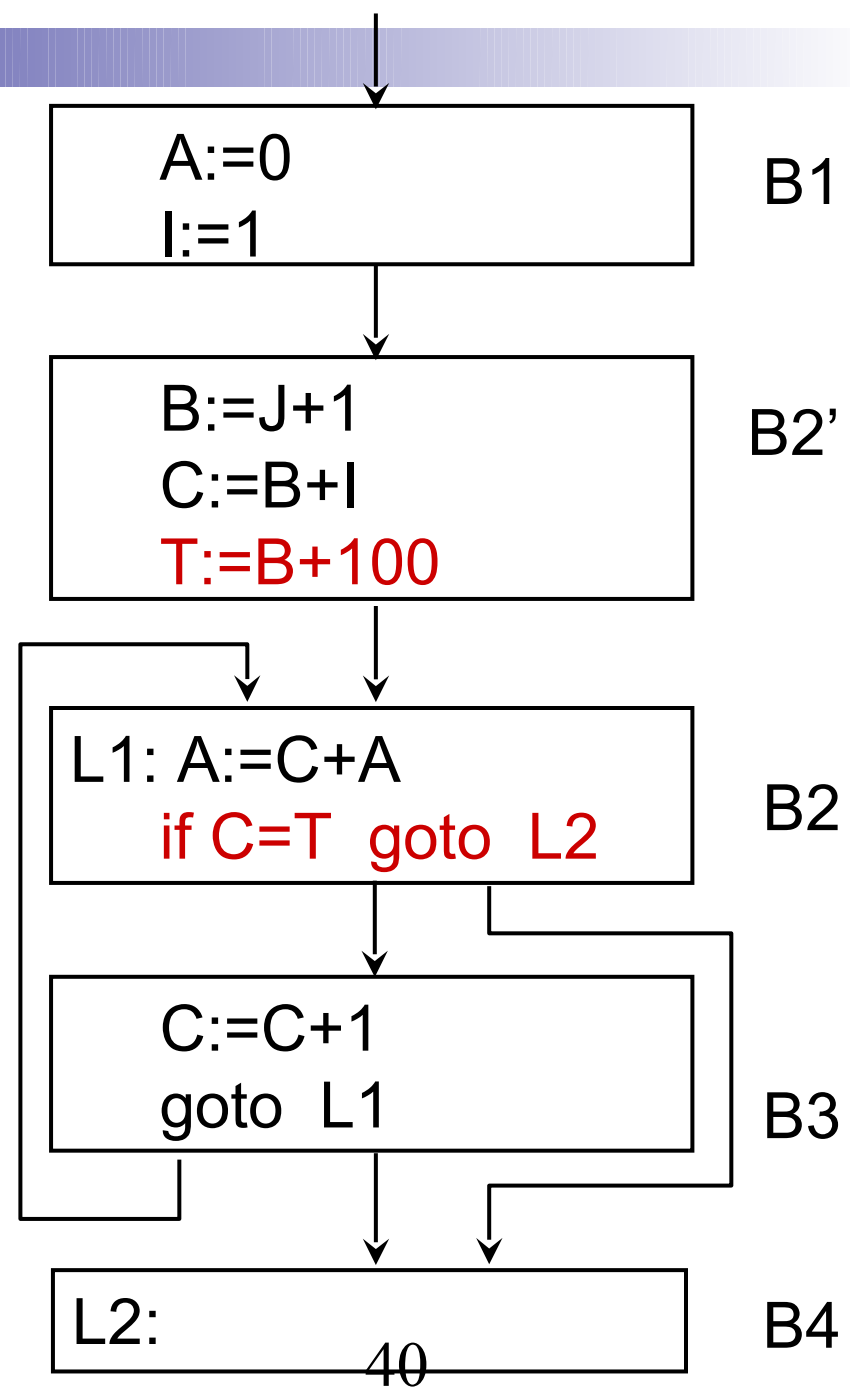
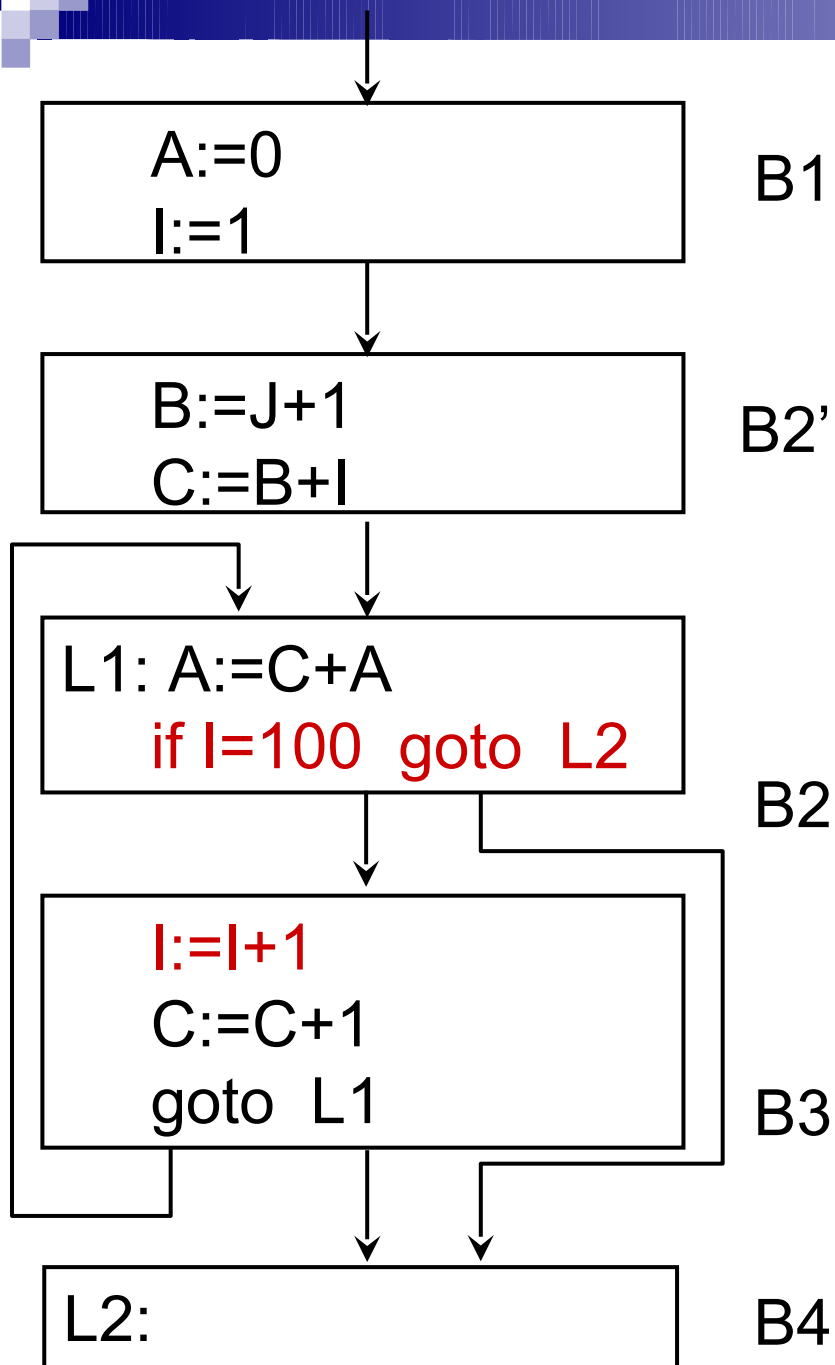
P306-5. 以下程序是某程序的最内循环，是
对它进行循环优化。

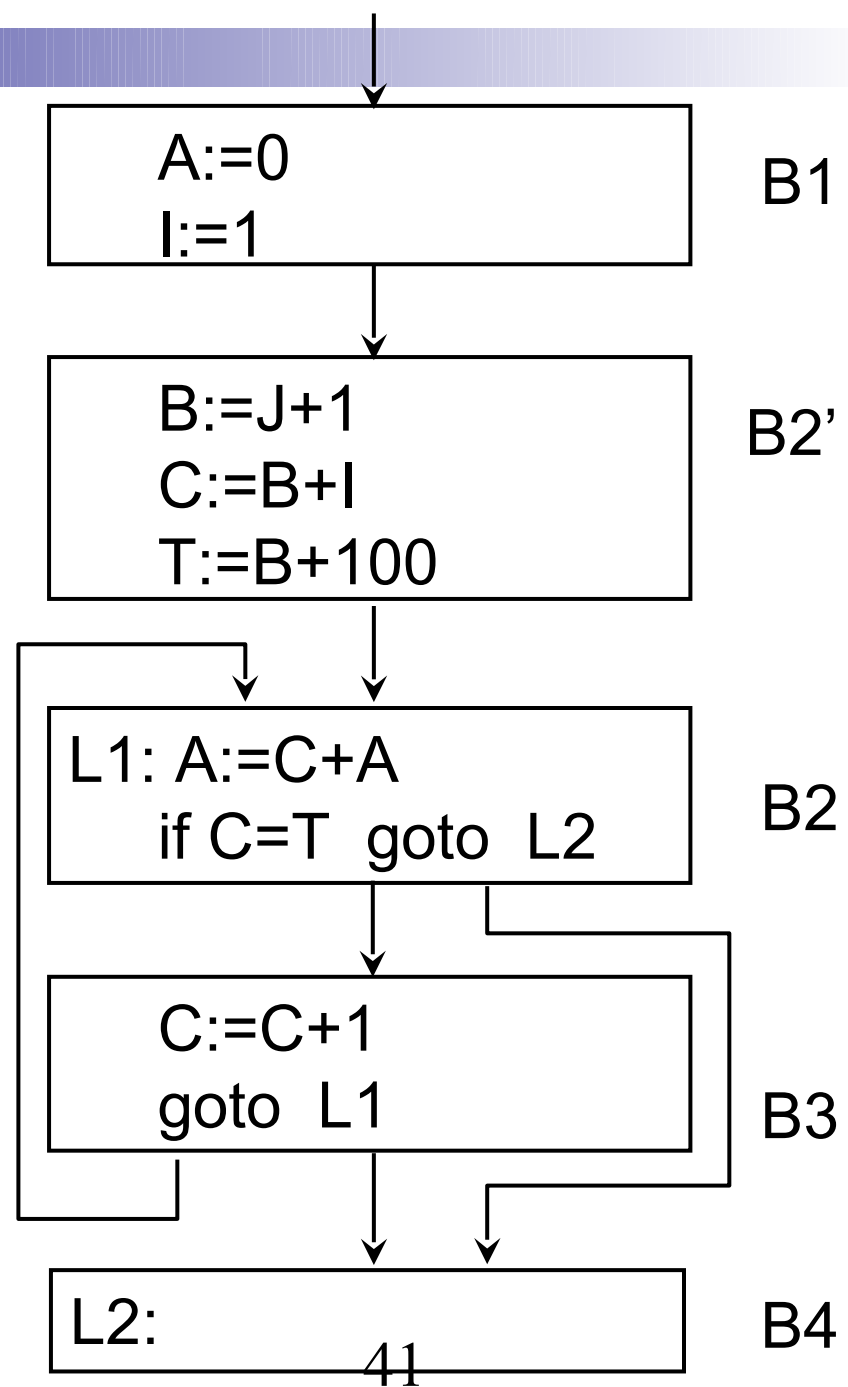
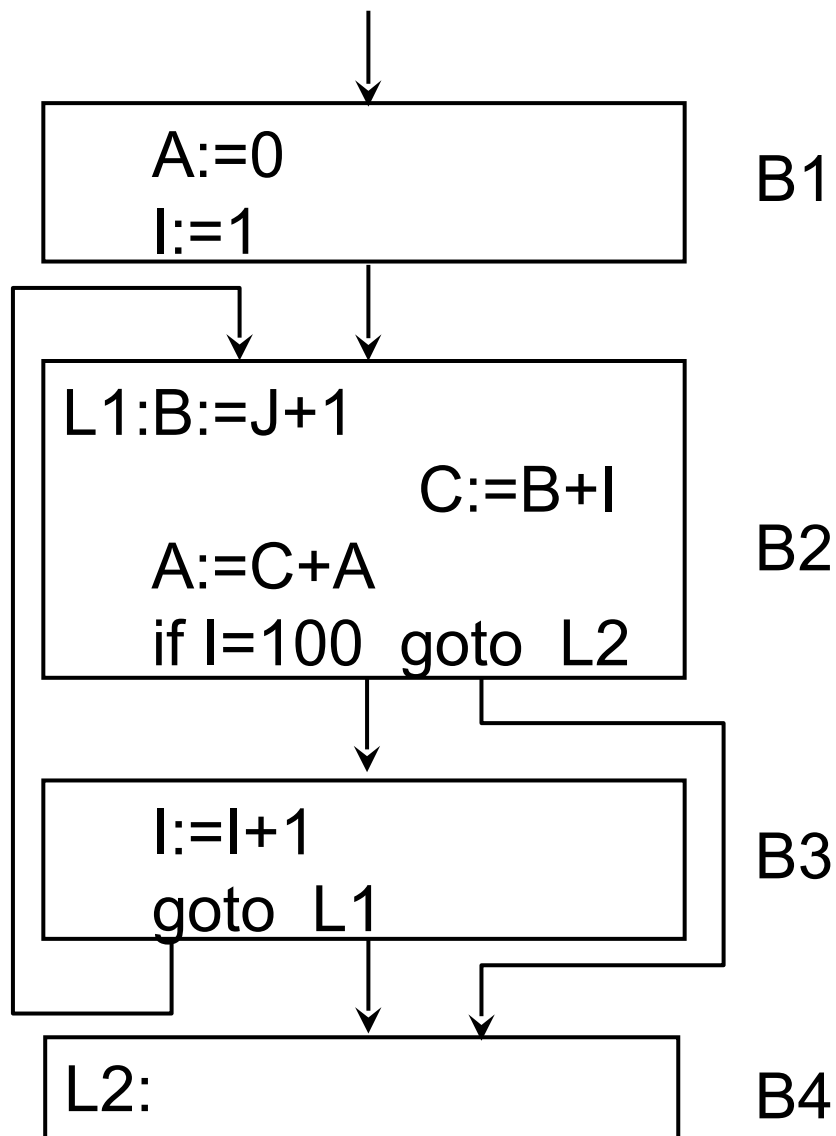
```
A:=0
I:=1
L1:  B:=J+1
     C:=B+I
     A:=C+A
     if I=100 goto L2
     I:=I+1
     goto L1
L2:
```











第十一章 代码生成

- 基本问题
- 目标机器模型
- 一个简单代码生成器

P327-1. 对以下中间代码序列 G :

$T1 := B - C$

$T2 := A * T1$

$T3 := D + 1$

$T4 := E - F$

$T5 := T3 * T4$

$W := T2 / T5$

假设可用寄存器为 R0 和 R1，W 是基本块出口的活跃变量，用简单代码生成算法生成其目标代码，同时列出代码生成过程中的寄存器描述和地址描述。

假设只有 R0 和 R1 是可用寄存器，生成的目标代码和相应的 RVALUE 和 AVALUE
UE:

中间代码	目标代码	RVALUE	AValue
T1:=B - C	LD R ₀ , B SUB R ₀ , C	R ₀ 含有 T1	T1 在 R ₀ 中
T2:=A*T1	LD R ₁ , A MUL R ₁ , R ₀	R ₀ 含有 T1 R ₁ 含有 T2	T1 在 R ₀ 中 T2 在 R ₁ 中
T3:=D + 1	LD R ₀ , D ADD R ₀ , 1	R ₀ 含有 T3 R ₁ 含有 T2	T3 在 R ₀ 中 T2 在 R ₁ 中
T4:=E-F	ST R ₁ , T2 LD R ₁ , E SUB R ₁ , F	R ₀ 含有 T3 R ₁ 含有 T4	T2 在 T2 中 T3 在 R ₀ 中 44 T4 在 R ₁ 中

假设只有 R0 和 R1 是可用寄存器，生成的目标代码和相应的 RVALUE 和 AVALUE:

中间代码	目标代码	RVALUE	AValue
T5:=T3*T4	MUL R ₀ , R ₁	R ₀ 含有 T5 R ₁ 含有 T4	T2 在 T2 中 T5 在 R ₀ 中 T4 在 R ₁ 中
W:=T2/T5	LD R ₁ , T2 DIV R ₁ , R ₀	R ₀ 含有 T5 R ₁ 含有 W	T2 在 T2 中 T5 在 R ₀ 中 W 在 R ₁ 中
	ST R ₁ , W		

小结

- 语义分析和中间代码产生
 - 翻译成四元式、构造翻译模式
 - 布尔表达式
 - 赋值语句
 - 控制语句

小结

- 运行时存储空间组织
 - 参数传递
- 优化
 - 划分基本块
 - 局部优化： DAG
 - 循环优化
- 代码生成
 - 寄存器分配