第九章　　目标代码的生成

代码生成是把语法分析优化后的中间代码变换成目标代码。目标代码一般有以下三种形式：

1、能立即执行机器语言代码（地址定位）。

2、待装配的机器语言模块（可重定位）。当需要执行时由链接装入程序把它们和某些运行程序连接起来，转换成能执行的机器语言代码。

3、汇编语言代码，尚须经过汇编程序汇编，转换成可执行的机器语言代码。

§10.1 基本块

一、基本块的定义

所谓基本块是指程序中一顺序执行的**语句（四元式）序列**，其中只有一个入口和一个出口，**入口就是第一个语句，出口就是最后一个语句。**

- 对于基本块来说，**执行只能从其入口进入，出口退出。**
- 对于一指定的程序，可以把它划分为**一系列的基本块**,在各上基本块范围内，**进行优化以及代码生成。**

二、基本块划分的算法（适用于程序和四元式，三元式序列））

1．求出程序中各个基本块的入口语句，它们是：

(1) 程序的第一个语句，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或无条件转移语句后面的语句.

2．对以上求出的每一入口语句，构造其所属的代码块

（即找出口语句）由该入口语句到：

(1) 下一入口语句（不包括该入口语句）或到

(2) 转移语句（包括该转移语句）或到

(3) 停语句（包括该停语句）之间的语句序列组成。

3．凡未被纳入某一基本块的语句，都是程序中控制流无法到达的语句，可从程序中删除。

例： for I:=1 to N do M:=N+I;

入口> 100 (:=, 1, _ , I)

出口< 101 (J, _ , _ , 103)

< > 102 (+, I, 1 , I)

< > 103 (J<=, I, N , 105)

< > 104 (J, _, _ , 108)

> 105 (+, M, I , T)

106 (:=, T, _ , M)

< 107 (J, _, _ , 102)

> 108

§10.2 各类四元式的翻译方法

中间代码（四元式）翻译为类8086汇编指令

1、四元式（Program, Prog_id,_,_）表示主程序的开始

对应以下汇编代码

MAIN SEGMENT:

ASSUME CS：MAIN，DS：MAIN，ES：MAIN

其中：MAIN表示主程序段，程序名未使用。

ASSUME伪指令用于指定本程序所使用的各段的开始地址，CS，DS，ES各段的开始地址都与主程序一致。

2、四元式（sys，—，—，—）表示程序结束（end）

它的作用是终止程序的运行返回OS。

3、四元式（+，A，B，T）对应的指令为：

MOV AX，A；

ADD AX，B；

MOV T，AX；

其余二元操作四元式的翻译与此类拟；

4、四元式（=，B，—，A）对应的指令为：

MOV AX，B；

MOV A，AX；

5、四元式（jnZ，A，—，P）对应的指令为

MOV AX，A；

CMP AX,0;

JNZ P’；

9、四元式（J，—，—，P）对应的指令为：

JMP P’；

10、四元式（Jrop，A，B，P）对应的指令为

MOV AX，A；

CMP AX，B；

Jrop P’；

四元式目标地址的确定：在四元式中增加栏目

(1) 在把四元式翻译为目标指令时，是按四元式表中四元式的顺序依次执行，为计算每个四元式所对应的目标地址，必须累计每条指令的字节数。到达某四元式时，所累计的字节数就是该四元式的目标地址（相对的）。

(2) 转移语句中转移地址所指的四元式是基本块的入口语句，在划分基本块时基本块的入口语句，把这种语句按出现的顺序生成汇编代码标号L_1，L_2..... 等，并用一张标号表存放起来,表格内容为四元式序号与对应汇编代码标号。

§3、寄存器分配与基本块代码的生成

本节主要讨论如何生成每个基本块的较优代码。

（1）较优的标准有两条：

一是总的指令条数要少

二是尽量少使用访问内存指令

（2）合理利用寄存器需解决两个问题：

一是尽可能把后面还要使用的变量仍保存在寄存器中.

二是应把不再使用的变量所占用的寄存器及时释放掉, 为此需引入两个概念： 基本块内变量的引用信息和活跃信息。

一、引用信息与活跃信息

1. 活跃信息与引用信息（活跃变量）：

从整个程序范围内，如果变量A在某点P的值（P是四元式的编号）在从P可达的某四元式q中被引用，则称变量A在P点是**活跃的**，并称四元式q是变量A的**引用信息**。

(p) A=B+C;

(r) E=A+F;

…..

(q) D=A+V;

A在P点是活跃的

q是变量A的引用信息

r是变量A的引用信息

假定

①所有的非临时变量都看作是出基本块后的活跃变量

②所有的临时变量均看作是出基本块后的非活跃变量。

二、建立基本块内的引用信息链和活跃信息链

1. 数据结构：

为四元式中的每个变量设置引用信息栏和活跃信息栏

在符号表中也增设引用信息栏和活跃信息栏，用于暂存各变量的引用与活跃信息，临时变量也如此。

例：四元式的引用信息与活跃信息表示法

序号	四元式	结果	左变量	右变量
i	(op, B,C,A)	(引用 活跃)	(引用 活跃)	(引用 活跃)
i+1	(op,A,C,T)	(引用 活跃)	(引用 活跃)	(引用 活跃)

符号表

名称	类型			引用/活跃	
A				(引用 活)	
B				(引用 活)	

2．算法：

(1) 从基本块出口向入口方向逐个扫描每个四元式，根据每个变量的引用和定值情况建立各变量的引用和活跃信息链。

(2) 用（U，L）表示引用（U）与活跃（L）信息。

用N表示不引用或不活跃。

用Y表示活跃，四元式编号表示下一个引用位置

(3) 用FIRST和LAST分别表示基本块的入口、出口语句号。

(4) 计算基本块各变量引用信息和算法NEXT-USE如下（设四元式（OP，B，C，A））

Procedure NEXT-USE（FIRST，LAST）；

把符号表中每个变量的引用栏初始化为N，

并把每个变量是否活跃的标记填入符号表的活跃栏内；

for I= LAST to FIRST do

begin GET(QUAD)　　(i) A=B+C

把符号表中A的（U，L）附加到四元式中的A上；

让符号表中A的（U，L）=（‘N’，‘N’）；

把符号表中B和C的（U，L）附加到四元式中的B和C上；

让符号表中B的（U，L）=（i，‘Y’）；

让符号表中C的（U，L）=（i，‘Y’）；

end

end;

例如：对以下基本块，执行算法：

序号	四元式	结果	左变量	右变量
1	（—，A, B, T)	(5,Y)	(2,Y)	(2,Y)
2	(+ ，A, B, A)	(3,Y)	(N,N)	(5,Y)
3	(—，A, C, U)	(6,Y)	(N,Y)	(4,Y)
4	(+ , C , D, V)	(N,N)	(N,Y)	(N,Y)
5	(+, T, B, V)	(6,Y)	(N,N)	(N,Y)
6	(+, V, U, W)	(N,Y)	(N,Y)	(N,Y)

例如：

6(+,V,U,W)将W，V，U符号表的内容填四元式附加信息，而后符号表

W→ (N，N)

V→ (6，Y)

U→ (6，Y)

5(+,T，B，V）将V，T，B序号表的内容填四元式附加信息，而后符号表

V→ (N，N)

T→ (5，Y)

B→ (5，Y)

4（+,C,D,V）将V,C,D符号表的内容填四元式附加信息

而后符号表 V→（N，N）

C→（4，Y）

D→（4，Y）

3（－,A,C,V）将V,A,C符号表的内容填四元式附加信息

而后符号表 V→（N，N）

A→（3，Y）

C→（3，Y）

2（+，A，B，A）将A符号表内容填结果栏四元式，

而后符号表　A→（N，N）

符号表内容填四元式（左右变量）

A→（2，Y）

B→（2，Y）

1（－,A,B,T）将T，A，B符号表内容填四元式，

而后符号表　T→（N，N）

A→（1，Y）

B→（1，Y）

符号表的变化情况

A	（N，Y）→（3，Y）→（N，N）→（2，Y）→（1，Y）
B	（N，Y）→（5,Y）→（2，Y）→（1，Y）
T	（N，N）→（5，Y）→（N，N）
C	（N，Y）→（4，Y）→（3，Y）
U	（N，Y）→（6，Y）→（N，N）
D	（N，Y）→（4，Y）
V	（N，Y）→（6，Y）→（N，N）→（N，N）
W	（N，Y）→（N，N）

例如：对以下基本块，执行算法：

序号	四元式	结果	左变量	右变量
1	（－，A, B, T)	(5,Y)	(2,Y)	(2,Y)
2	(+ ，A, B, A)	(3,Y)	(N,N)	(5,Y)
3	(－，A, C, U)	(6,Y)	(N,Y)	(4,Y)
4	(+ , C , D, V)	(N,N)	(N,Y)	(N,Y)
5	(+, T, B, V)	(6,Y)	(N,N)	(N,Y)
6	(+, V, U, W)	(N,Y)	(N,Y)	(N,Y)

三、寄存器的分配问题：变量多寄存器少时

1、描述寄存器分配信息的数据结构：

VAR栏表示把寄存器分配给哪个变量或哪些变量

（复写）（A：=B）

MEM栏表示占用R的 哪几个变量的值又在内存中

2、寄存器分配信息的描述结构

R	VAR	MEM
AX		
BX		
CX		
DX		

3、寄存器分配算法

一般在生成器四元式A：=B OP C的代码中，通常把左操作数B取到寄存器R中，再和C操作（C可在内存中，也可在寄存中），R中的结果就是A的值，或者说A占用了R.

GETREG（QUAD，R）：

QUAD是待分配寄存器的四元式： i: A:=B OP C

i:(OP,B,C,A)

R是分配的寄存器。

(算法中需查看附在四元式i上的活跃与引用信息及上表中的结构)

为变量A分配寄存器的算法应遵循以下原则：

（1）如果B已在寄存器R_i中（VAR（R_i）),且以后不再引用（查四元式i的附加信息，B（N，N）为“非活跃”和“非引用”），则选择R_i.(因为最后R_i中存的是A)。

（2）从空闲寄存器中选一R_i。

（3）从已分配寄存器中选取一个R_i，查MEM（R_i）条件是：

- 占用R_i的变量，其值同时也在其内存中；
- 占用R_i的变量，其值虽未在内存中，其值将在最远的将来才会使用（查四元式的附加信息）；

生成一条存数指令（MOV　M，R_i）把R_i中内容存入M单元。

四、基本块的代码生成算法

算法中使用以下过程。

(1) ADDR（B）：获得变量B当前存放的地方，只要B在寄存器中就返回R，否则B在内存中。

(2) FILL（B，R）：如果变量B不在VAR（R）中，则填入；如果B同时又在内存中，则把B填入MEM（R）中。

(3) EMIT（OP R，X）：向目标文件QBJ中输出一条指令OP R，X。

(4) DELETE（B，R）：如果B在VAR（R）和MEM（R）中的话，则删除其中的B。

算法GENOBJ的一般描述如下：

输入：QUAD中的四元式 i： A=B OP C

输出：四元式I的目标代码，并将其存入目标文件OBJ

```
PROCEDURE GENOBJ(QUAD);
BEGIN /*QUAD中的四元式形成I：A=B OP C */
GETREG（QUAD,R）
  IF ADDR（B）≠R THEN
     BEGIN EMIT (MOV R, ADDR(B));
                EMIT(OP  R, ADDR(C));
     END
  ELSE
    BEGIN EMIT (OP R, ADDR(C));
          DELETE( B, R);
    END
```

FILL(A,R);

for 每个$R_k \neq R$ do DELETE(A,R_k);

for 每个R_k do

begin if B(i)=(N,N) then Delete (B,R_k)

if C (i)=(N ,N) then Delete (C,R_k)

end

end

算法中第一个for语句保证A只占有R；

第二个for语句释放非活跃跃和非引用变量B与C所占用的寄存器。

例：对以下四元式利用GENOBJ产生目标代码

假定只有AX和BX两个寄存器可用。

四元式	AX	BX	QBJ
(1) (-, A, B, T)	T		(1) MOV AX, A;
			(2) SUB AX, B;
(2) (+, A, B, A)		A	(3) MOV BX, A;
			(4) ADD BX, B;
(3) (- , A, C, U)		U	(5) MOV A, BX;
			(6) SUB BX , C
(4) (+, C, D, V)		V	(7) MOV U, BX;
			(8) MOV BX, C;
			(9) ADD BX, D;

四元式		AX	BX	QBJ
(5)	(+, T, B, V)	V	空	(11) ADD AX, B;
(6)	(+, V, U, W)		W	(12) MOV BX, AX;
				(13) ADD BX, U;
				(14) MOV W, BX;
				(15) MOV V, AX;

对算法GENOBJ还可进一步改进，如果按照程序控制流程图去生成每一个基本块的目标代码，那么在出基本块的时候，可以把该基本块的寄存器占用情况保留到下一基本块。在下一基本块中应尽量利用在寄存器的变量值，这样在每个基本块出口时，可以不存贮每个活跃变量的值。

利用上术基本块的目标代码生成算法，按此种顺序所产生的目标代码，其有效性未必是最佳的，我们还可以利用基本块的DGA表示来重构基本块以改变基本块的运行次序，从而得到更有效的目标代码。

§4、基本块的DAG表示及DAG的目标代码生成

一、基本块的DAG（无回路有向图）表示

(1)（+，A，B，T_1）

T_1-R_0

(2)（+，C，D，T_2）

T_2-R_1

(3)（+，E，T_2，T_3）

T_3-R_0 (R_0->T_1)

(4)（+，T_1，T_3，F）

F-R_1 (T_1->R_1)

(1)（+，C，D，T_2）

T_2-R_0

(2)（+，E，T_2，T_3）

T_3-R_1

(3)（+，A，B，T_1）

T_1-R_0

(4)（+，T_1，T_3，F）

F-R_0

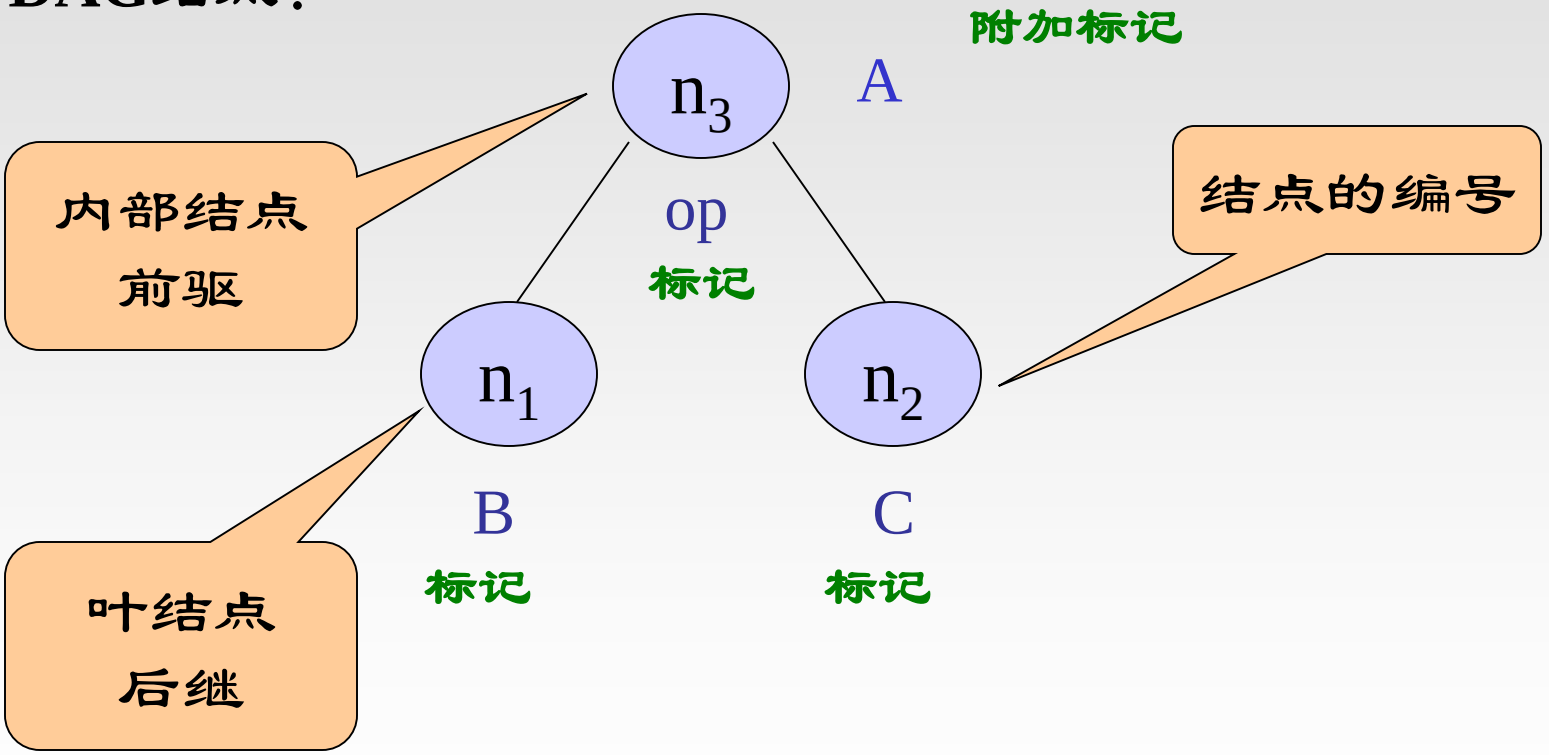
目标代码更有效

1、由基本块构造DAG的算法

删除公共子表达式 ；删除无用赋值

四元式（op , B, C , A)　　假定B,C均为非常数

DAG结点：



1、由基本块构造DAG的算法

假设DAG各结点信息将用某种适当的数据结构来存放（例如：链表），并设有一个标识符与结点的对表；NODE（A）是描述这种对应关系的函数。

设过程NODE(A)= null　　DAG上无标记或附加标记为A的结点

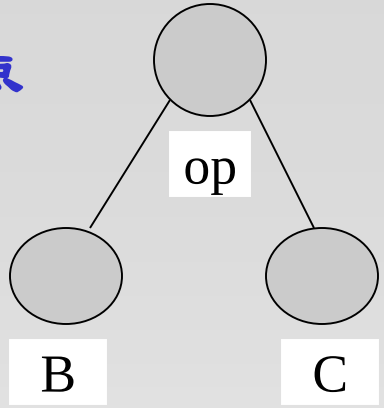
n_i　　DAG上有标记或附加标记为A的结点n_i

1) 对基本块的每一条四元式 A:=B op C　执行算法

（1）若NODE(B)=null，则建立NODE(B)=n_i 否则

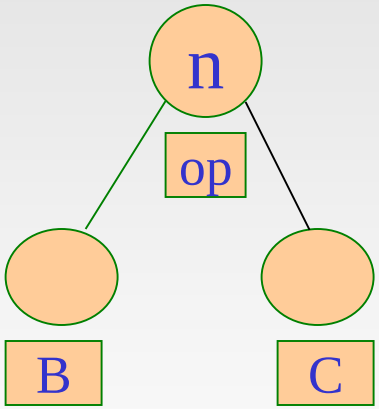
（2）若NODE(C)=null，则建立NODE(B)=n_j 否则

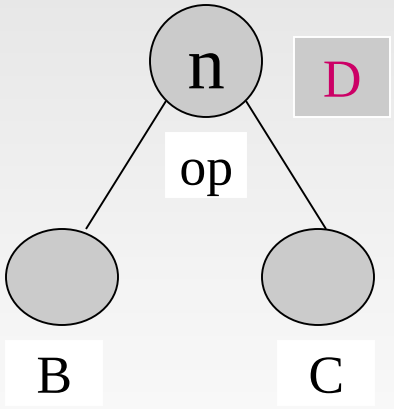
(3) 查找DAG中有无结点



无：建立结点n

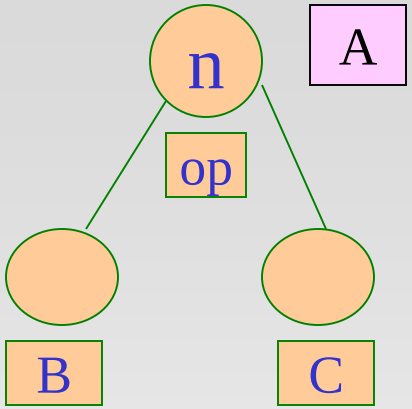
有：设此结点为n





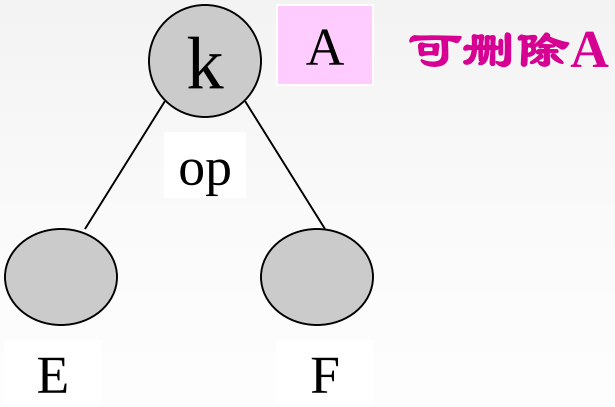
查公共子表达式

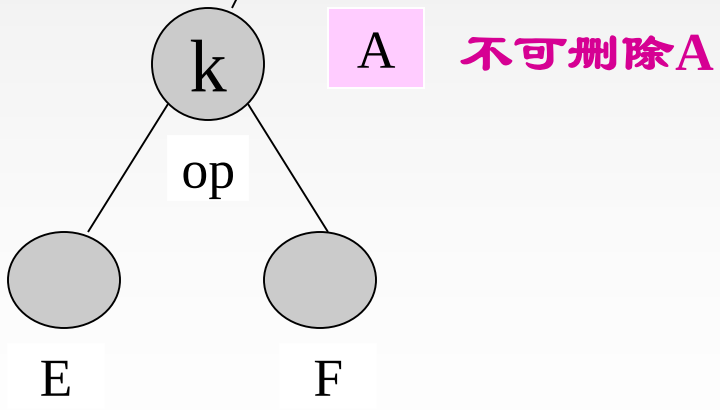
（4）若NODE(A)=null，则令NODE(A)=n　否则



（5）若NODE(A)=K，且NODE(A) 不是叶，无前驱

则从k上删去A；　　令NODE（A)=n



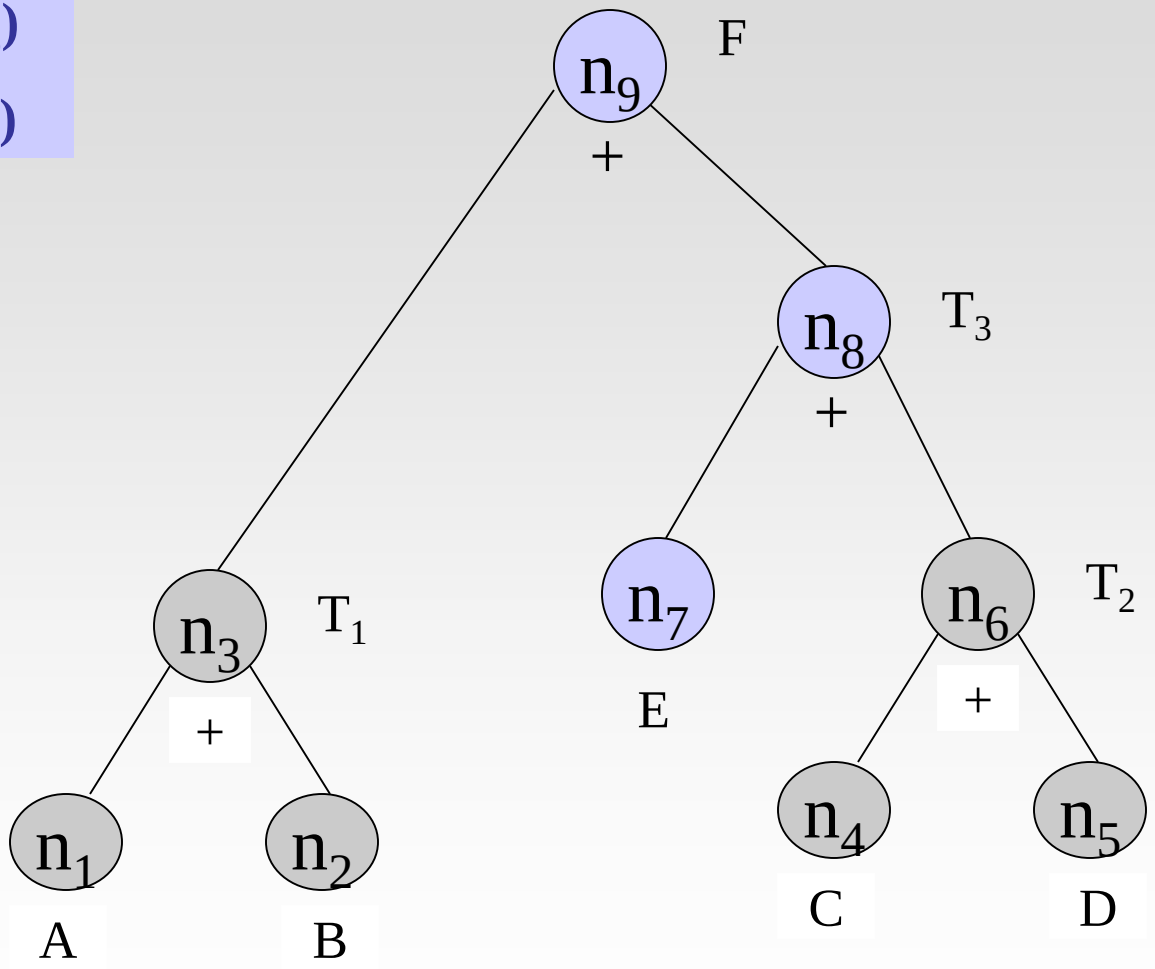


(1) （+， A , B , T_1)

(2) （+， C , D , T_2)

(3) （+， E , T_2 , T_3)

(4) （+， T_1 , T_3 , F)



2）DAG中结点的重构算法

设DAG中有N个内部结点，T是一个线形表，共N项

```
For  k:=1  to  N  Do   T[k]:=null;
       I:=N;
  while 存在未列入T的内部结点 do
   begin 选一未列入T但其全部前驱均列入T或无前驱的内部结点 n;
        T[I]:=N;  I:=I-1;                /*把n列入T中*/
       while   n的最左子结点m不为叶结点且其全部前驱入T中   do
        begin  T[I]:=m; I:=I-1;
                  n:=m
        end
    end
```

(1) (+, A, B, T_1)

(2) (+, C, D, T_2)

(3) (+, E, T_2, T_3)

(4) (+, T_1, T_3, F)

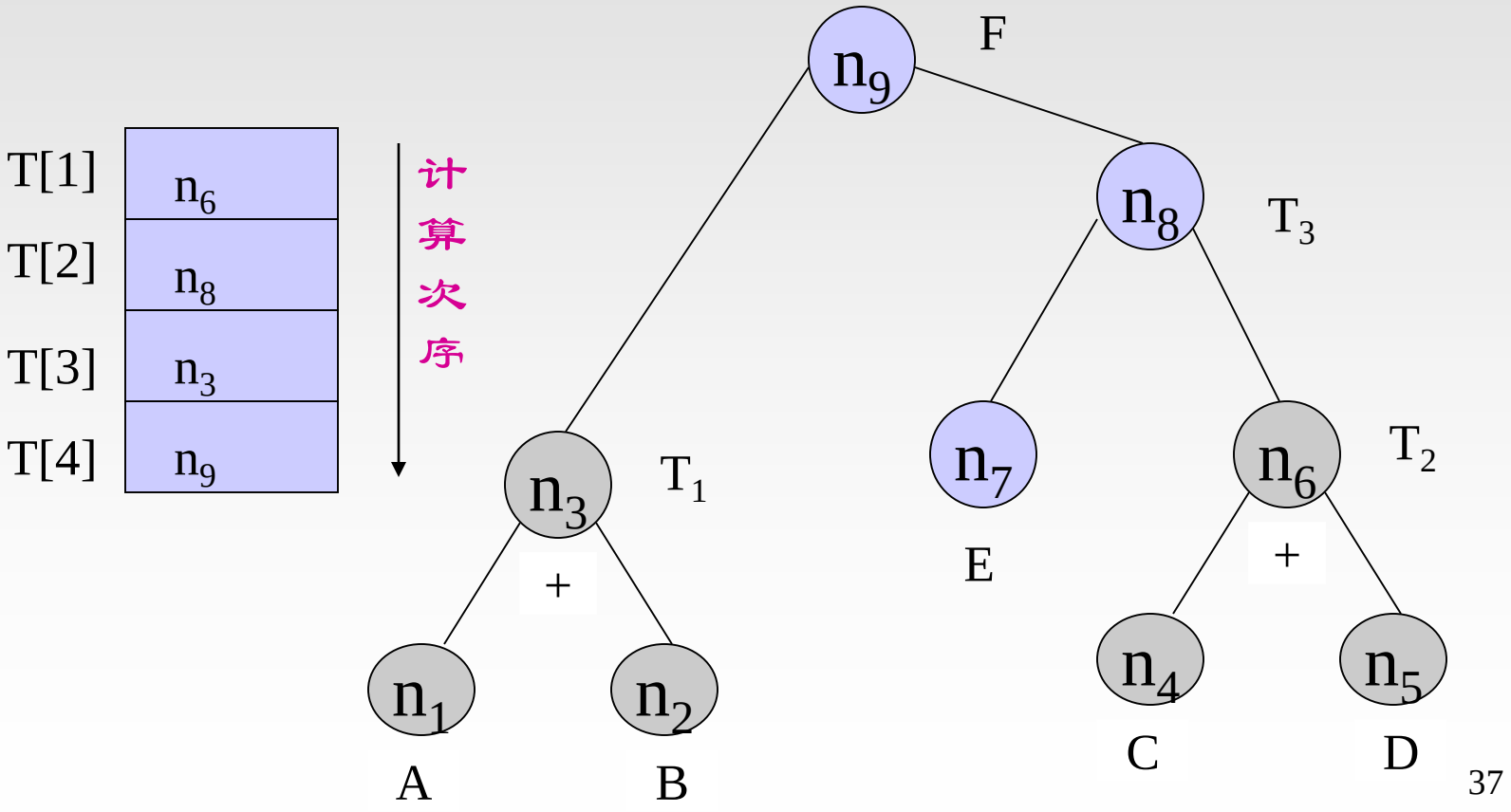
DAG重构

(1) (+, C, D, T_2)

(2) (+, E, T_2, T_3)

(3) (+, A, B, T_1)

(4) (+, T_1, T_3, F)



例：对以下四元式利用GENOBJ产生目标代码

假定只有AX和BX两个寄存器可用。(前例的另一法)

四元式	AX	BX	QBJ
(1) (-, A, B, T)	T		(1) MOV AX, A;
			(2) SUB AX, B;
(2) (+, A, B, A)		A	(3) MOV BX, A;
			(4) ADD BX, B;
(3) (- , A, C, U)	U		(5) MOV T,AX
			(6) MOV AX, BX;
			(7) SUB AX , C
(4) (+, C, D, V)		V	(8) MOV A, BX;
			(9) MOV BX, C;
			(10) ADD BX, D;

四元式	AX BX	QBJ
（5）（+，T，B，V）	空V	（11）MOV BX, T;
		(12) ADD BX，B;
（6）（+，V，U，W）	W	(13）MOV V,BX;
		（14）ADD BX，AX;
		（15）MOV U，AX;
		（16）MOV W，BX;