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》
中间代码生成 I

计算机科学与技术学院

李 诚

12/11/2018

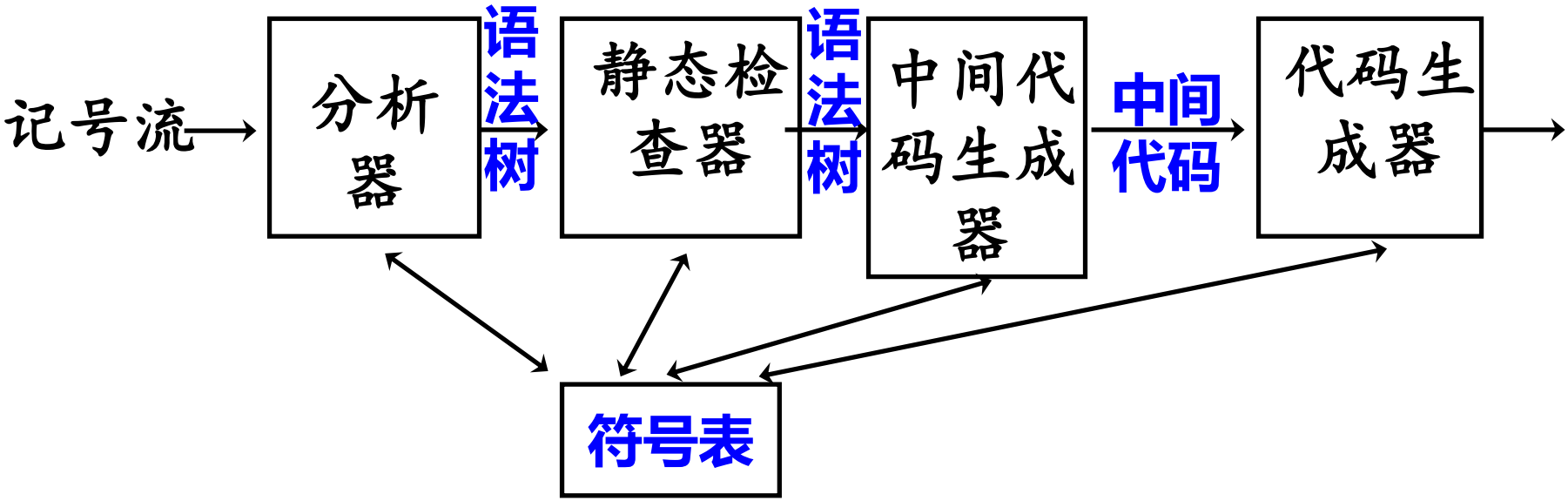
关于课程实验

- 目标：为PL0语言实现一个简单的编译器
  - **Project 1：** 词法分析
  - **Project 2：** 语法分析
  - **Project 3：** 语法错误处理+对前两个project的扩展，11.15 release，11.30提交
  - **Project 4：** 代码生成，12.1 release，12.15提交
  - 项目结束后，需要进行答辩，每一组准备ppt，每一名同学都要汇报
    - 自己做了什么
    - 学到了什么
    - 对课程和实验的意见与建议

通知

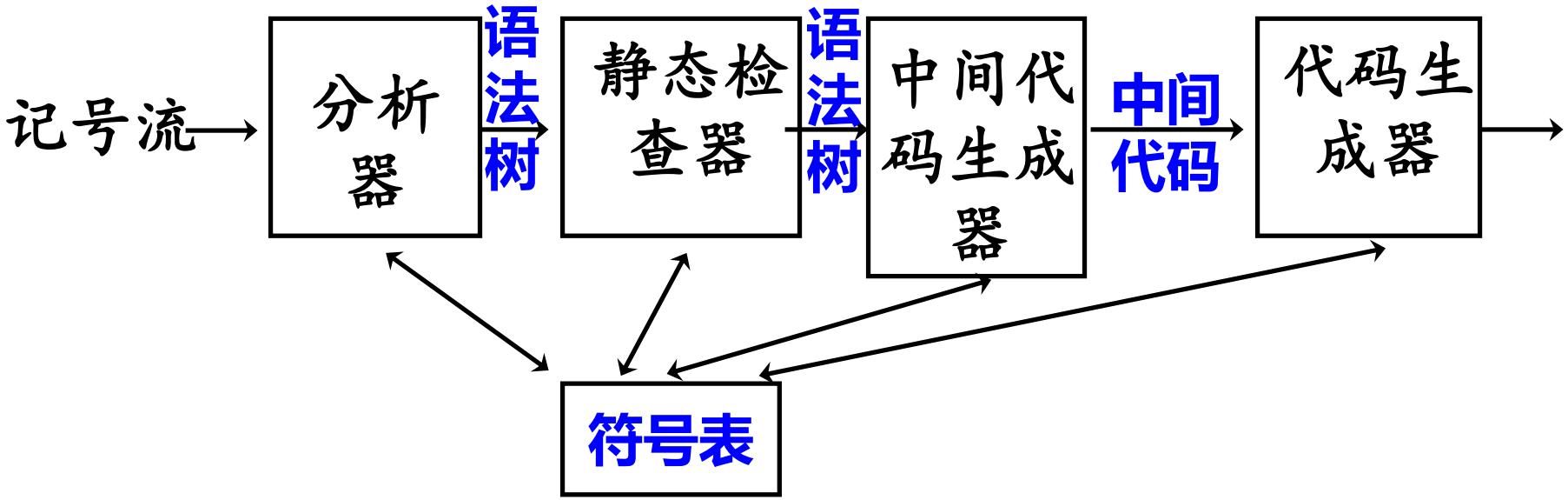
- 期中考试的成绩在本周之内公布。
- 由于老师出差，周四的课取消，后面补回来。
- 周四的上机课正常进行，助教会发布新的项目内容，并对之前项目的完成情况进行说明，请大家尽量都去。

主要内容



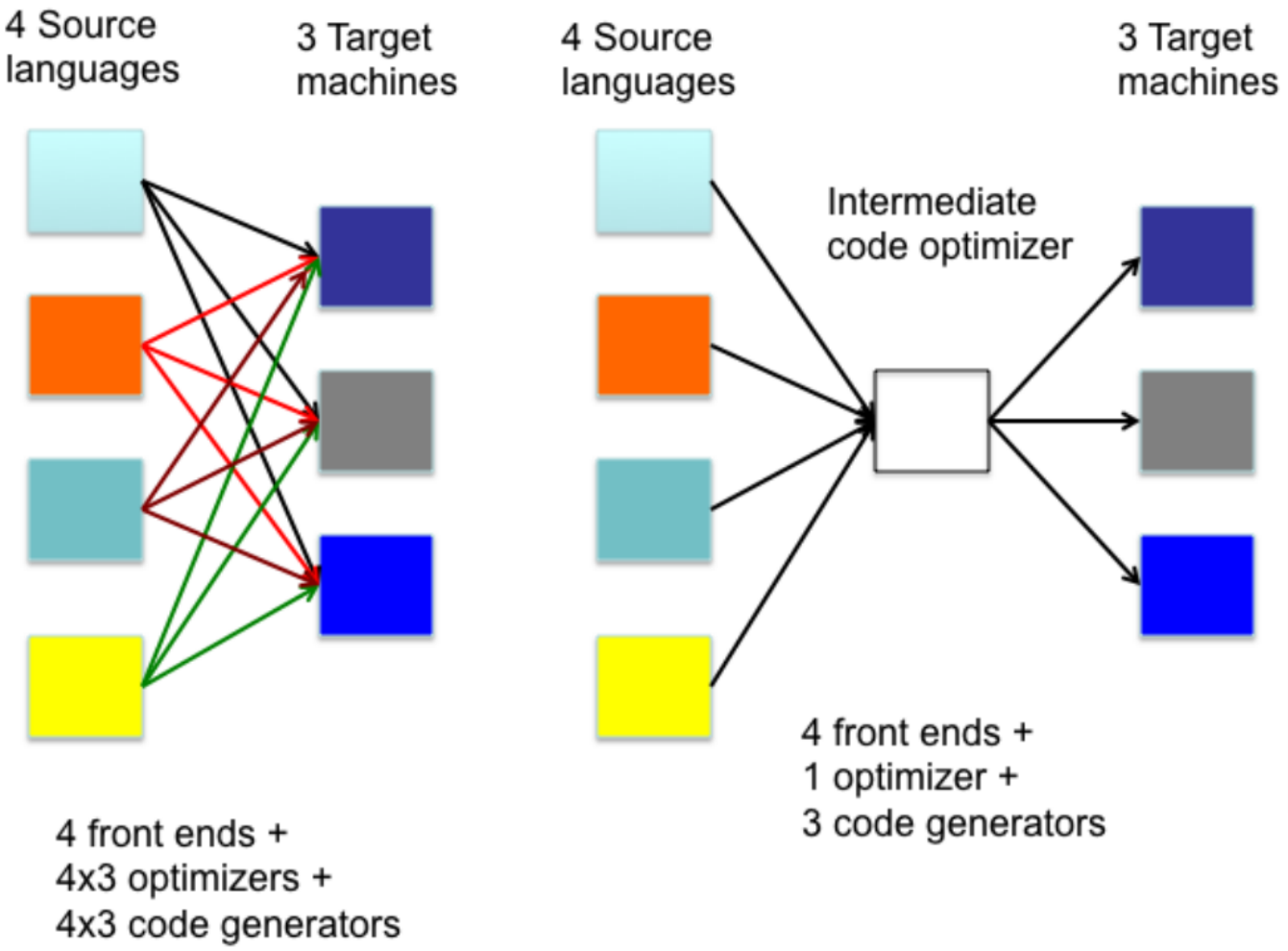
- **中间语言(Intermediate Representation)**
  - 后缀表达式、图表示、三地址码、静态单赋值
- **中间代码生成**
  - 声明语句(更新符号表)
  - 表达式、赋值语句(产生临时变量、查询符号表)
  - 布尔表达式、控制流语句(标号/回填、短路计算)

主要内容



- **中间语言(Intermediate Representation)**
  - 后缀表达式、图表示、三地址码、静态单赋值
- **中间代码生成**
  - 声明语句(更新符号表)
  - 表达式、赋值语句(产生临时变量、查询符号表)
  - 布尔表达式、控制流语句(标号/回填、短路计算)

为什么需要中间语言?



实践过程中，推陈出新的语言、不断涌现的指令集、开发成本之间的权衡

后缀表示

*uop*是一元运算符

$$E \rightarrow E\ opE \mid uopE \mid (E) \mid \mathbf{id} \mid \mathbf{num}$$

表达式E	后缀式E'
id	id
num	num
$E_1\ op\ E_2$	$E_1'\ E_2'\ op$
$uopE$	$E'uop$
(E)	E'

后缀表示

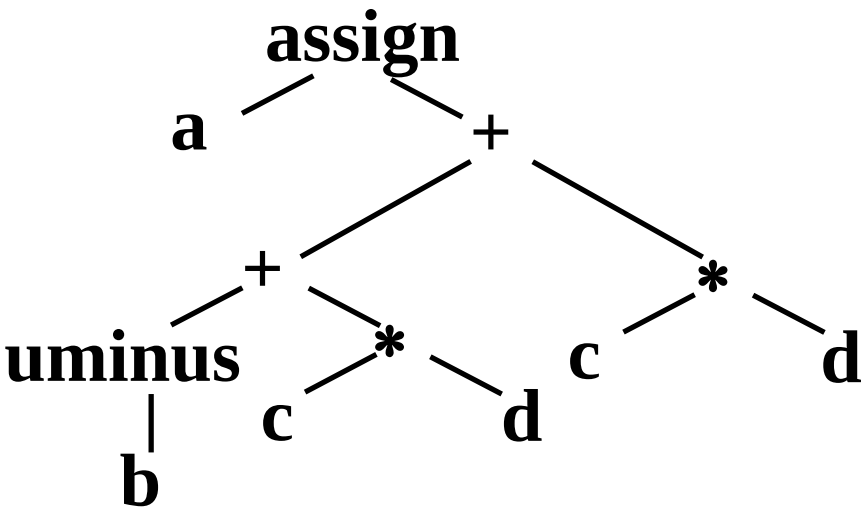
- 后缀表示不需要括号
  - (8 – 5) + 2 的后缀表示是8 5 –2 +
- 后缀表示的最大优点是便于计算机处理表达式

计算栈	输入串
	8 5 –2 +
8	5 –2 +
8 5	–2 +
3	2 +
3 2	+
5	

后缀表示

- 后缀表示不需要括号
  - (8 – 5) + 2 的后缀表示是8 5 –2 +
- 后缀表示的最大优点是便于计算机处理表达式
- 后缀表示的表达能力
  - 可以拓广到表示赋值语句和控制语句
  - 但很难用栈来描述控制语句的计算

- 语法树是一种图形化的中间表示

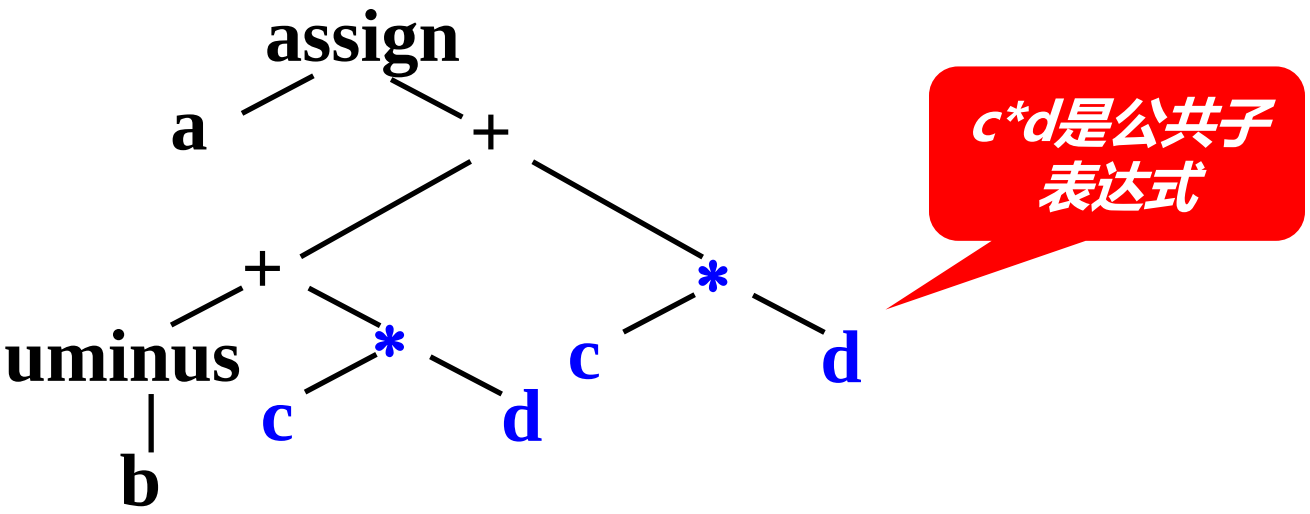


(a) 语法树

a = (–b + c*d) + c*d的图形表示

图形表示

❑语法树是一种图形化的中间表示

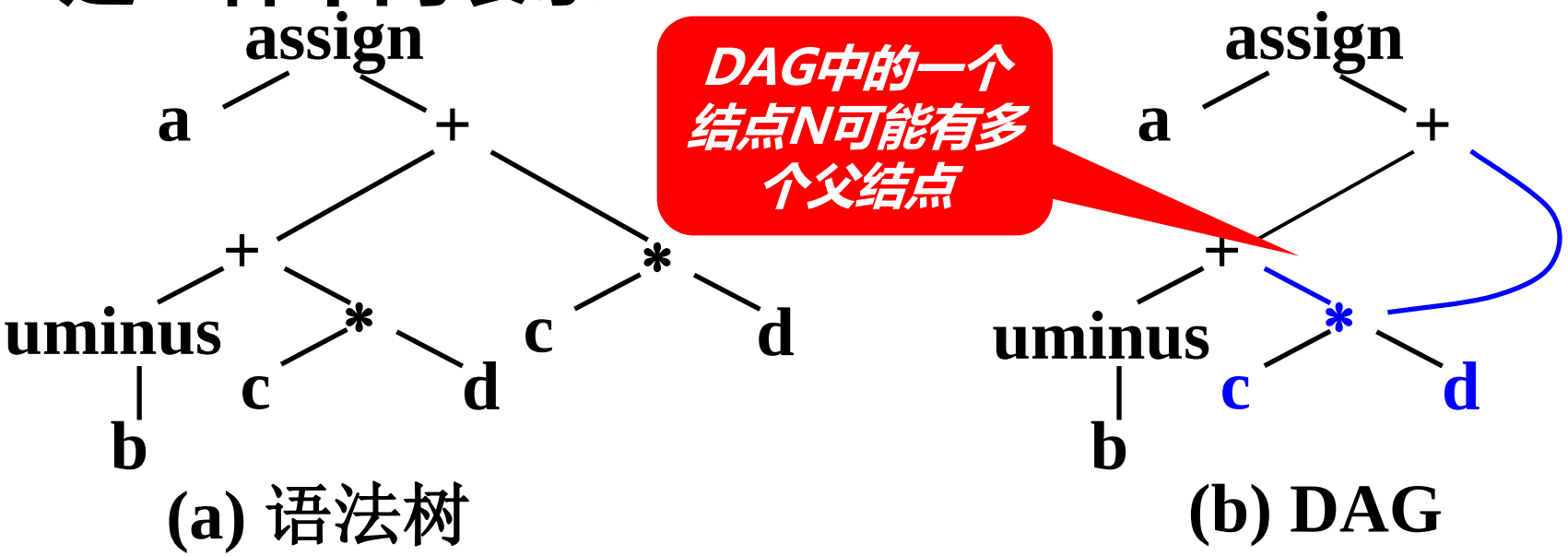


(a) 语法树

a = (–b + c*d) + c*d的图形表示

图形表示

- 语法树是一种图形化的中间表示
- 有向无环图(Directed Acyclic Graph, DAG)也是一种中间表示



a = (–b + c*d) + c*d的图形表示

构造赋值语句语法树的语法制导定义(第四章内容)

修改构造结点的函数可生成有向无环图

产 生 式	语 义 规 则
$S \rightarrow \text{id} = E$	$S.nptr = mkNode(\text{'assign'}, mkLeaf(\text{id}, \text{id}.entry), E.nptr)$
$E \rightarrow E_1 + E_2$	$E.nptr = mkNode(\text{'+'}, E_1.nptr, E_2.nptr)$
$E \rightarrow E_1 * E_2$	$E.nptr = mkNode(\text{'*'}, E_1.nptr, E_2.nptr)$
$E \rightarrow -E_1$	$E.nptr = mkUNode(\text{'uminus'}, E_1.nptr)$
$E \rightarrow (E_1)$	$E.nptr = E_1.nptr$
$F \rightarrow \text{id}$	$E.nptr = mkLeaf(\text{id}, \text{id}.entry)$

❑三地址代码 (three-address code)

一般形式：$x = y\ op\ z$

- 最多一个算符
- 最多三个计算分量
- 每一个分量代表一个地址，因此三地址

❑ 例 表达式$x + y * z$翻译成的三地址语句序列

$t_1 = y * z$

$t_2 = x + t_1$

❑三地址代码是语法树或DAG的一种线性表示

❖例　　a = (–b + c*d) + c*d

语法树的代码

$t_1 = -b$

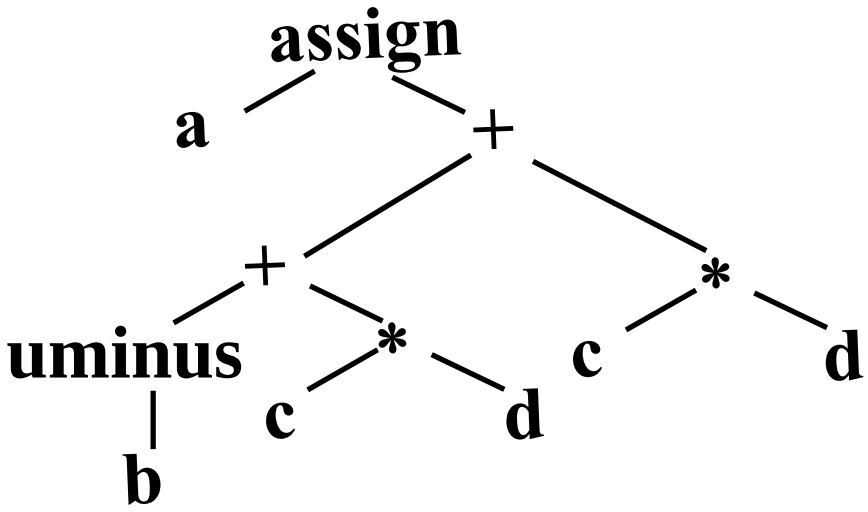
$t_2 = c * d$

$t_3 = t_1 + t_2$

$t_4 = c * d$

$t_5 = t_3 + t_4$

$a = t_5$



❑三地址代码是语法树或DAG的一种线性表示

❖例　　a = (–b + c*d) + c*d

语法树的代码

$t_1 = -b$

$t_2 = c * d$

$t_3 = t_1 + t_2$

$t_4 = c * d$

$t_5 = t_3 + t_4$

$a = t_5$

DAG的代码

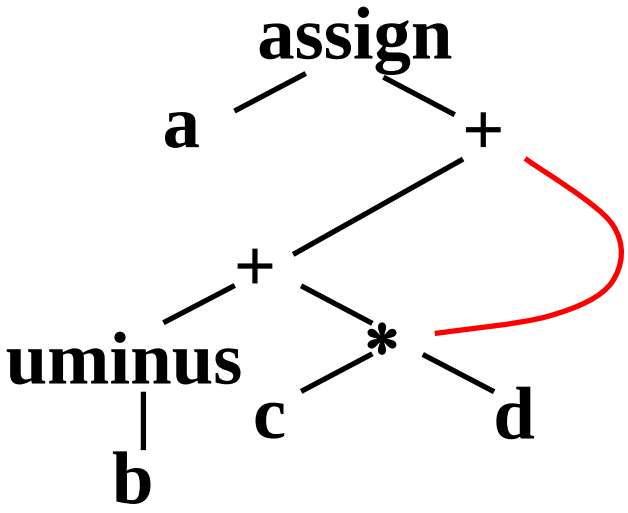
$t_1 = -b$

$t_2 = c * d$

$t_3 = t_1 + t_2$

$t_4 = t_3 + t_2$

$a = t_4$



❑常用的三地址语句

- ❖赋值语句 $x = y\ op\ z$， $x = op\ y$， $x = y$
- ❖无条件转移 **goto L**
- ❖条件转移 **if** x ***relop*** y **goto L**
- ❖过程调用 **param** x 和 **call p ,** n
- ❖过程返回 **return** y
- ❖索引赋值 $x = y[i]$ 和 $x[i] = y$
- ❖地址和指针赋值 $x = \&y$， $x = *y$ 和 $*x = y$

三地址代码的实现方式

- 三地址代码只说明了指令的组成部分，为描述其在编译器中的具体数据结构实现
- 常见的实现方式有三种：
  - 四元式：(op, arg1, arg2, result)
  - 三元式：(op, arg1, arg2)
  - 间接三元式：(三元式的指针表)

三地址代码的实现方式

❑四元式(Quadruple)

❖例： a = b * – c + b * – c

#	Op	Arg1	Arg2	Result
(0)	uminus	c		t1
(1)	*	t1	b	t2
(2)	uminus	c		t3
(3)	*	t3	b	t5
(4)	+	t2	t4	t5
(5)	=	t5		a

缺点：临时变量太多，增加时间和空间成本

三地址代码的实现方式

❑三元式(Triple)

❖例：a = b * – c + b * – c

#	Op	Arg1	Arg2
(0)	uminus	c	
(1)	*	(0)	b
(2)	uminus	c	
(3)	*	(2)	b
(4)	+	(1)	(3)
(5)	=	a	(4)

缺点：隐式的临时变量，代码位置调整会造成引用该位置的代码也要修改。

三地址代码的实现方式

❑间接三元式(Indirect triple)

❖例：a = b * – c + b * – c

指令序列可以任意调整顺序

List of pointers to table

#	Op	Arg1	Arg2
(14)	+	x	y
(15)	+	y	z
(16)	*	(14)	(15)
(17)	+	(14)	z
(18)	+	(16)	(17)

#	Statement
(1)	(14)
(2)	(15)
(3)	(16)
(4)	(17)
(5)	(18)

优势：比四元式空间开销小，比三元式更灵活

三地址代码的实现方式总结

四元式	按编号次序计算	计算结果存于**result**	方便移动，计算次序容易调整	大量引入临时变量
三元式	按编号次序计算	由编号代表	不方便移动	在代码生成时进行临时变量的分配
间接三元式	按编号次序计算		方便移动，计算次序容易调整	在代码生成时进行临时变量的分配

静态单赋值形式

- 一种便于某些代码优化的中间表示
- 和三地址代码的主要区别
  - 所有赋值指令都是对不同名字的变量的赋值

三地址代码	静态单赋值形式
$p = a + b$	$p_1 = a + b$
$q = p - c$	$q_1 = p_1 - c$
$p = q * d$	$p_2 = q_1 * d$
$p = e - p$	$p_3 = e - p_2$
$q = p + q$	$q_2 = p_3 + q_1$

静态单赋值形式

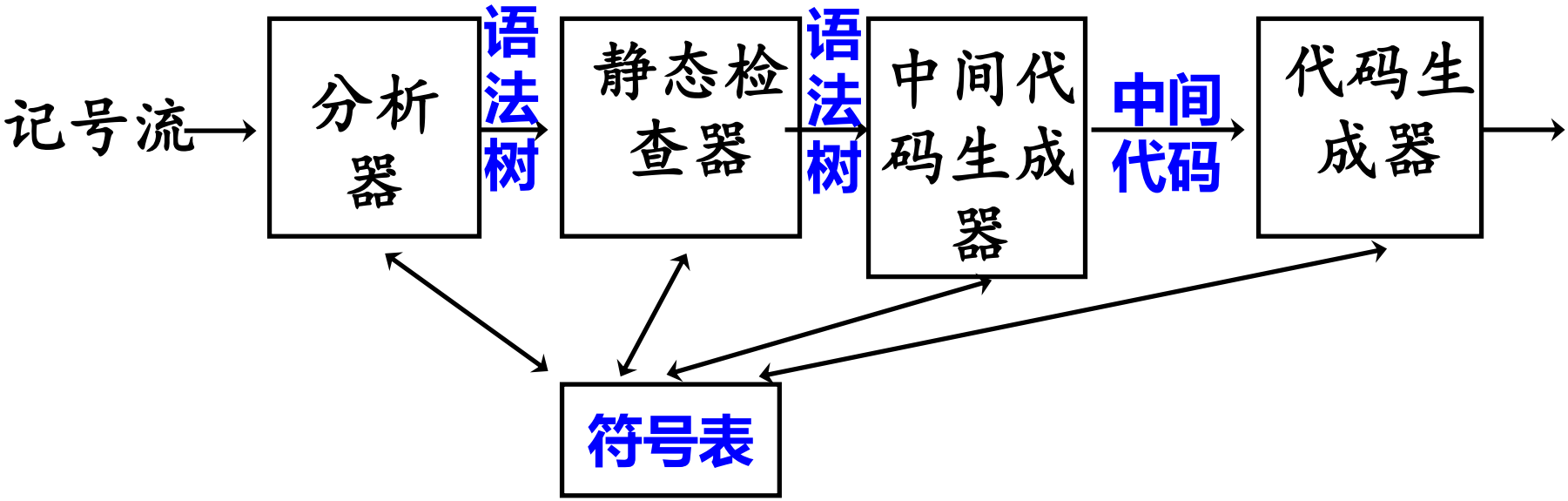
- 一种便于某些代码优化的中间表示
- 和三地址代码的主要区别
  - 所有赋值指令都是对不同名字的变量的赋值
  - 一个变量在不同路径上都定值的解决办法

```
if (flag) x = –1; else x = 1;
y = x * a;
```

改成

```
if (flag) x1 = –1; else x2 = 1;
x3 = φ(x1, x2);      //由flag的值决定用x1还是x2
```

主要内容



- 中间语言(Intermediate Representation)
  - 后缀表达式、图表示、三地址码、静态单赋值
- 中间代码生成
  - 声明语句(更新符号表)
  - 表达式、赋值语句(产生临时变量、查询符号表)
  - 布尔表达式、控制流语句(标号/回填、短路计算)

中间代码生成的关键问题

- 类型与符号表的变化
  - 多样化类型 => 整型(字节、字)、浮点型、类型符号表
  - 1个某类型的数据 => m 个字节(m为类型对应的字宽)
- 语句的翻译
  - 声明语句：不生成指令，但会更新符号表(作用域， 字宽及存放的相对地址)
  - 赋值语句：引入临时变量、数组/记录元素的地址计算、类型转换
  - 控制流语句：跳转目标的确定(引入标号或使用回填技术)、短路计算

符号表的设计

❑类型检查后的符号表

- ❖符号表条目：(标识符、存储类别、类型信息)
- ❖存储类别：**extern, static, register, …**
- ❖类型信息：(类别标识, 该类别关联的其他信息)
  - ➢如数组**(Array，(len, elemtype))**

❑本章符号表的变化

- ❖作用域 =>多个符号表
- ❖变量：字宽、存储的相对地址(以字节为单位)
- ❖记录类型：用符号表管理各个成员的字宽、相对地址

声明语句的翻译

- 一般不产生代码；仅将有关变量的属性填入符号表（如类型、存储偏移位置 - offset）
- 例：文法G_1如下：

$P \rightarrow D$

$D \rightarrow D \ ; \ D$

$D \rightarrow id : T$

$T \rightarrow int \mid real \mid array [number] of T_1 \mid \uparrow T_1$

声明语句的翻译

- 有关符号的属性

T.type - 变量所具有的类型，如

整型 INT

实型 REAL

数组类型 array（元素个数，元素类型）

指针类型 pointer（所指对象类型）

T.width - 该类型数据所占的字节数

offset - 变量的存储偏移地址

声明语句的翻译

T.type		T.width
整型	INT	4
实型	REAL	4
数组	array（number, T_1）	number.val * T_1.width
指针	pointer（T_1）	4

enter(name,type,offset)—将类型type和偏移offset填入符号表中name所在的表项。

声明语句的翻译

(1) P→*M* D

(2) D→D_1 ; D_2

(3) D→id : T

```
{ // 填入变量的相关属性
  enter( id.name, T.type, offset)
  // 计算下一可用偏移位置
  offset  = offset + T.width
}
```

(4) *M*→ε　{ offset := 0 // 偏移初始为0}

声明语句的翻译

(5) $T \rightarrow$ int { T.type := INT; T.width := 4 }

(6) $T \rightarrow$ real { T.type := REAL; T.width := 4 }

(7) $T \rightarrow$ array [number] of T_1

{

T.type := array(number.val, T_1.type);

T.width := number.val * T_1.width

}

(8) $T \rightarrow$ pointer(T_1)

{ T.type := pointer(T_1.type) ; T.width := 4 }

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》
中间代码生成 I

TBA