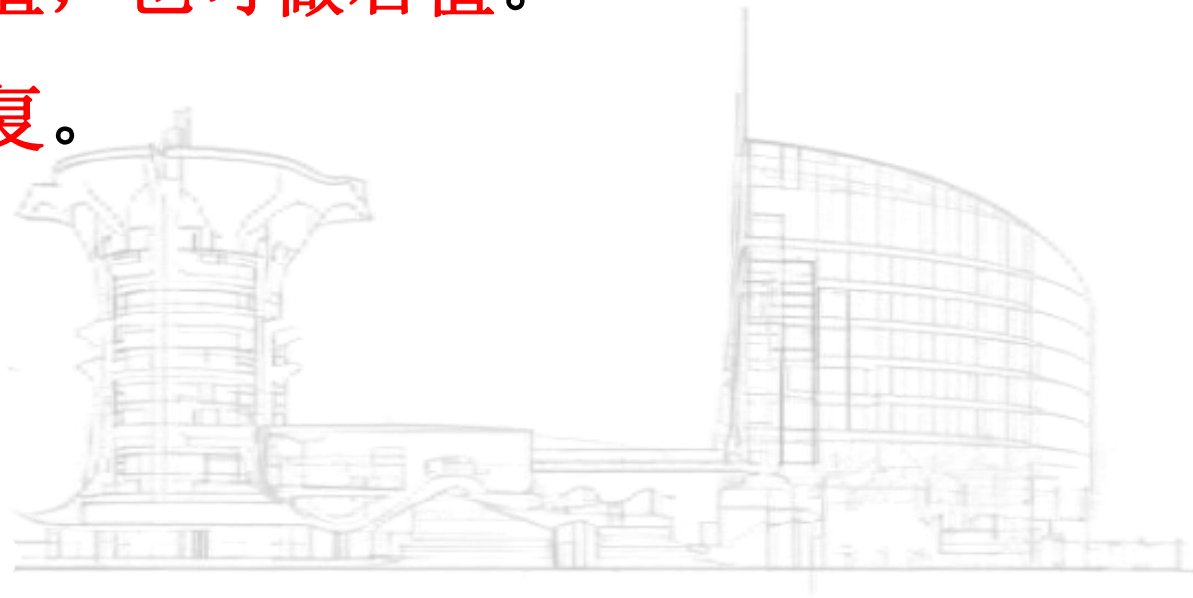
西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

DragonIR

——自定义线性中间表示

标识符

- 全局变量：以%g或者@开头，后面紧跟原始的全局变量名。
  - 建议全局变量按@开头。
  - 全局变量可赋值多次，即可做左值，也可做右值。
  - 要求全局变量全局唯一，不可重复。

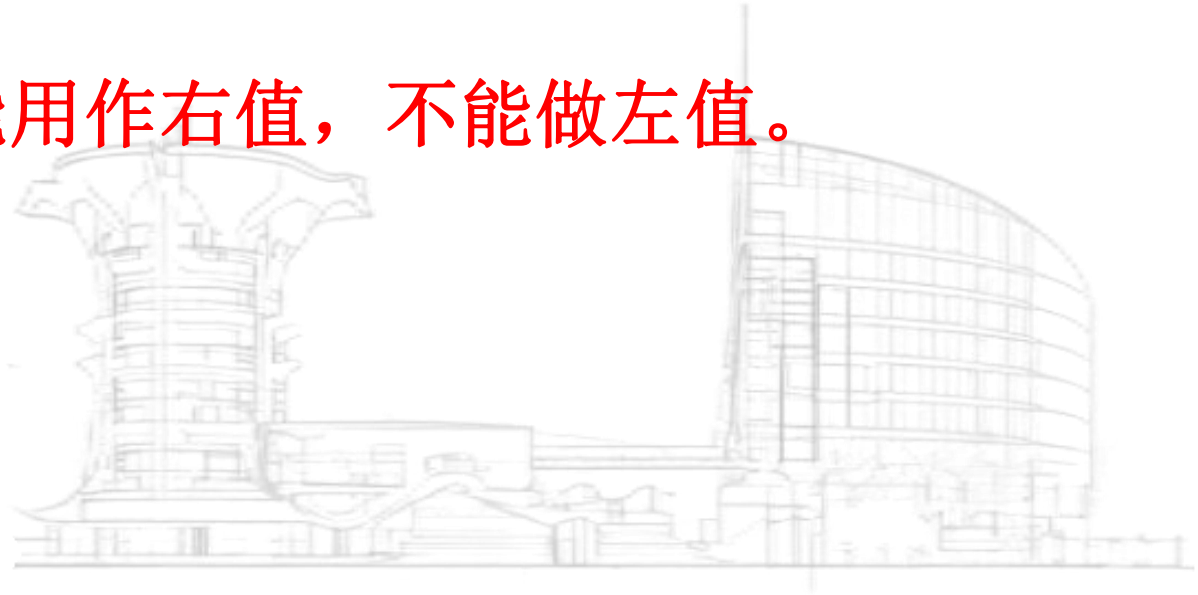


标识符

- 局部变量：以%*l*开头，后面为数字或字母组成的符号串，
  - 建议数字。
  - 注意%后面的是小写字母*l*（对应的大写为L），不是数字1。
  - 局部变量可赋值多次，即可做左值，也可做右值。

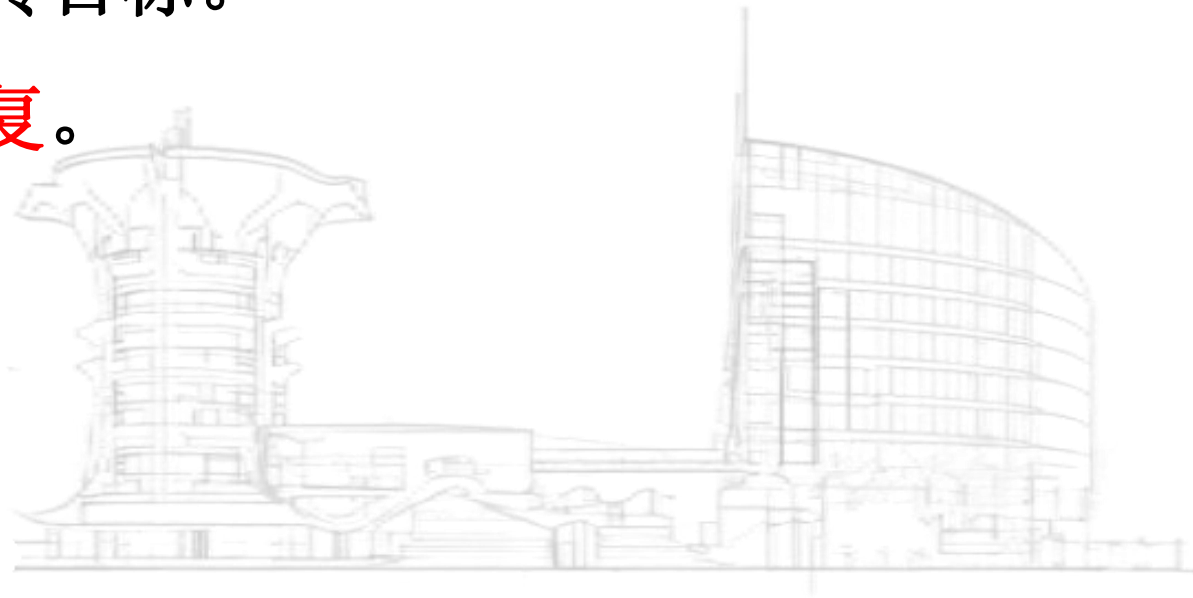
标识符

- 临时变量：以%t开头，后面为数字或字母组成的符号串，
  - 建议数字。
  - 编译器内部生成的临时变量；
  - 临时变量只能赋值1次，以后只能用作右值，不能做左值。



标识符

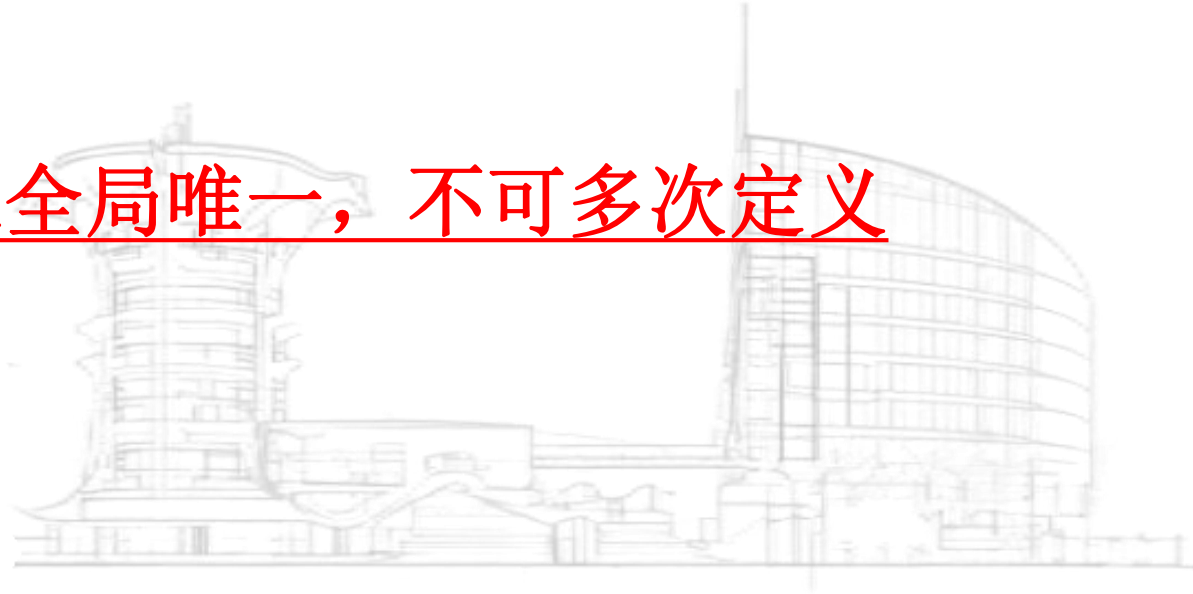
- 标签：以.L开头，后面为数字或字母组成的符号串，
  - 建议数字。
  - 用于条件或者无条件跳转指令跳转目标。
  - 在同一个函数内部，标签不能重复。



标识符

- 函数名：以@开头，后面紧跟原始的函数名
  - 要求函数名全局唯一，不可重复。

以@开头的符号成为全局符号，要求全局唯一，不可多次定义



常量或字面量

- 十进制形式的非负的整型字面量，如123

类型

- 基本类型：i32、i8、i1和void，支持指针
  - i32：32位int类型，相当于C语言的int类型
  - i8：8位int类型，相当于C语言的char类型
  - i1：1位int类型，相当于布尔类型
  - void：C语言的void类型，主要用于表示函数无返回值。
  - i32 *：指向i32的指针变量类
  - i8 *：指向i8的指针变量。

基本变量定义

- 变量定义要以declare开头，后面是类型，最后是标识符；
- 变量遵从先定义后使用的规则，否则语义错误。
  - declare i32 @f2 全局变量的声明，i32为类型，@开头的是全局变量，变量名为f2。
  - declare i32 %gf2 全局变量的声明，i32为类型，%g开头的是全局变量，变量名为f2。
  - declare i32 %l2 局部变量的声明，需要在栈内分配空间
  - declare i32 %t10 临时变量的声明，不需要在栈内分配空间，切记此类变量只能定值一次。
- 全局变量的定义需放在IR文件的头部，第一个函数定义的前面。
- 函数内的局部变量和编译器生成的临时变量在函数体的开始部分声明，且在第一条入口指令的前面。----这部分是函数内的变量符号表显示。

注释

- Dragon IR只支持单行注释；
- 以分号(;)开始，分号(;)开始到行末的所有内容为注释。

变量赋值指令

- 全局变量取值与设值；
  - @a=3　　常量赋值
  - @a=%t1　　临时变量赋值
  - @a=%*l*1　　局部变量赋值，建议先取到临时变量后赋值给全局变量
  - @a=@b　　全局变量赋值，建议先取到临时变量后赋值给全局变量
  - %t0=@a　　全局变量设值给临时变量
  - %*l*0=@a　　全局变量赋值局部变量，建议先取到临时变量后赋值局部变量
  - @b=@a　　全局变量赋值全局变量，建议先取到临时变量后赋值全局变量

变量赋值指令

- 局部变量取值与设值；
  - %*l*0=3　　常量赋值
  - %*l*0=%t1　　临时变量赋值
  - %*l*0=%*l*1　　局部变量赋值，建议先取到临时变量后赋值给局部变量
  - %*l*0=@b　　全局变量赋值，建议先取到临时变量后赋值给局部变量
  - %t0=%*l*0　　局部变量赋值临时变量
  - %*l*0=%*l*1　　局部变量赋值局部变量，建议先取到临时变量后赋值给局部变量
  - @b=%*l*0　　局部变量赋值全局变量，建议先取到临时变量后赋值给全局变量

变量赋值指令

- 临时变量取值与设值；
- 临时变量只能设值一次，之后可多次使用。
- 约定临时变量不能赋值给临时变量，常量不能赋值给临时变量。
  - %t0=@a　　全局变量赋值给临时变量
  - %t0=%l0　　局部变量赋值给临时变量

指针操作&内存操作

- 从内存取值
  - 形式定义：变量 = * 变量
  - 功能：从右侧变量（指针变量）所指向的内存中获取值赋值给左侧的变量（如左值的临时变量）。
  - 要求右侧的变量要是指针类型，并且左侧与右侧指向的类型一致。

指针操作|内存操作

- 把值保存到内存中
  - 形式定义： *变量 = 变量
  - 功能：把右侧的变量值保存到左侧变量（指针变量）所指向的内存中。
  - 要求左侧的变量要是指针类型，并且左侧与右侧指向的类型一致。

表达式运算指令

- 表达式运算结果赋值给临时变量，一般不直接赋值给局部变量或全局变量
- 表达式右侧的源操作数变量可以是可右值操作的所有变量，包含常量

二元算术运算

- 格式：变量1 = 算术运算符 变量2, 变量3
- add：加法运算　　sub：减法运算　　mul：乘法运算　　div：求商运算　　mod：求余运算
- 三个变量的类型一般要求要一致，但是add和sub指令还允许右侧变量一个为指针类型变量，一个为基本类型变量，结果为指针类型变量。
- 特别说明，add或sub指令可用于类似指针的加整型变量的偏移获取元素地址的功能，要求add或sub指令的第一个源操作数（逗号左侧的变量）是指针型变量或者数组首地址，第二个源操作数（都好右侧的变量）是整型的临时变量或常量，局部变量不建议。

表达式运算指令

- 表达式运算结果赋值给临时变量，一般不直接赋值给局部变量或全局变量
- 表达式右侧的源操作数变量可以是可右值操作的所有变量，包含常量

- 一元算术运算
  - 格式：变量1 = neg 变量2
  - neg：求负运算
  - 两个变量都是基本类型。

表达式运算指令

- 表达式运算结果赋值给临时变量，一般不直接赋值给局部变量或全局变量
- 表达式右侧的源操作数变量可以是可右值操作的所有变量，包含常量

关系运算

- 格式：变量1 = cmp 比较运算符 变量2，变量3
- le: 小于等于　lt：小于　gt：大于　ge：大于等于　ne：不等于　eq：等于
- 说明：比较结果（变量1）要求为bool类型，即i1。

表达式运算指令

- 表达式运算结果赋值给临时变量，一般不直接赋值给局部变量或全局变量
- 表达式右侧的源操作数变量可以是可右值操作的所有变量，包含常量

逻辑运算

- 在语义分析的过程中，逻辑运算采用短路求值，转换为有条件或者无条件指令实现。因此中间IR不定义逻辑运算指令。

跳转与标签指令

Label指令

- 作用：用于指令的跳转目标，类似与C语言的label语句
- 格式：标识符
- 说明：标识符用于定义Label名称，以.L开头的标识符。要求函数内唯一，不可重复。

无条件指令

- 格式：br label 标识符
- 说明：无条件跳转到标识符所代表的位置，标识符是Label指令所定义的Label名，其中关键字label可省略。
- 注意：Label名可在本条指令的前面，也可在本条指令的后面。。

跳转与标签指令

有条件指令

- 格式1：bc condvar, label X, label Y
- 说明：condvar条件临时变量，类型为i1； X为真跳转标签，Y假跳转标签；本指令表示若condvar为真，则跳转到X处执行，否则跳转到Y处执行。
- 格式2： bt condvar, label X
- 说明： condvar条件临时变量，类型为i1； X为真跳转标签；本指令表示若condvar为真，则跳转到X处执行，否则顺序执行下一条指令。
- 格式3： bf condvar, label X
- 说明： condvar条件临时变量，类型为i1； X为假跳转Label名；本指令表示若condvar为假，则跳转到X处执行，否则顺序执行下一条指令。
- 建议使用bc指令，不要使用bt或者bf指令

数组定义与操作指令

数组定义

- 函数内的局部数组变量定义类似与C语言的数组定义，只是把类型int修改为i32，变量名采用%*l*开头的标识符。
- **int a[10][15]; ----->declare i32 %*l*1[10][15**
- 对于全局或局部的数组定义，数组的维度必须大于0，但是对于形式参数数组，翻译后的数组的第一个维要为0，其它维大于0。
- 形参形式的数组变量，在编译器内部实际是按指针进行处理的，并且忽略第一维度的数值。例如对于数组int a[6][5]与int a[][5]是等价的，因此这里形参数组可翻译为i32 %t0[0][5]，第一维度的数值设置为0，形参名修改为以%t开头的变量，同时会分配同样类型的局部变量，declare i32 %l1[0][5]。

数组定义与操作指令

一维数组元素访问

一维数组元素取值

- 假设%*l*6代表变量m，%*l*2代表数组a，%*l*5代表变量k，数组元素类型为int
- m = a[k] 翻译为：

① %t10 = mul %*l*5, 4 ；元素偏移转换成字节偏移

② %t11 = add %*l*2, %t10 ；数组元素首地址+偏移得到元素的字节位置

③ %*l*6 = *%t11 ；从内存中取值，假定取4个字节的数据

数组定义与操作指令

一维数组元素访问

一维数组元素设值

- 假设%l6代表变量m，%l2代表数组a，%l5代表变量k，数组元素类型为int
- a[k] = m　翻译为：

① %t10 = mul %*l*5, 4

② %t11 = add %*l*2, %t10

③ *%t11=%*l*6

数组定义与操作指令

多维数组元素访问

多维数组元素取值

```
int a[10][10];
int m, k, t;
t = a[m][k];
```

```
declare i32 %l1[10][10]    ;数组a
declare i32 %l2            ;变量m
declare i32 %l3            ;变量k
declare i32 %l4            ;变量t

declare i32 %t5
declare i32 %t6
declare i32 %t7            ;保存a[m][k]相对数组首地址的偏移，单位字节
declare i32* %t8           ;数组元素的位置，类型为指向i8的指针

%t5 = mul %l2, 10
%t6 = add %t5, %l3
%t7 = mul %t6, 4           ;(m*10+k)*sizeof(int)
%t8 = add %l1, %t7         ; 数组元素首地址+元素的字节偏移后的指针
%l4 = *%t8                 ;从内存中获取i32类型的元素值
```

数组定义与操作指令

- 多维数组元素访问
- 多维数组元素设值

```
int a[10][10];
int m, k, t;
a[m][k] = t;
```

```
declare i32 %l1[10][10]     ;数组a
declare i32 %l2             ;变量m
declare i32 %l3             ;变量k
declare i32 %l4             ;变量t

declare i32 %t5
declare i32 %t6
declare i32 %t7
declare i32* %t8            ;数组元素的位置，类型为指向i8的指针

%t5 = mul %l2, 10
%t6 = add %t5, %l3
%t7 = mul %t6, 4            ;(m*10+k)*sizeof(int)
%t8 = add %l1, %t7          ; 数组元素首地址+元素的字节偏移后的指针
*%t8 = %l4                  ;把t的值设置到%t8所指向的内存中
```

函数定义

- 函数进行返回时，要事先创建一个用于保存返回值的局部变量（称为返回值局部变量）和一个函数出口标签指令
- 函数出口标签指令放在函数的尾部，用于MiniC语言的return语句的翻译时跳转到函数的出口，保持函数只有一个出口

- 函数返回值变量
  - 函数翻译时，如果返回值类型不是void，则需要在函数的开头申请一个返回值局部变量，用于保存返回值。
  - C语言函数内return语句的翻译首先把返回值保存到该返回值局部变量中，然后跳转到函数出口标签。

函数定义

函数入口指令

- 格式:**entry**
- 功能：函数的入口。每个函数有且只有一个。必须存在，用于表示函数的第一条有效指令。放置在declare语句指令的后面，其它指令的前面。

函数出口指令

- 格式：**exit 变量**
- 格式：**exit 常量**
- 格式：**exit**
- 功能：函数的出口指令。每个函数只能有一个出口。只存在于函数的最尾部。必须存在。这里的变量或常量是函数的返回值

函数定义

return语句翻译

- 有返回值的return语句翻译：先把返回值保存到返回值局部变量中，然后通过br指令跳转到函数出口标签。
- 没有返回值的return语句翻译：直接通过br指令跳转到函数出口标签。

函数形参处理

- 首先分配与形参对应的局部变量（将来用于在栈内分配空间），然后把原始的形参变成临时变量类型，在entry指令后把形参的值（临时变量）保存到对应的局部变量中。

函数定义

```
int test(int a, int b)
{
...
    return 0;
...
    return 1;
}
```

```
define i32 @test(i32 %t0, i32 %t1) {
declare i32 %l2      ;              返回值局部变量
            declare i32 %l3      ;形参a对应的局部变量
            declare i32 %l4     ; 形参b对应的局部变量
...
            entry
            %l3 = %t0    ;a赋值给a对应的局部变量
            %l4 = %t1    ;b赋值给b对应的局部变量
            ;后续所有对a和b的操作修改成对应的局部变量操作
...
            %l2 = 0
            br .L1
...
             %l2 = 1
            br .L1
            .L1:                    ;函数出口标签
            exit %l2
}
```

函数调用

- C语言的函数调用时函数参数求值顺序是未指定的，由编译器实现决定。
- Dragon IR规定函数实参按照自左往右运算，实参结果按自右往左入栈。
- 有函数值返回的函数调用
  - 使用call指令实现函数调用，或者借助arg指令与call指令完成。

```
e = test(c, d);
```

```
%t1 = %l1
%t2 = %l2
%t3 = call i32 @test(i32 %t1,i32 %t2)
%l3 = %t3
或者
%t3 = call i32 @test(i32 %l1,i32 %l2)
%l3 = %t3
其中%l1代表局部变量c，%l2代表局部变量d，%l3代表局部变量e
%t1,%t2,%t3为编译器内部生成的临时变量
```

函数调用

没有函数值返回的函数调用

```
putint(c)
```

```
%t1 = %l1
call void @putint(i32 %t1)
```

内置函数

- IR虚拟机内置了一些支持标准终端的输入与输出功能的函数功能。这些函数作为标准的库函数使用，自定义函数不能使用这些函数名。在翻译中可直接调用这些函数，不用定义和声明

```
int getint();
int getch();
int getarray(int a[]);

void putint(int a);
void putch(int a);
void putarray(int n,int a[]);
```

表达式语句翻译

```
int g1;

int main()
{
    int a, b, c;
    int sum;

    a = getint();
    b = getint();
    c = 3;

    sum = -(a + b * c) + a;

    putint(sum);

    return 0;
}
```

```
declare i32 @g1
define i32 @main() {
                declare i32 %l0
                declare i32 %l1 ; variable: a
                declare i32 %l2 ; variable: b
                declare i32 %l3 ; variable: c
                declare i32 %l4 ; variable: sum
                declare i32 %t5
                declare i32 %t6
                declare i32 %t7
                declare i32 %t8
                declare i32 %t9
                declare i32 %t10
                entry
                %t5 = call i32 @getint()        ; 函数调用 a = getint()
                %l1 = %t5
                %t6 = call i32 @getint()        ; 函数调用 b = getint()
                %l2 = %t6
                %l3 = 3                         ; c = 3
                %t7 = mul %l2, %l3              ; sum = -(a + b * c) + a;
                %t8 = add %l1, %t7
                %t9 = neg %t8
                %t10 = add %t9, %l1
                %l4 = %t10
                call void @putint(i32 %l4)      ; putint(sum);
                %l0 = 0                         ; return 0; 优化掉了跳转
                exit %l0
}
```

分支语句翻译

```
int g1;

int main ()
{
   int a;
   int b;
   int c;

   g1 = 1;

   a = getint();
   b = getint();
   c = getint();

   if((a < b) && (a < c)) {
      g1 = 0;
   }

   putint(g1);

   return 0;
}
```

```
declare i32 @g1
define i32 @main() {
                declare i32 %l0
                declare i32 %l1 ; variable: a
                declare i32 %l2 ; variable: b
                declare i32 %l3 ; variable: c
                declare i32 %t4
                declare i32 %t5
                declare i32 %t6
                declare i1 %t7
                declare i1 %t8
                entry
                @g1 = 1
                %t4 = call i32 @getint()
                %l1 = %t4
                %t5 = call i32 @getint()
                %l2 = %t5
                %t6 = call i32 @getint()
                %l3 = %t6
                %t7 = cmp lt %l1, %l2          ; a < b
                bc %t7, label .L2, label .L4    ; 逻辑与短路求值
.L2:
                %t8 = cmp lt %l1, %l3          ; a < c
                bc %t8, label .L3, label .L4
.L3:                                    ; 真出口
                @g1 = 0
                br label .L4
.L4:                                    ; 假出口和if语句的出口
                call void @putint(i32 @g1)
                %l0 = 0
                exit %l0
}
```

While语句翻译

```
int g1;

int main()
{
    int a = 1;
    int sum = 0;

    while(a < 101) {
        sum = sum + a;
        a = a + 1;
    }

    putint(sum);

    return 0;
}
```

```
declare i32 @g1
define i32 @main() {
                declare i32 %l0
                declare i32 %l1 ; variable: a
                declare i32 %l2 ; variable: sum
                declare i1 %t3
                declare i32 %t4
                declare i32 %t5
                entry
                %l1 = 1
                %l2 = 0
                br label .L2                    ; 这个跳转指令可省略
.L2:                                            ; while循环的入口
                %t3 = cmp lt %l1, 101
                bc %t3, label .L3, label .L4
.L3:                                            ; while循环体入口
                %t4 = add %l2, %l1
                %l2 = %t4
                %t5 = add %l1, 1
                %l1 = %t5
                br label .L2                    ; 跳转到循环的入口
.L4:                                            ; while循环的出口
                call void @putint(i32 %l2)
                %l0 = 0
                exit %l0
}
```

函数调用翻译（不使用arg指令）

```
int test(int a, int b)
{
    int c;
    c = a + b;
    return c;
}

int main()
{
    int a, b;
    int sum;
    a = getint();
    b = getint();
    sum = test(a, b);
    putint(sum);
    return 0;
}
```

```
define i32 @test(i32 %t0, i32 %t1) {
                declare i32 %l2 ; variable: a
                declare i32 %l3 ; variable: b
                declare i32 %l4
                declare i32 %l5 ; variable: c
                declare i32 %t6
                entry
                %l2 = %t0
                %l3 = %t1
                %t6 = add %l2, %l3
                %l5 = %t6
                %l4 = %l5
                exit %l4
}
define i32 @main() {
                declare i32 %l0
                declare i32 %l1 ; variable: a
                declare i32 %l2 ; variable: b
                declare i32 %l3 ; variable: sum
                declare i32 %t4
                declare i32 %t5
                declare i32 %t6
                entry
                %t4 = call i32 @getint()
                %l1 = %t4
                %t5 = call i32 @getint()
                %l2 = %t5
                %t6 = call i32 @test(i32 %l1,i32 %l2)  ; sum = test(a, b)
                %l3 = %t6
                call void @putint(i32 %l3)
                %l0 = 0
                exit %l0
}
```

函数调用翻译（使用arg指令）

```
int test(int a, int b)
{
    int c;
    c = a + b;
    return c;
}

int main()
{
    int a, b;
    int sum;
    a = getint();
    b = getint();
    sum = test(a, b);
    putint(sum);
    return 0;
}
```

```
define i32 @test(i32 %t0, i32 %t1) {
                declare i32 %l2 ; variable: a
                declare i32 %l3 ; variable: b
                declare i32 %l4
                declare i32 %l5 ; variable: c
                declare i32 %t6
                entry
                %l2 = %t0
                %l3 = %t1
                %t6 = add %l2, %l3
                %l5 = %t6
                %l4 = %l5
                exit %l4
}
define i32 @main() {
                declare i32 %l0
                declare i32 %l1 ; variable: a
                declare i32 %l2 ; variable: b
                declare i32 %l3 ; variable: sum
                declare i32 %t4
                declare i32 %t5
                declare i32 %t6
                entry
                %t4 = call i32 @getint()
                %l1 = %t4
                %t5 = call i32 @getint()
                %l2 = %t5
                arg %l1                 ; 自左往右传递参数，传递第一个实参
                arg %l2                 ; 传递第二个实参
                %t6 = call i32 @test()   ; 这里不写实参，只写调用
                %l3 = %t6
                arg %l3
                call void @putint()
                %l0 = 0
                exit %l0
```

一维数组翻译（1）

```
int sum(int a[10], int n)
{
   int k, sum;

   k = 0;
   sum = 0;
   while(k < n) {
      sum = sum + a[k];
      k = k + 1;
   }

   return sum;
}

int main()
{
   int n;
   int a[10];

   n = getarray(a);

   int t;
   t = sum(a, n);

   return t;
}
```

```
define i32 @sum(i32 %t0[0], i32 %t1) {
                declare i32 %l2[0] ; variable: a     ; 一维数组对应的栈内局部变量
                declare i32 %l3 ; variable: n
                declare i32 %l4
                declare i32 %l5 ; variable: k
                declare i32 %l6 ; variable: sum
                declare i1 %t7
                declare i32* %t8
                declare i32 %t9
                declare i32 %t10
                declare i32 %t11
                declare i32 %t12
                declare i32 %t13
                entry
                %l2 = %t0
                %l3 = %t1
                %l5 = 0
                %l6 = 0
                br label .L2
.L2:
                %t7 = cmp lt %l5, %l3
                bc %t7, label .L3, label .L4
.L3:
; 下面三行对应a[k]的翻译，其值保存在%t11中
                %t9 = mul %l5, 4
                %t8 = add %l2, %t9
                %t11 = *%t8
                %t12 = add %l6, %t11
                %l6 = %t12
                %t13 = add %l5, 1
                %l5 = %t13
                br label .L2
```

一维数组翻译（2）

```
int sum(int a[10], int n)
{
   int k, sum;

   k = 0;
   sum = 0;
   while(k < n) {
      sum = sum + a[k];
      k = k + 1;
   }

   return sum;
}

int main()
{
   int n;
   int a[10];

   n = getarray(a);

   int t;
   t = sum(a, n);

   return t;
}
```

```
.L4:
                %l4 = %l6
                exit %l4
}
define i32 @main() {
                declare i32 %l0
                declare i32 %l1 ; variable: n
                declare i32 %l2[10] ; variable: a
                declare i32 %l3 ; variable: t
                declare i32 %t4
                declare i32 %t5
                entry
        ; 一维数组实参传递
                %t4 = call i32 @getarray(i32 %l2[10])
                %l1 = %t4
 ; 一维数组和整型的实参传递
                %t5 = call i32 @sum(i32 %l2[10],i32 %l1)
                %l3 = %t5
                %l0 = %l3
                exit %l0
}
```

多维数组翻译（1）

```
int getarray(int a[]);
void putint(int);

int test(int a[10][5]) {
    int sum = 0;
    int k, m;

    k = 0;
    while (k < 10) {

        m = 0;
        while (m < 5) {

            sum = sum + a[k][m];
            m = m + 1;
        }

        k = k + 1;
    }

    return sum;
}
```

```
int main() {
    int temp;
    int a[10][10][5];
    int m, k, n;

    k = 0;
    while (k < 10) {

        m = 0;
        while (m < 10) {
            n = 0;
            while (n < 5) {
                a[k][m][n] = m * k * n;
                n = n + 1;
            }
            m = m + 1;
        }
        k = k + 1;
    }
    k = 0;
    while (k < 10) {

        temp = test(a[k]);
        putint(temp);

        k = k + 1;
    }

    return 0;
}
```

多维数组翻译（2）

```
define i32 @test(i32 %t0[0][5]) {
        declare i32 %l1[0][5] ; variable: a
        declare i32 %l2
        declare i32 %l3 ; variable: sum
        declare i32 %l4 ; variable: k
        declare i32 %l5 ; variable: m
        declare i1 %t6
        declare i1 %t7
        declare i32 %t8
        declare i32 %t9
        declare i32* %t10
        declare i32 %t11
        declare i32 %t12
        declare i32 %t13
        declare i32 %t14
        declare i32 %t15
        declare i32 %t16
        entry
        %l1 = %t0
        %l3 = 0
        %l4 = 0
        br label .L2
```

```
.L2:
        %t6 = cmp lt %l4, 10
        bc %t6, label .L3, label .L7
.L3:
        %l5 = 0
        br label .L4
.L4:
        %t7 = cmp lt %l5, 5
        bc %t7, label .L5, label .L6
.L5:
        %t8 = mul %l4, 5
        %t9 = add %t8, %l5
        %t11 = mul %t9, 4
        %t10 = add %l1, %t11
        %t13 = *%t10
        %t14 = add %l3, %t13
        %l3 = %t14
        %t15 = add %l5, 1
        %l5 = %t15
        br label .L4
.L6:
        %t16 = add %l4, 1
        %l4 = %t16
        br label .L2
.L7:
        %l2 = %l3
        exit %l2
}
```

多维数组翻译（3）

```
define i32 @main() {
        declare i32 %l0
        declare i32 %l1 ; variable: temp
        declare i32 %l2[10][10][5] ; variable: a
        declare i32 %l3 ; variable: m
        declare i32 %l4 ; variable: k
        declare i32 %l5 ; variable: n
        declare i1 %t6
        declare i1 %t7
        declare i1 %t8
        declare i32 %t9
        declare i32 %t10
        declare i32 %t11
        declare i32 %t12
        declare i32* %t13
        declare i32 %t14
        declare i32 %t15
        declare i32 %t16
        declare i32 %t17
        declare i32 %t18
        declare i32 %t19
        declare i32 %t20
        declare i1 %t21
        declare i32 %t22
        declare i32 %t23[10][5]
        declare i32 %t24
        declare i32 %t25
        declare i32 %t26
        entry
        %l4 = 0
        br label .L2
```

```
.L2:
        %t6 = cmp lt %l4, 10
        bc %t6, label .L3, label .L10
.L3:
        %l3 = 0
        br label .L4
.L4:
        %t7 = cmp lt %l3, 10
        bc %t7, label .L5, label .L9
.L5:
        %l5 = 0
        br label .L6
.L6:
        %t8 = cmp lt %l5, 5
        bc %t8, label .L7, label .L8
.L7:
        ; a[k][m][n] = m * k * n，正确是先右侧的乘法后赋值给数组元素
        ; IR虚拟机输出参考先左后右是错误的，需注意。
        %t16 = mul %l3, %l4
        %t17 = mul %t16, %l5
        %t9 = mul %l4, 10
        %t10 = add %t9, %l3
        %t11 = mul %t10, 5
        %t12 = add %t11, %l5
        %t14 = mul %t12, 4
        %t13 = add %l2, %t14
        *%t13 = %t17
        %t18 = add %l5, 1
        %l5 = %t18
        br label .L6
```

多维数组翻译（4）

```
.L8:
        %t19 = add %l3, 1
        %l3 = %t19
        br label .L4
.L9:
        %t20 = add %l4, 1
        %l4 = %t20
        br label .L2
.L10:
        %l4 = 0
        br label .L11
.L11:
        %t21 = cmp lt %l4, 10
        bc %t21, label .L12,
```

```
.L12:
; 下面五行是temp = test(a[k])语句的翻译
; a[k]的类型是i32 [10][5]
        %t22 = mul %l4, 50
        %t24 = mul %t22, 4
        %t23 = add %l2, %t24
        %t25 = call i32 @test(i32 %t23[10][5])
        %l1 = %t25
        call void @putint(i32 %l1)
        %t26 = add %l4, 1
        %l4 = %t26
        br label .L11
.L13:
        %l0 = 0
        exit %l0
}
```

函数return语句翻译

```
int main()
{
    int a;

    a = getint();

    if(a > 0) {
        return a + 1;
    } else if(a < 0) {
        return a - 1;
    }

    return 0;
}
```

```
define i32 @main() {
                    declare i32 %l0
                    declare i32 %l1 ; variable: a
                    declare i32 %t2
                    declare i1 %t3
                    declare i32 %t4
                    declare i1 %t5
                    declare i32 %t6
                    entry
                    %t2 = call i32 @getint()
                    %l1 = %t2
                    %t3 = cmp gt %l1, 0
                    bc %t3, label .L2, label .L3
.L2:
; return a + 1  先把a+1的结果保存到返回值局部变量，后跳转函数出口
                    %t4 = add %l1, 1
                    %l0 = %t4
                    br label .L6
.L3:
                    %t5 = cmp lt %l1, 0
                    bc %t5, label .L4, label .L5
.L4:
; return a - 1  先把a-1的结果保存到返回值局部变量，后跳转函数出口
                    %t6 = sub %l1, 1
                    %l0 = %t6
                    br label .L6
.L5:
; return 0 先把0保存到返回值局部变量，后跳转函数出口
                    %l0 = 0
                    br label .L6
; 函数出口和函数退出指令并带返回值
.L6:
                    exit %l0
}
```

西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

谢谢大家