编译器课程设计中间代码格式2022

为了方便课程组对编译器产生的中间代码进行审核，本文档**对中间代码的输出格式给出一些建议。**请各位同学结合本文建议，合理实现中间代码的输出。选择生成LLVM IR时不受此限制。

表达式

表达式运算均需要以`Z = exp`的形式进行输出，其中`Z`为结果，`exp`为右值表达式。复杂表达式应当按照运算的顺序拆解为若干二元运算与一元运算的中间代码序列，过程中产生的临时变量名自行定义。

- 二元运算输出格式应尽量满足**中缀表达式的四元式形式**，`exp`为`X op Y`，其中`X`为第一个操作数，`Y`为第二个操作数，`op`为运算符。
- 一元运算输出格式中`exp`为`op X`，其中`X`为操作数，`op`为运算符。
- 若设计了其他类型的表达式运算，需要具有良好的可读性。

如：

```
z = a * (b + c);
```

```
t1 = b + c
z = a * t1
```

函数

- **函数声明**：

  ```
  int foo(int a, int b) {
      // ...
  }
  ```

  ```
  int foo()
  para int a
  para int b
  ```

- **函数调用**：

  ```
  i = tar(x, y);
  ```

  ```
  push x
  push y
  call tar
  i = RET
  ```

- **函数返回**：

  ```
  return x + y;
  ```

  ```
  t1 = x + y
  ret t1
  ```

变量和常量

- **变量声明及初始化**：

```
int i;
int j = 1;
```

```
var int i
var int j = 1
```

- **常量声明**：

```
const int c = 10;
```

```
const int c = 10
```

分支和跳转

- **标签**：分支跳转的目标地址。

```
label:
    z = x + y
```

- **条件分支**：注意涉及`&&`和`||`的地方要满足逻辑短路。

```
if (a == 1 && b <= 2) {
    // ...
}
```

```
cmp a, 1
bne end_if
cmp b, 2
bgt end_if
# if body ...
end_if:
```

- **跳转**：典型如`continue`语句和`break`语句。

```
while (/* ... */) {
    if (/* ... */) {
        break;
    }
}
```

```
loop_begin:
# some statements ...
goto loop_end
# some statements ...
loop_end:
```

数组

数组的定义及读写应满足`L = R`的形式。

- **数组定义**：

```
int a[4] = {1, 2, 3, 4};
```

```
arr int a[4]
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
```

- **数组读取**：

```
z = a[x][y]; // a[4][2]
```

```
t1 = x * 2
t2 = y + t1
z = a[t2]
```

- **数组存储**：

```
a[x][y] = z; // a[4][2]
```

```
t1 = x * 2
t2 = y + t1
a[t2] = z
```

其他说明

其他本文档未提及的语法现象（如全局变量、字符串、数组类型函数参数等），或自行定义的中间代码操作（如基本块、φ函数等），原则上均需要尽可能满足**中缀形式的四元式格式**，并具有良好的可读性。

本文档仅建议同学们采用合适的**输出格式**来输出中间代码，便于课程组审核，编译器中具体使用的中间代码**数据结构**由同学们自由设计实现。