实验报告

PA4

1. 实验目标

本阶段，我们将在之前的编译前端的基础上，实现**代码生成**，即将**带注释的AST转换为MIPS汇编代码**。至此，我们就实现了一个完整的Cool语言编译器。

2. 代码整体结构一览

说是要生成代码，但具体要生成怎样的代码呢？在官方提供的各manual中都没有很清晰的描述，只给出了大致需要实现的内容，这让人很一头雾水。这个时候，我们可以**先看看官方编译出的MIPS代码是长什么样子的，从而对整体结构和各自的功能有一个明确的认识**。

以下面这段十分简单的Cool代码为例：

```
class A {
    x:Int <- 1;
};

class Main{
    main():Object {
        1
    };
};
```

下面分段介绍官方生成的MIPS代码。

2.1 数据部分(Data)

以下部分属于数据。

2.1.1 全局常量与垃圾回收测试

```
    .data
    .align  2
    .globl  class_nameTab
    .globl  Main_protObj
    .globl  Int_protObj
    .globl  String_protObj
    .globl  bool_const0
    .globl  bool_const1
    .globl  _int_tag
    .globl  _bool_tag
    .globl  _string_tag
_int_tag:
    .word   2
_bool_tag:
    .word   3
_string_tag:
    .word   4
    .globl  _MemMgr_INITIALIZER
_MemMgr_INITIALIZER:
    .word   _NoGC_Init
    .globl  _MemMgr_COLLECTOR
_MemMgr_COLLECTOR:
    .word   _NoGC_Collect
    .globl  _MemMgr_TEST
_MemMgr_TEST:
    .word   0
    .word   -1
```

最开始这一部分是全局常量的定义，包括class名字表(name table)，Main、Int、String的原型(prototype) Object，True和False以及Int、Bool和String的Class tag。这些概念在后面会详细解释。

此外，下面还定义了若干垃圾回收(garbage collect)相关内容，在实现过程中我们暂时不考虑这部分内容。

2.1.2 常量定义

```
str_const9:
    .word   4
    .word   5
    .word   String_dispTab
    .word   int_const1
    .byte   0
    .align  2
    .word   -1
str_const8:
    .word   4
    .word   6
    .word   String_dispTab
    .word   int_const2
    .ascii  "Main"
    .byte   0
    .align  2
    .word   -1
str_const7:
    .word   4
    .word   5
    .word   String_dispTab
    .word   int_const0
    .ascii  "A"
    .byte   0
```

接下来是常量的定义。这些常量包括了程序中所有出现的字符串、整数和布尔值，包括类名、常量字符串、方法名等。

2.1.3 Class name table

```
class_nameTab:
    .word   str_const2
    .word   str_const3
    .word   str_const4
    .word   str_const5
    .word   str_const6
    .word   str_const7
    .word   str_const8
```

类名表(Class name table)**记录了程序中定义的每个类（包括基本类）的名字**。这里每个str_const对应的就是类名字符串。cool-runtime 4. Expected Labels中对其的描述是：

> A table, which at index (class tag) * 4 contains a pointer to a String object containing the name of the class associated.

这里，**class tag是一个唯一标识每个类的整数**，类名表可以通过class tag来获取对应的类名。另一方面，class tag在类的比较中也会用到。

2.1.4 Class object table

```
class_objTab:
    .word   Object_protObj
    .word   Object_init
    .word   IO_protObj
    .word   IO_init
    .word   Int_protObj
    .word   Int_init
    .word   Bool_protObj
    .word   Bool_init
    .word   String_protObj
    .word   String_init
    .word   A_protObj
    .word   A_init
    .word   Main_protObj
    .word   Main_init
```

类对象表(Class object table)**记录了程序中每个类（包括基本类）的原型对象(Prototype Object)和初始化方法(Init Method)的地址**。cool-runtime 4. Expected Labels中对其的描述是：

> A table, which at index (class tag) * 8 contains a pointer to the prototype object and at index (class tag) * 8 + 4 contains a pointer to the initialization method for that class.

可以看到，类对象表也是**通过class tag**来访问的。关于原型对象和初始化方法，我们在后面介绍。

2.1.5 Dispatch Table

```
Object_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
Main_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   Main.main
A_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
String_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   String.length
    .word   String.concat
    .word   String.substr
Bool_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
Int_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
```

类对象表的下面是**一系列调用表(Dispatch Tables)**。每个调用表**记录了该类可以调用的所有方法（包括继承而来的方法）的地址**。其记录的格式是**类名.方法名**。我们会在处理调用/静态调用表达式等地方时用到它。

2.1.6 Prototype Object

```
Object_protObj:
    .word   0
    .word   3
    .word   Object_dispTab
    .word   -1
Main_protObj:
    .word   6
    .word   3
    .word   Main_dispTab
    .word   -1
A_protObj:
    .word   5
    .word   4
    .word   A_dispTab
    .word   int_const1
    .word   -1
String_protObj:
    .word   4
    .word   5
    .word   String_dispTab
    .word   int_const1
    .word   0
    .word   -1
```

原型对象(Prototype Object)是一个类的默认实例化结果。我们**主要在需要创建一个新对象(如new表达式)时用到它，通过Object.copy()方法复制某个类的原型对象来完成这件事**。具体可见cool-runtime 2.2 Prototype Objects的描述。

每个原型对象内部的结构涉及到Cool中的**对象分布(Object Layout)**。参考cool-runtime 2.1 Object Layout的描述：

```
class Grandparent {
  first : Object;
};
class Parent inherits Grandparent {
  second : Object;
}
class Child inherits Parent {
  third : Object;
  fourth : Object;
}
```

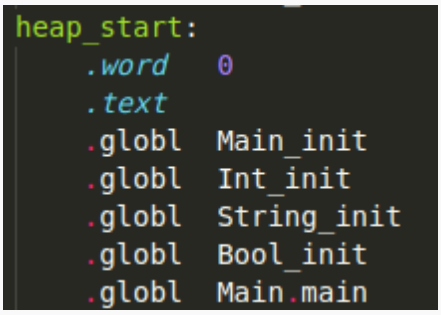
offset -4	Garbage Collector Tag
offset 0	Class tag
offset 4	Object size (in 32-bit words)
offset 8	Dispatch pointer
offset 12	Attribute `first`
offset 16	Attribute `second`
offset 20	Attribute `third`
offset 24	Attribute `fourth`

Cool中**所有变量都是对象(object)**，且Cool按字节寻址，以4字节为1字(word)。其内部构成为：

- 偏移 -4:垃圾回收标记。同时也标识上一个object的结束。
- 偏移 0 :**Class tag**。

- 偏移 4 : 对象大小。
- 偏移 8 : 调用指针。指向该类对象的**调用表**。
- 偏移 12及以后：**属性。包括继承来的属性**。

2.1.7 Heap



堆区(Heap)用于存放动态对象，如通过new创建的对象等。我们可以看到，最开始是Main、Int、Bool和String的初始化方法放在堆上，然后执行main函数。这和cool-runtime 6.1 Execution Startup中描述的Cool程序启动过程是一致的。

2.2 Code

以下部分属于实际的代码。

2.2.1 Init Method

```
Object_init:
    addiu   $sp $sp -12
    sw  $fp 12($sp)
    sw  $s0 8($sp)
    sw  $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    move    $a0 $s0
    lw  $fp 12($sp)
    lw  $s0 8($sp)
    lw  $ra 4($sp)
    addiu   $sp $sp 12
    jr  $ra
Main_init:
    addiu   $sp $sp -12
    sw  $fp 12($sp)
    sw  $s0 8($sp)
    sw  $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    jal Object_init
    move    $a0 $s0
    lw  $fp 12($sp)
    lw  $s0 8($sp)
    lw  $ra 4($sp)
    addiu   $sp $sp 12
    jr  $ra
A_init:
    addiu   $sp $sp -12
    sw  $fp 12($sp)
    sw  $s0 8($sp)
    sw  $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
```

代码段以**每个类的初始化方法(Init Method)开始**。初始化方法**用于初始化对应类的对象（主要是对其中的属性进行赋值），相当于C++中的构造函数**。其通常是**与原型对象配合使用：复制一个原型对象并使用初始化方法对其赋值**。我们后面会看到，这也正是new表达式的基本操作语义。

2.2.2 Method Definition

```
Main.main:
    addiu   $sp $sp -12
    sw  $fp 12($sp)
    sw  $s0 8($sp)
    sw  $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    la  $a0 int_const0
    lw  $fp 12($sp)
    lw  $s0 8($sp)
    lw  $ra 4($sp)
    addiu   $sp $sp 12
    jr  $ra
```

最后，就是我们定义的各方法的对应代码了。这也是代码的主要部分。

2.3 总览

根据PA5 Handout 3.Design中给出的需求，以及上述结构分析，我们将**按照代码结构顺序，逐步完成以下任务**：

- 生成**常量**的代码，即Int/Bool/String。
- 生成**类名表**的代码。
- 生成**类对象表**的代码。
- 生成每个类的**调用表**的代码。
- 生成每个类的**原型对象**的代码。
- 生成每个类的**初始化方法**的代码。
- 生成每个**方法**的对应代码。

每个部分的具体结构参照上面给出的实例。因此后续也不再详细分析具体结构以及生成的代码各部分的构造方式，只在涉及实现逻辑时详细说明。

3. 数据部分代码生成

3.1 生成常量代码

常量代码对应的函数为`code_constants()`。分别为String、Int和Bool常量生成代码。其对应的具体函数分别为

- `StringEntry::code_def`
- `IntEntry::code_def`
- `BoolConst::code_def`

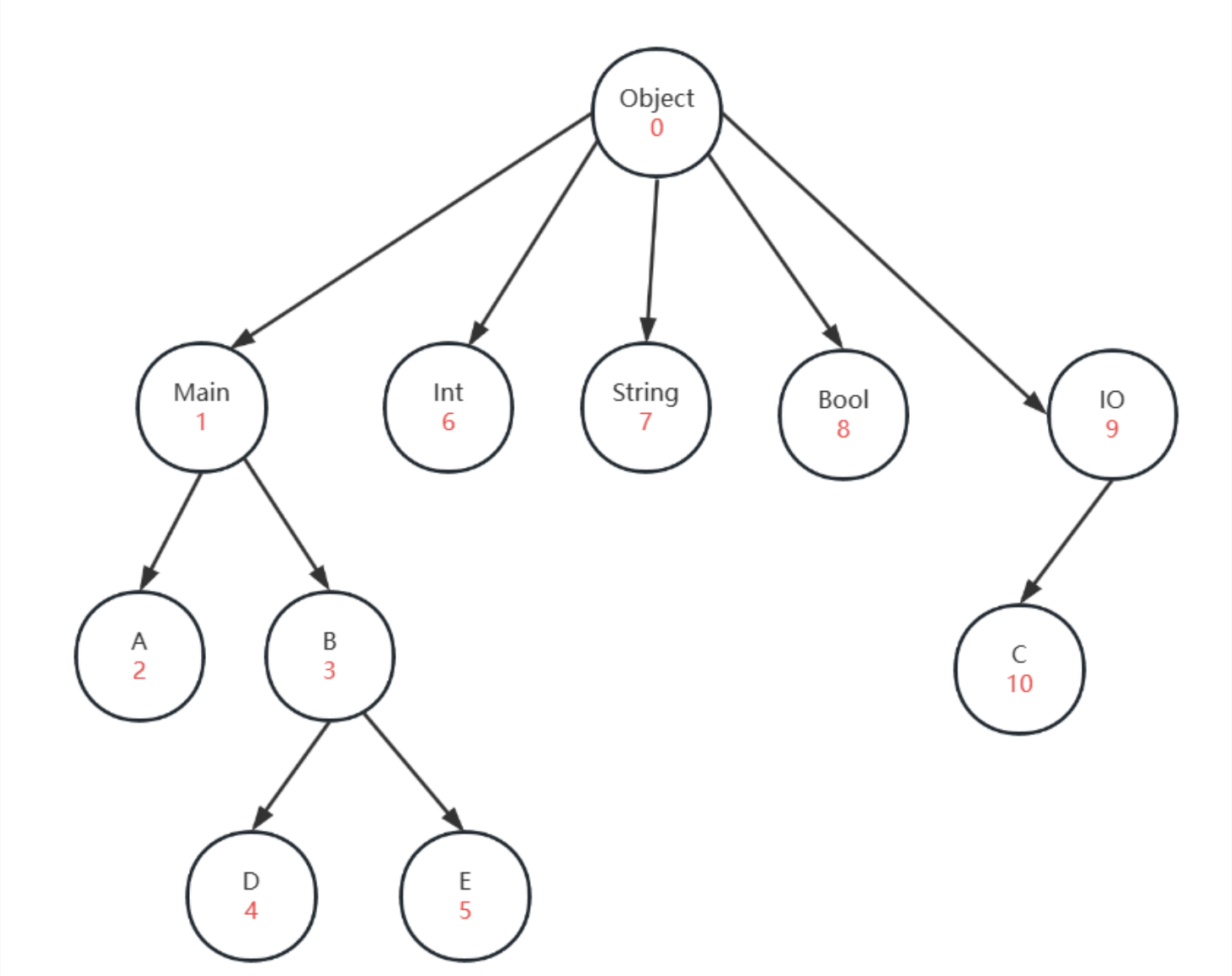
我们以`StringEntry::code_def`为例，见代码8.1：
这部分代码已经给出了部分实现。和官方生成的代码对比，我们可以发现主要有两点差别：

- 没有指定调用表；
- 没有确定class tag。

调用表可以直接添加一句`emit_disptable_ref(Str,s)`来添加对应的调用表。至于class tag的处理，见3.2 设定class tag。
Int和Bool变量的生成改动是一样的，都是添加调用表的声明，略。

3.2 设定class tag

表面上，class tag的要求只有每个类拥有一个独特的class tag整数即可。但实际上并没有那么简单。事实上，我们需要**按照DFS序，也即整棵AST的前序来设定class tag**。例如，某程序形成的AST可能是这样的形状：



图中已经在每个结点下面标好了其**按照前序对应的class tag**。这样做的用处在很长一段时间内看不出来，主要在**处理typcase语句时起到重要的作用**。见5.6 case表达式。

于是，我们可以简单地实现一个DFS函数，前序遍历整棵AST，并在过程中为每个`CgenNode`分配class tag。见代码8.2。

- 这里我们将class tag作为每个`CgenNode`的属性来保存，便于后续使用。
- `tag_to_CgenNode`是一个索引为class tag，内容为`CgenNodeP`的数组。我们利用该数组**按照class tag**遍历所有类，这主要在类名表的代码生成用到，见3.2 生成Class name table的代码。

- 我们需要特殊处理Int/String/Bool三类。因为它们的class tag是**全局常量**，也需要在对应类常量的数据中用到。碰到这三类时，就设定对应的tag。

3.2 生成Class name table的代码

类名表的内容是：**在对应class tag偏移处，保存对应类字符串的指针**。因此，我们**需要按照class tag从小到大的顺序来生成类名表的代码**。这可以利用我们在上一部分生成的`tag_to_CgenNode`来完成。也即直接遍历`tag_to_CgenNode`数组，对其中每个`CgenNodeP`，获取其名字对应的字符串，并直接生成对应的信息。

代码见代码8.3。

- 注意这里是通过`Stringtable.lookup_string`方法来找到类名Symbol对应的字符串的。
- 然后对于每个字符串，调用其**常量代码生成方法**。

3.3 生成Class object table的代码

类对象表的内容是：**在每个classtag偏移处，分别保存对应类的原型对象和初始化方法的指针**。因此可以同样按照class tag的顺序遍历`tag_to_CgenNode`数组，然后直接生成需要的代码。

代码见代码8.4。

- 直接用`emit_protobj_ref`和`emit_init_ref`方法生成对应的代码即可。

3.4 生成每个类的Dispatch table的代码

每个类的调用表的内容是：**保存每个类可以调用的所有方法的指针**，从而在实际调用时可以在相应的调用表中找到需要调用的方法。其性质**类似于C++中的虚函数表，实现了Cool语言的多态和动态绑定**。具体表现可以见5.2 dispatch表达式。

为了生成类的调用表，我们就**需要找到一个类的继承路径(Inheritance Path)，收集路径上所有的方法，并把其记录到对应类中**。代码见代码8.5。

- 这里`methodTable`和`attrTable`主要记录每个类**本类的方法和属性**。和语义分析中的实现类似，主要出于方便考虑。
- `getInheritancePath()`获取继承路径，只需要一直往上获取父类。
- 遍历整条继承路径，**注意我们要从Object类开始，从上往下检查**。一方面我们可以从生成的代码中看到，调用表的顺序也是如此；另一方面，这样的检查顺序也符合方法定义和重载的规范。
- 这里，我们定义了两个数据结构：`dispatchTables`和`methodOffsetTable`。前者就是每个类的调用表数据结构，形式为**从类名到<类名.方法名>的映射**。例如：A的调用表中含有<Object.copy()>方法；而后者是每个类的每个方法**在调用表中的偏移**：这是因为在生成的汇编代码中我们需要**通过相**

对于调用表的偏移来访问需要的方法。例如，A的调用表中<Object.abort()>偏移为0，<Object.copy()>偏移为1，即**按照在调用表中排列的顺序，偏移依次增大**。
- 当我们遇到一个方法没有被本类记录时，将其记录到数据结构中；否则，则说明**属于重载**，我们需要找到**被重载的方法**，在它**原来的位置（偏移）上用当前的方法将其替换掉**。

此时我们只是在内部在数据结构上进行了记录，还需要生成实际的代码。对应代码见代码8.6。

- 只需要根据我们的`dispatchTables`，遍历逐条生成代码即可。

3.5 生成Prototype Objects的代码

原型对象的结构在上面已经叙述过，按照其结构逐条生成代码，对应代码见代码8.7。

- 遍历每个类。通过`get_classtag()`获取每个类的class tag，并生成代码。
- 对于size，遍历**继承路径**，将路径上每个类的所有属性个数加起来即为答案。
- 对于调用表，通过`emit_disptable_ref()`生成其引用代码。
- 剩下的是所有属性（包括继承属性）的代码。还是通过**遍历继承路径并考察每个类的属性来完成**。注意这里的遍历顺序**仍然是从Object到当前类，这也是Cool中Object Layout的要求**。
- 这里使用了数据结构`attrOffsetTable`。和`methodOffsetTable`一样，是用来记录**每个属性在对象中的偏移的**，之后在生成的代码中会用于查找需要的属性。注意偏移从3(即12字节)开始。
- 我们还需要对每个属性进行**默认初始化**。对于`Int/Bool/String`这三类，分别设置默认值为`0/false/""`；对于其他类，Cool-manual中指出默认值为`void`，具体实现可以任意。这里我设置默认值为0。

4. 代码部分代码生成

4.1 确定栈结构

PA5 Handout 3.Design中给出的建议是：

> The first pass decides the object layout for each class, particularly the offset at which each attribute is stored in an object. Using this information, the second pass recursively walks each feature and generates stack machine code for each expression.

也就是说，我们**最好先决定栈的分布，再实际生成对应的代码**。我们可以先参考一下官方生成的代码：

```
Object_init:
    addiu   $sp $sp -12
    sw  $fp 12($sp)
    sw  $s0 8($sp)
    sw  $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    move    $a0 $s0
    lw  $fp 12($sp)
    lw  $s0 8($sp)
    lw  $ra 4($sp)
    addiu   $sp $sp 12
    jr  $ra
Main_init:
    addiu   $sp $sp -12
    sw  $fp 12($sp)
    sw  $s0 8($sp)
    sw  $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    jal Object_init
    move    $a0 $s0
    lw  $fp 12($sp)
    lw  $s0 8($sp)
    lw  $ra 4($sp)
    addiu   $sp $sp 12
    jr  $ra
A_init:
    addiu   $sp $sp -12
    sw  $fp 12($sp)
    sw  $s0 8($sp)
    sw  $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
```

- 首先，栈指针`$sp`减12，表示**扩张3个word**。
- 然后，分别在对应的偏移处**保存帧指针`$fp`、SELF寄存器`$s0`和返回地址寄存器`$ra`**。其中，**帧指针是作为“基准”，访问当前调用栈下的参数和变量；SELF寄存器保存当前类object，返回地址寄存器保存调用该方法的返回地址**。
- 将帧指针`$fp`设在`$sp`上方。
- 将`$a0`复制到`$s0`，表示本类。
- 结束处理后，**恢复帧指针、SELF和返回地址**。并跳回返回地址。

可以看到，我们**至少需要在每个方法的栈上保存帧指针、SELF和返回地址**。此外，还有**如下情况涉及到需要在栈上保存寄存器**：

- 如果涉及加法等**二元表达式**，此时**必须把第一个表达式的结果存到栈上**。因为如果我们是把它存到临时寄存器上，**在生成第二个表达式的代码时临时寄存器有可能会被修改**。
- 如果涉及let、case等**引入局部变量的表达式**，同样**需要保存到栈上**。这是因为，它们**可能在后面的表达式中用到，必须保存**。具体可以见5.17 object表达式。
- 如果涉及到**方法的参数**，其同样属于引入的新变量(标识符)，需要保存到栈上。
- 对于属性，我们知道在原型对象中其保存在对象12及以后的偏移上，所以我们**只需要将其保存在`$s0`（SELF）对应的偏移上即可**。

具体来讲按怎样的方案保存到栈上呢？我们看如下示例Cool代码及生成的代码：

```
class A {
    x:Int <- 1;
    y:Int <- 2;
    func(a:Int,b:Int):Int{
        let c:Int in 1 + 2
    };
};
```

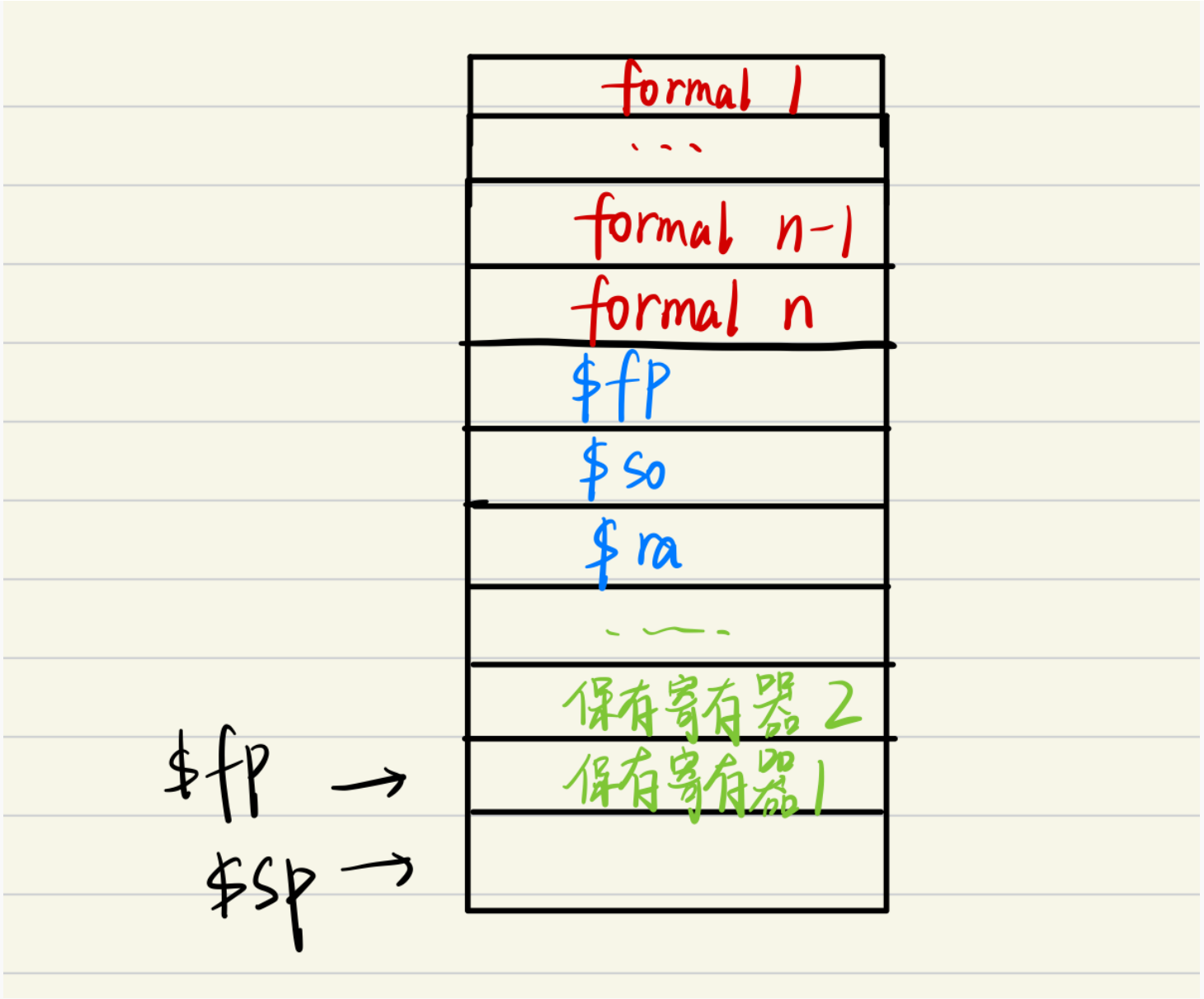
```
A.func:
    addiu   $sp $sp -20
    sw  $fp 20($sp)
    sw  $s0 16($sp)
    sw  $ra 12($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    sw  $s1 4($fp)
    sw  $s2 0($fp)
    la  $s2 int_const2
    la  $s1 int_const0
    la  $a0 int_const1
    jal Object.copy
    lw  $t2 12($a0)
    lw  $t1 12($s1)
    add $t1 $t1 $t2
    sw  $t1 12($a0)
    lw  $s1 4($fp)
    lw  $s2 0($fp)
    lw  $fp 20($sp)
    lw  $s0 16($sp)
    lw  $ra 12($sp)
    addiu   $sp $sp 28
    jr  $ra
```

保存寄存器：c和1

多加的8对应2个参数

- 最开始，栈扩张5个word。这里3个预留给帧指针、SELF和返回地址，剩余两个**保存方法中的两个需要保存的变量：let声明的标识符c和加法表达式的左侧1。**
- 在最后，栈收缩了7个word。多的2个word是**对应的2个参数的位置。**也就是说**参数应该保存到最上面。**至于为什么，我们可以在5.2 dispatch表达式中看到。

大致上我总结栈的分布为：



- 最上面是**参数**。在5.2 dispatch表达式中可知，这些参数**是在dispatch表达式**的时候压入栈的。
- 中间是三个需要保存的寄存器：`$fp`、`$s0`、`ra`。
- 下面是用于保存**临时结果的寄存器**。

4.2 确定栈大小

在确定好栈结构后，我们首先需要知道**每次栈需要扩张多少，收缩多少**。这也就是说我们需要知道**最多有多少个参数和局部变量**。

对于参数，可以在对应方法里通过`get_formals()`简单地计数。但对于局部变量，我们就**必须先对每个feature内部的表达式进行一次遍历，进行统计**。

这里，我通过实现了每个`Expression`类的`check()`方法，通过不断递归检查获取**局部变量最多的个数**，并使用`curLocalNum`和`maxLocalNum`两个全局变量进行计数。在碰到**let、case和二元表达式时，增加一个计数**。具体代码见代码8.8。

4.3 为每个类的初始化方法生成代码

类的初始化方法的作用是：**为属性的初始化表达式生成代码，并为属性赋值。**

首先，在方法开始时我们需要扩张栈，这里首先**check所有属性的初始化表达式，从而获取初始化方法需要的最多需要保存的局部变量个数。**从而扩张栈的word个数为**3 + 最多需要保存的局部变量个数。**

完成4.1描述的准备工作后，就可以开始正式对类进行初始化了。**首先我们需要调用父类的初始化方法，从而初始化继承而来的属性。**因为这个过程**是递归的**，从而可以初始化所有继承而来的属性。然后我们需要具体处理的就只有本类的属性：考虑**遍历本类的属性**，对每个属性，**判断是否存在初始化表达式，如果存在，调用其表达式的生成代码函数**，然后**把初始化的结果装载到对应的属性上。**最后，还需要**将结果`$s0`复制到`$a0`上，便于后续使用。**

- 我们的生成代码函数都把**返回结果放在寄存器`$a0`上。**也就是说初始化表达式的结果也在`$a0`上。
- 利用我们之前构建的`attrOffsetTable`，获取**当前属性在本类中的偏移**，并store到对应的位置。

具体代码见代码8.9。

- 这里`begin_method()`和`end_method()`是自定义的每个方法的通用开头和结尾。普遍来讲，
  - 方法开头需要扩张栈，保存帧指针、SELF和返回地址，将`$a0`复制到`$s0`。
  - 方法结尾需要先恢复帧指针、SELF和返回地址，然后收缩栈。
  - 注意扩张和收缩的大小不一定一致。取决于是否有函数参数，因为**调用的时候是先装入参数再是方法开头。**
- 如果当前类是Basic Class，就不需要进行初始化。

4.4 为每个方法定义生成代码

方法定义即**对方法体表达式进行适当的处理。**这是整个代码生成中最麻烦的部分之一。

首先遍历所有类，然后获取类的所有方法，并对每个方法生成其对应的定义代码。和每个类的初始化方法的处理一样：仍然**首先进行方法体表达式的check，确定栈大小。**然后`begin_method()`。此时我们需要注意的一点是**参数的处理。**类似语义分析中的处理，这里也需要**构建一个类似的SymbolTable来模拟作用域。**在将参数加入到作用域后，才正式进行方法体表达式的生成代码，最后`end_method()`。

这里一个需要思考的问题是：**以怎样的形式构建SymbolTable。**对于参数，我们需要除了需要保存其名字外，现在还需要保存**其相对于帧指针的偏移**，以便在方法体表达式的代码中访问参数。**但注意此时我们并不需要把参数保存到栈上，因为现在的参数只是形式参数，我们需要保存的是实际参数，这在调用时完成。**

具体代码见代码8.10。

- 注意基本类的处理：如果是基本类，我们不需要为其方法生成代码。
- `objectEnvironment`是实现的SymbolTable。其形式为`SymbolTable<Symbol,int>`，记录标识符及其对应的偏移。

- 与语义分析中实现的SymbolTable不一样，现在我们并不记录属性。这是因为**属性可以通过`$s0`访问，而不是`$fp`**。
- SymbolTable中记录的有参数和临时变量。可以看到**参数偏移起始位置是最大临时变量个数+3**，说明**参数被中间3个需要保存的寄存器隔开来了**，后续的处理中也贯彻这一点，因此不存在记录冲突的情况。

5. 各表达式代码的生成

各表达式代码的生成才涉及到主要语义的实现。参考cool manual 13. Operational Semantics中给出的规范。

5.1 assign表达式

assign表达式需要：

- 调用右侧表达式的`code()`方法；
- 查找左侧标识符的定义：可能是参数、临时变量或属性。我们**先在`objectEnvironment`中查找是否是参数或临时变量，并获取偏移**，获取`$fp`相应内容；否则就是属性，通过`attrOffsetTable`获取偏移，并获取`$s0`对应内容。

代码实现见代码8.11。

5.2 dispatch表达式

dispatch表达式需要：

- 将实际参数存入栈中。这也对应了**含有参数时，栈扩张比栈收缩的word少**。这里少的部分就是**参数这部分**，因为**顺序是先调用，再跳到对应方法执行**；
- 注意**参数入栈的顺序是从左到右**。这也对应了栈上参数的分布是**更前面的参数放在栈的更下面**，见cool-runtime 4.Calling Convetions；
- 调用左侧表达式的`code()`方法；
- 判断**左侧表达式是否为void**，即**通过与0比较来决定**。这里是通过`emit_bne()`和`emit_label_def()`来完成的。如果表达式是void，则属于**Dispatch on void 错误**，我们**需要调用`_dispatch_abort`方法**，并在这之前，**将文件名记录到`$a0`，错误行号记录到`$t1`**。具体要求见cool-runtime 6.Runtime Interface；
- 表达式不是void的话，则**需要查找该类的调用表，并根据方法偏移找到需要的方法**。具体实现来讲，首先**从当前类偏移8的地方取出该类的调用表**，然后**根据`methodOffsetTable`找到当前方法在当**

前类的偏移，最后从**调用表中对应偏移的地方获取需要方法的地址**；
- 需要注意，如果表达式是SELF_TYPE的话需要转换为当前类；
- 通过`emit_jalr`，即跳到寄存器中地址的方式来跳转到对应方法。

具体代码见代码8.12。可以看到，这里实际上就实现了**动态绑定**，即**实际调用的方法取决于调用主体类**。

5.3 static_dispatch表达式

static_dispatch的基本实现逻辑和dispatch一致。主要有以下几个区别：

- 现在**并不需要获取当前类的调用表，而是获取声明类的调用表**。这里实现思路略有不一样，因为我们不能通过object来获取其中的调用表了。此时只能**通过直接指定声明类调用表的地址**。这是通过字符串拼接来完成的，构成`[调用类]_dispTab`的形式，然后通过`emit_load_address()`直接获取该调用表。

具体代码见代码8.13。

5.4 if表达式

if表达式需要：

- 生成介词表达式的代码；
- 根据介词表达式的结果，**从中取出Bool值**，即取出该Bool object偏移12的内容；
- 将取出的Bool值与0比较，然后设置相应的label，并在不同的label下执行对应表达式的生成代码；
- 这里，label是一个全局变量，每生成一个label就增加1。

具体代码见代码8.14。

5.5 while loop表达式

while表达式需要：

- 设定好循环入口和出口的label；
- 仍然生成介词表达式的代码，并**从中取出Bool值**；
- 根据Bool值与0的比较结果，决定是否跳出；
- 如果不跳出，就**生成循环体表达式的代码**，并跳回循环入口label；
- 最后跳出后，**需要把结果标记为void，即**`$a0 = 0`。这在cool manual 13. Operational Semantics中有相应要求。

具体代码见代码8.15。

5.6 case表达式

case表达式是生成代码中最复杂的一个表达式。具体要求如下：

- 首先生成**表达式的代码**；
- 判断是否为void。如果是，则为**case on void错误**。需要将文件名记录到`$a0`，行号记录到`$t1`，并通过`emit_jalr`调用`_case_abort2`方法；
- 如果非void，首先**我们需要将生成表达式的内容存到栈上**。
  - 这里我是通过一个全局变量`curLocalNum`来决定其在栈上的偏移的；
  - 需要注意：**case的所有branch声明的标识符对应的都是这个表达式object，也即case表达式虽然有很多branch，但实际上只对应一个临时变量**。
- 如何判断**要选取哪个branch**？cool manual 7.9 Case中做出了规定：**选择最近的当前表达式的祖先类对应的分支执行**。因此在生成的代码中就涉及到**如何在汇编代码中找到这样的祖先类**。
- 首先，我们**按照class tag从大到小的顺序**对所有branch进行排列。因为我们只能**依次遍历每个branch，遍历到正确的branch后就直接执行并不再往下继续**。因此，我们应该**先检查那些相对更近的类，也即class tag更小的**。
- 然后，**如何判断一个类是否是另一个类的子类**？这需要在汇编代码上完成，而我们只能**利用class tag**。因为class tag**按照先序排列**，从而有：

$$A \leq B \iff b \leq a \leq b + k$$

这里，A和B是两个类，a和b是对应的class tag，而k是**B的子孙个数**。从直观上，也容易证明这个式子的正确性。从而将子类的判断转换为了**class tag与立即数之间的比较**。

- 如何获取某个类的子孙数？通过简单的**后序遍历**即可。
- 对于每个branch，生成对应的label，在其**object偏移0处取出class tag**。通过`emit_blti()`和`emit_bgti()`来确定表达式类的class tag是否在对应范围之间。
  - 如果不在区间内，则**直接跳到下一个branch**的label。
  - 如果在区间内，**生成对应表达式的代码，并跳到结束label**。
  - 在执行某个分支表达式时，还需要**设定环境表**，把分支声明标识符加入到SymbolTable。
- 如果没有分支匹配，则为**没有分支匹配错误**。需要执行`_case_abort`。

具体代码见代码8.16。

5.7 block表达式

block表达式需要：

- 直接生成每个表达式的代码；
- 最后的结果是最后一个表达式的结果，正好也存放在`$a0`中。

具体代码见代码8.17。

5.8 let表达式

let表达式需要：

- 对声明的标识符进行**初始化**。如果没有初始化表达式，进行**默认初始化**。这里需要对Int/String/Bool做特殊处理。
- 设置**环境表**。将声明的标识符记录到SymbolTable中。
- 将声明标识符对应的object存到栈上对应偏移位置。
- 生成let内部表达式的代码。

具体代码见代码8.18。

5.9 算术表达式

算术表达式需要：

- 生成第一个表达式的代码。
- **将第一个表达式的结果保存到栈上，并增加当前临时变量计数**。这一步非常重要。
- 生成第二个表达式的代码。
- **调用`object.copy`，复制一个新Int object作为结果对象**。
- 从两个表达式对应的object中**取出int值（偏移12处）**。
- 生成对应运算的代码，并把结果保存回`$a0`。

具体代码见代码8.19。

5.10 比较表达式

比较表达式需要：

- 生成第一个表达式的代码。
- 同样地将第一个表达式结果存栈。
- 生成第二个表达式的代码。
- 从两个表达式对应的object中取出int值。
- 生成对应比较的代码，并设定相应的label。

- 根据比较情况，`$a0`的结果为`truebool`或`falsebool`。

具体代码见代码8.20。

5.11 等于表达式

等于表达式需要：

- 生成第一个表达式的代码。
- 将第一个表达式结果存栈。
- 生成第二个表达式的代码。
- **首先直接比较两个表达式的object，即比较地址。**
- 如果地址相等，**直接相等，返回true。**
- 否则，**调用**`equality_test`，此时按照要求，**令**`$a0 = true, $a1 = false`。

具体代码见代码8.21。

5.12 取反表达式

取反表达式需要：

- 生成表达式代码。
- 通过`object.copy`，复制Int object作为结果对象。
- 取出表达式的Int值(偏移12)，然后直接取反。
- 将结果存到`$a0`。

具体代码见代码8.22。

5.13 取补表达式

取补表达式和取反表达式的实现基本一致。区别在于：**取补表达式不需要复制一份。**这是因为实际实现的时候，**我们是通过比较表达式的Bool值，来直接返回true或false的，而不需要基于对表达式原本的修改。**

具体代码见代码8.23。

5.14 new表达式

new表达式比较复杂，其要求为：

- 分为两种情况：**new SELF_TYPE 和 new 其他类**。
- 如果是new 其他类，则**首先获取该类的原型对象**，并**通过`Object.copy()`复制一份**。最后，再找到该类的初始化方法进行初始化。
- 如果是new SELF_TYPE，这种情况比较麻烦：
  - 首先，我们需要知道**SELF_TYPE指代哪个类**，从而此时**需要取出当前类的class tag**，然后**以class tag作为偏移索引，在`class_objTab`中查找此类对应的原型对象和初始化方法，并用类似的方法复制、初始化**。
  - 这里我们通过`emit_sll`生成代码将class tag * 8 作为偏移。

具体代码见代码8.24。

5.15 isvoid表达式

isvoid表达式需要：

- 生成表达式代码。
- 检查**表达式object是否为void(即0)**。
- 如果为void返回true，否则false。

具体代码见代码8.25。

5.16 no_expr表达式

no_expr表达式**直接令`$a0`为0(即void)即可**。
具体代码见代码8.26。

5.17 object表达式

object表达式需要：

- 如果是self，直接返回`$s0`，即`$a0 = $s0`。
- 和assign一样，先在`objectEnvironment`中查找。
- 没找到再到`attrOffsetTable`中查找。

具体代码见代码8.27。

6. 问题与解决

6.1 为什么我们需要确定栈结构?

最开始我的想法是，**碰到临时变量时就压入栈中**，用完**再出栈**。这样实现其实也不会有很大问题。但主要有两方面的隐患：

- 在进入方法时，没有确定栈的结构，从而在**不同的选择分支上栈的大小可能不一样**。这与cool-runtime 5. Register and Calling Conventions中的要求可能不一致。
- 另一方面，临时压栈的方式生成的代码可读性较差，**从调试的角度来看**较难辨别栈情况。

确定栈结构也是参照了官方的代码结构：我是在看了官方生成的代码，通过多次测试后确定的其栈结构。这样实现下来虽然需要一整次的遍历，但结构清晰了不少。

6.2 二元表达式的临时保存

最开始我并没有对加法等二元表达式进行特殊处理，只是简单的把两个表达式结果放在两个寄存器`$t1`、`$t2`中。但这样问题很大，例如：

```
1 + (2 + 3)
```

如果不把结果存栈，那么**第一个表达式的结果1就会被计算2+3的第一个表达式的结果2覆盖掉，从而计算出错误的结果**。也就是说，**只要涉及二元表达式，我们都需要把第一个表达式的结果存栈。**

6.3 属性初始表达式的check

最初我只在method definition处对每个method进行了check。但在一个测试中直接报了`bad address`的错误：

```
class A{
    a:Int <- 2 + 3;
    ...
}
```

我花了很久也没看出哪里有问题。后来我仔细又对比了一遍我的生成代码和官方代码，总是发现了区别：**我的代码栈扩张只有12，比官方的少4**。这里就是因为**没有对属性的初始化表达式也进行检查**，从

而**使得`$fp`这个位置同时用来存放`$ra`和第一个表达式的结果**。这样，返回地址**就被覆盖掉了**，返回的时候返回了一个错乱的地址，从而提示`bad address`。

6.4 callee saved寄存器与优化相关

我的代码基本参考官方生成代码的结果，逐条对照分析写成。但观察就能看到：**官方代码中多次用到了`$s1`、`$s2`等寄存器**。这些寄存器是**Callee saved**，即被调用者保存，意思是说**进入方法时保存这些寄存器，然后这些寄存器就可以随意使用了。当结束方法时，恢复这些寄存器**。

可以看到，callee saved寄存器很好地模拟了**调用栈的临时变量变化情况**。比起我的代码，有如下优势：

- 使用callee saved 寄存器，对于**临时变量就可以将其保存在这些寄存器中，从而在需要使用时不需要像我一样每次都需要读存储器，而可以直接使用该寄存器，从而效率更高**。
- 通过callee saved 寄存器，使得临时变量的使用更加层次分明，并保证了绝不混淆，从**健壮性上**考虑，比我依靠`curLocalNum`来管理要好不少。

最开始我也尝试过实现callee saved寄存器，但由于以下原因放弃了。

- 我们只能使用`$s1`到`$s6`这几个寄存器。如果涉及更多临时变量，就需要跟我一样读存储器。这种区分式的实现比较麻烦。
- callee saved寄存器的实现更偏向于代码生成之后的**优化**领域内容。观察代码也能看出官方生成的代码有一定优化。

7. 总结与感想

本次实验是所有实验中最难的一次。一方面是因为**各手册给的信息都很模糊，只给了大致的方向但没有具体的描述**，导致初步接触时完全摸不着头脑；另一方面，本次实验在逻辑上也有不少难以简单想到的实现，如case表达式通过class tag和子孙数目来比较和确定branch、如何确定栈结构及保存临时变量和结果等。从代码量上来看工作量也十分夸张，我自己的代码总行数达到了2100行之多。

整体实现完这个实验后，我认为最有帮助的**一定是官方生成的代码参照，比所有文档加起来都有用的多**。首先在看到官方后**可以在心里对生成哼的代码结构有一个明晰的把握**；其次是通过官方生成的代码，能够**掌握生成代码需要的细节和逻辑**。

关于测试，我这边是通过一个评级脚本提供的几十个测试样例来逐一生成代码并在SPIM上测试运行结果的。它们非常有用，我自己构造样例难以检查出的错误在这里就检查出了五六个错误。在一步步检查、修正这些错误的过程中，我对整个代码生成和汇编代码逻辑也有了更深的理解。

最后，因为测试样例过多，这里也不给出全部样例和结果了。就以我在PA1中所写的`stack.cl`为例，展示所产生的结果。代码和测试结果分别见代码8.28和测试结果8.29。此外，还完成了**垃圾回收的测**

试。测试结果见测试结果8.30。最后，我**将前面词法分析、语法分析和语义分析的模块也用自己前面PA的实现替换，真正实现了从0到Cool编译器**。测试结果见测试结果8.31。

在代码生成的基础上，我们还可以思考：

- 我跟官方的代码有什么不同？有什么缺点？
- 如果要进行读写效率、代码行数和健壮性上的优化，该朝着怎样的方向前进？
- ……

8. 代码与测试结果

8.1 String常量代码生成

```
void StringEntry::code_def(ostream &s, int stringclasstag)
{
    IntEntryP lensym = inttable.add_int(len);

    // Add -1 eye catcher
    s << WORD << "-1" << endl;

    code_ref(s);
    s << LABEL                                                // label
      << WORD << stringclasstag << endl                       // tag
      << WORD << (DEFAULT_OBJFIELDS + STRING_SLOTS + (len + 4) / 4) << endl // size
      << WORD;

    /***** Add dispatch information for class String ******/
    // we just need to declare.
    emit_disptable_ref(Str, s);

    s << endl; // dispatch table
    s << WORD;
    lensym->code_ref(s);
    s << endl;                    // string length
    emit_string_constant(s, str); // ascii string
    s << ALIGN;                   // align to word
}
```

8.2 分配class tag

```
void CgenClassTable::install_classtags()
{
    int tag = 0;
    CgenNodeP node = lookup(Object);
    dfs(node, tag);
}

void CgenClassTable::dfs(CgenNodeP node, int &tag)
{
    // assign current tag to this node
    node->set_classtag(tag);
    tag_to_CgenNode[tag] = node;

    // need to handle String/Int/Bool
    Symbol name = node->get_name();
    if (name == Str)
        stringclasstag = tag;
    else if (name == Bool)
        boolclasstag = tag;
    else if (name == Int)
        intclasstag = tag;
    tag += 1;
    // dfs for all children
    for (auto child = node->get_children(); child; child = child->tl())
        dfs(child->hd(), tag);
}
```

8.3 生成 Class name table 代码

```
// A table, which at index (class tag) * 4 contains a pointer
// to a String object containing the name of the class associated
void CgenClassTable::code_class_nameTab()
{
    // we just need to traverse the class then emit code
    // notice the sequence: the tag from 0 to max.
    // i.e. the preorder of the AST

    // here we first declare the class name tab LABEL
    str << CLASSNAMETAB << LABEL;

    for (int tag = 0; tag < int(tag_to_CgenNode.size()); ++tag)
    {
        // the nameTab should be like:
        // class_nameTab:
        //      .word   str_const1
        //      .word   str_const2
        //      ...
        Symbol name = tag_to_CgenNode[tag]->get_name();
        auto str_entry = stringtable.lookup_string(name->get_string());
        str << WORD;
        str_entry->code_ref(str);
        str << endl;
    }
}
```

8.4 生成 Class object table 代码

```
// A table, which at index (class tag) * 8 contains a pointer to
// the prototype object and at index (class tag) * 8 + 4 contains
// a pointer to the initialization method for that class.
void CgenClassTable::code_class_objTab()
{
    // the same construction as nameTab.
    str << CLASSOBJTAB << LABEL;

    for (int tag = 0; tag < int(tag_to_CgenNode.size()); ++tag)
    {
        // the objTab should be like:
        // class_objTab:
        //      .word   Object_protObj
        //      .word   Object_init
        //      ...
        Symbol name = tag_to_CgenNode[tag]->get_name();
        str << WORD;
        emit_protobj_ref(name, str);
        str << endl;

        str << WORD;
        emit_init_ref(name, str);
        str << endl;
    }
}
```

8.5 构建方法调用表和方法偏移表

```
std::vector<CgenNodeP> CgenNode::getInheritancePath()
{
    // recursively get its parent,until no_class
    // there will be no semantic error, of course!
    std::vector<CgenNodeP> path;
    auto node = this;
    while (true)
    {
        path.push_back(node);
        if (node->get_name() == Object)
            break;
        node = node->get_parentnd();
    }
    return path;
}

void CgenClassTable::install_attrs_and_methods()
{
    CgenNodeP node;
    Features features;
    // traverse the class list
    for (auto nodelist = nds; nodelist; nodelist = nodelist->tl())
    {
        node = nodelist->hd();
        features = node->get_features();

        // traverse this class's features
        for (int i = features->first(); features->more(i); i = features->next(i))
        {
            auto feature = features->nth(i);
            if (feature->is_method())
                methodTable[node->get_name()].push_back((method_class *)feature);
            else
                attrTable[node->get_name()].push_back((attr_class *)feature);
        }
    }

    // after installing the methodTable and attrTable
    // now check the whole path for each class(CgenNode)
```

```
    for (auto nodelist = nds; nodelist; nodelist = nodelist->tl())
    {
        node = nodelist->hd();
        auto cur_classname = node->get_name();
        auto path = node->getInheritancePath();

        // path is from Object to this class
        for (auto it = path.rbegin(); it != path.rend(); ++it)
        {
            auto class_name = (*it)->get_name();
            auto methods = methodTable[(*it)->get_name()];
            for (auto method : methods)
            {
                if (methodOffsetTable[cur_classname].find(method->name)
                    == methodOffsetTable[cur_classname].end())
                {
                    dispatchTables[cur_classname].push_back
                                    ({class_name, method->name});
                    methodOffsetTable[cur_classname][method->name] =
                                    dispatchTables[cur_classname].size() - 1;
                }
                // if find, need to override
                // change cur class's this offset method into a newer version
                else
                {
                    auto offset = methodOffsetTable[cur_classname][method->name];
                    // override
                    dispatchTables[cur_classname][offset] =
                                    {class_name, method->name};
                }
            }
        }
    }
}
```

8.6 生成 Dispatch tables 代码

```
void CgenClassTable::code_object_dispatchTabs()
{
    // the dispatchTable for class A should be like:
    // A_dispatchtab:
    // Object.abort()
    // ...
    // A.func()
    // ...
    for (auto it = dispatchTables.begin(); it != dispatchTables.end(); ++it)
    {
        auto cur_name = it->first;
        auto dispatchTable = it->second;
        emit_disptable_ref(cur_name, str);
        str << LABEL;
        for (auto dit = dispatchTable.begin(); dit != dispatchTable.end(); ++dit)
        {
            str << WORD;
            auto class_name = dit->first;
            auto method_name = dit->second;
            emit_method_ref(class_name, method_name, str);
            str << endl;
        }
    }
}
```

8.7 生成 Prototype Objects 代码

```
void CgenClassTable::code_proto_objects()
{
    // layout of a prototype:
    // -4 Garbage Collector Tag
    // 0 Class Tag
    // 4 Object Size(in word)
    // 8 Dispatch Table
    // >=12 Attributes (from ancestor's to current class's
    for (auto nodelist = nds; nodelist; nodelist = nodelist->tl())
    {
        auto node = nodelist->hd();
        auto cur_name = node->get_name();
        // this is Garbage Collector Tag.
        str << WORD << -1 << endl;
        // should be like: Object_protObj
        emit_protobj_ref(cur_name, str);
        str << LABEL;
        // this is Class Tag
        str << WORD << node->get_classtag() << endl;
        int objSize = DEFAULT_OBJFIELDS;
        // traverse the attrs, including inherited attrs!
        auto path = node->getInheritancePath();
        for (auto it = path.begin(); it != path.end(); ++it)
        {
            auto class_name = (*it)->get_name();
            auto attrs = attrTable[class_name];
            objSize += attrs.size();
        }

        // output the object size
        str << WORD << objSize << endl;

        // next is the dispatch pointer
        // only need to generate a reference
        str << WORD;
        emit_disptable_ref(cur_name, str);
        str << endl;

        // the rest is the attrs. From Object to current class!
```

```
        // attr start from 12 (in word size is 3)
        int attr_pos = DEFAULT_OBJFIELDS;
        for (auto it = path.rbegin(); it != path.rend(); ++it)
        {
            auto class_name = (*it)->get_name();
            auto attrs = attrTable[class_name];
            for (auto attr : attrs)
            {
                // By the way we can install the attrOffsetTable
                // it is used to look for an attr's position(offset) in a class
                attrOffsetTable[cur_name][attr->name] = attr_pos;
                ++attr_pos;
                str << WORD;
                // we need to set the default value of the attr
                // especially notice Int/Bool/String

                // String: default ""
                if (attr->type_decl == Str)
                {
                    auto string_entry = stringtable.lookup_string("");
                    string_entry->code_ref(str);
                }
                // Bool: default false
                else if (attr->type_decl == Bool)
                    falsebool.code_ref(str);
                // Int: default 0
                else if (attr->type_decl == Int)
                {
                    auto int_entry = inttable.lookup_string("0");
                    int_entry->code_ref(str);
                }
                // other types(may be modified...)
                else
                    str << 0;
                str << endl;
            }
        }
    }
}
```

8.8 各表达式的check

```
void assign_class::check()
{
    expr->check();
}

void static_dispatch_class::check()
{
    expr->check();
    for (int i = actual->first(); actual->more(i); i = actual->next(i))
        actual->nth(i)->check();
}

void dispatch_class::check()
{
    expr->check();
    for (int i = actual->first(); actual->more(i); i = actual->next(i))
        actual->nth(i)->check();
}

void cond_class::check()
{
    pred->check();
    then_exp->check();
    else_exp->check();
}

void loop_class::check()
{
    pred->check();
    body->check();
}

void typcase_class::check()
{
    expr->check();
    for (int i = cases->first(); cases->more(i); i = cases->next(i))
    {
        auto branch = (branch_class *)cases->nth(i);
        // change local num
```

```
        ++currLocalNum;
        maxLocalNum = std::max(maxLocalNum, currLocalNum);
        branch->expr->check();
        --currLocalNum;
    }
}

void block_class::check()
{
    for (int i = body->first(); body->more(i); i = body->next(i))
        body->nth(i)->check();
}

void let_class::check()
{
    // change local num
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    init->check();
    body->check();
    --currLocalNum;
}

void plus_class::check()
{
    // change local num
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    e1->check();
    e2->check();
    --currLocalNum;
}

void sub_class::check()
{
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    e1->check();
    e2->check();
    --currLocalNum;
}
```

```
void mul_class::check()
{
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    e1->check();
    e2->check();
    --currLocalNum;
}

void divide_class::check()
{
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    e1->check();
    e2->check();
    --currLocalNum;
}

void neg_class::check()
{
    e1->check();
}

void lt_class::check()
{
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    e1->check();
    e2->check();
    --currLocalNum;
}

void eq_class::check()
{
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    e1->check();
    e2->check();
    --currLocalNum;
}

void leq_class::check()
```

```
{
    ++currLocalNum;
    maxLocalNum = std::max(maxLocalNum, currLocalNum);
    e1->check();
    e2->check();
    --currLocalNum;
}

void comp_class::check()
{
    e1->check();
}

void int_const_class::check()
{
    return;
}

void string_const_class::check()
{
    return;
}

void bool_const_class::check()
{
    return;
}

void new__class::check()
{
    return;
}

void isvoid_class::check()
{
    e1->check();
}

void no_expr_class::check()
{
    return;
}
```

```
void object_class::check()
{
    return;
}
```

8.9 类的初始化方法

```
void begin_method(int local_num, ostream &s)
{
    int total_bytes = (local_num + 3) * WORD_SIZE;
    // 12 bytes for: old fp,$s0,$ra
    emit_addiu(SP, SP, -total_bytes, s);
    emit_store(FP, local_num + 3, SP, s);   // store old fp
    emit_store(SELF, local_num + 2, SP, s); // store $s0
    emit_store(RA, local_num + 1, SP, s);   // store $ra

    // set current $fp
    emit_addiu(FP, SP, WORD_SIZE, s);
    // save ACC to SELF
    emit_move(SELF, ACC, s);
}

void end_method(int formal_num, int local_num, ostream &s)
{
    int total_bytes = (formal_num + local_num + 3) * WORD_SIZE;
    emit_load(FP, local_num + 3, SP, s);   // restore $fp
    emit_load(SELF, local_num + 2, SP, s); // restore $s0
    emit_load(RA, local_num + 1, SP, s);   // restore $ra

    // restore $sp
    emit_addiu(SP, SP, total_bytes, s);
    emit_return(s); // equals to: jr $ra
}

// this is used to init the attributes of a class(including inherited attrs)
// note we may need to dispatch ancestor's init method
void CgenClassTable::code_object_inits()
{
    // check each class
    for (auto nodelist = nds; nodelist; nodelist = nodelist->tl())
    {
        curr_class = nodelist->hd();
        emit_init_ref(curr_class->get_name(), str);
        str << LABEL;

        maxLocalNum = 0;
```

```
        if (!curr_class->basic())
        {
            auto attrs = attrTable[curr_class->get_name()];
            // first pass: determine the stack layout of init method
            for (auto attr : attrs)
            {
                if (attr->init->get_type())
                    attr->init->check();
            }
        }

        // prepare for method begin
        // init method has no formals
        begin_method(maxLocalNum, str);
        // handle parent class
        // diaspatch parent's init method
        // notice this should be recursive.So only dispatch once here
        auto parent_class = curr_class->get_parentnd();
        if (parent_class->get_name() != No_class)
        {
            str << JAL;
            emit_init_ref(parent_class->get_name(), str);
            str << endl;
        }

        // handle current class
        // This is the real part of handling attr init
        if (!curr_class->basic())
        {
            auto attrs = attrTable[curr_class->get_name()];
            for (auto attr : attrs)
            {
                // when no init, do not need to store into $s0
                // notice: No_expression's type is NULL,not No_type
                if (attr->init->get_type())
                {
                    attr->init->code(str); // expr object in $a0
                    int attr_offset =
                        attrOffsetTable[curr_class->get_name()][attr->name];
                    emit_store(ACC, attr_offset, SELF, str);
                }
            }
```

```
        }
        emit_move(ACC, SELF, str);
        end_method(0, maxLocalNum, str);
    }
}
```

8.10 生成每个方法的定义代码

```
void CgenClassTable::code_methods()
{
    for (auto nodelist = nds; nodelist; nodelist = nodelist->tl())
    {
        curr_class = nodelist->hd();
        if (curr_class->basic())
            continue;

        // only handle current class's methods
        auto methods = methodTable[curr_class->get_name()];
        for (auto method : methods)
        {
            // get the max local num to clarify the layout of method stack
            maxLocalNum = 0;
            // check the method to determine the stack layout
            method->expr->check();
            emit_method_ref(curr_class->get_name(), method->name, str);
            str << LABEL;
            auto formals = method->formals;
            begin_method(maxLocalNum, str);
            // put formal into stack in reverse orders
            // don't need to emit code for formals
            int formal_offset = maxLocalNum + 3;
            objectEnvironment.enterscope();
            for (int i = formals->len() - 1; i >= 0; --i)
            {
                auto formal = formals->nth(i);
                objectEnvironment.addid(formal->get_name(), new int(formal_offset));
                ++formal_offset;
            }

            // method body code
            method->expr->code(str);
            objectEnvironment.exitscope();
            end_method(formals->len(), maxLocalNum, str);
        }
    }
}
```

8.11 生成assign表达式的代码

```
void assign_class::code(ostream &s)
{
    expr->code(s);
    if (objectEnvironment.lookup(name) != NULL)
    {
        int offset = *(objectEnvironment.lookup(name));
        emit_store(ACC, offset, FP, s);
        return;
    }

    int offset = codegen_classtable->
            attrOffsetTable[codegen_classtable->curr_class->get_name()][name];
    emit_store(ACC, offset, SELF, s);
}
```

8.12 生成dispatch表达式的代码

```
void dispatch_class::code(ostream &s)
{
    // first the formals, in order
    for (int i = actual->first(); actual->more(i); i = actual->next(i))
    {
        auto cur_actual = actual->nth(i);
        cur_actual->code(s); // $a0 stores this actual object
        // push into stack
        emit_push(ACC, s);
    }
    // next LHS expr
    expr->code(s); // $a0 stores the dispatcher object
    // we should test if the dispatcher is NULL
    int nonvoid_label = get_label();
    emit_bne(ACC, ZERO, nonvoid_label, s);
    // this section is for dispatch on void
    // need to prepare runtime error information
    // see cool-runtime

    // store filename in $a0
    emit_load_string(ACC, stringtable.lookup_string
        (codegen_classtable->curr_class->filename->get_string()), s);
    // this is the line number
    emit_load_imm(T1, this->line_number, s);
    // immediately abort
    emit_jal("_dispatch_abort", s);

    // this section is for normal dispatch
    emit_label_def(nonvoid_label, s);
    emit_load(T1, DISPTABLE_OFFSET, ACC, s);
    Symbol expr_type = expr->type;
    if (expr_type == SELF_TYPE)
        expr_type = codegen_classtable->curr_class->get_name();
    // now find the corresponding method in dispatch table
    int offset = codegen_classtable->methodOffsetTable[expr_type][name];
    emit_load(T1, offset, T1, s); // now $t1 stores the pointer to the method we need
    emit_jalr(T1, s);             // dispatch the method
}
```

8.13 static_dispatch表达式的代码生成

```
void static_dispatch_class::code(ostream &s)
{
    // first the formals, in order
    for (int i = actual->first(); actual->more(i); i = actual->next(i))
    {
        auto cur_actual = actual->nth(i);
        cur_actual->code(s); // $a0 stores this actual object
        // push into stack
        emit_push(ACC, s);
    }
    // next LHS expr
    expr->code(s); // $a0 stores the dispatcher object
    // we should test if the dispatcher is NULL
    int nonvoid_label = get_label();
    emit_bne(ACC, ZERO, nonvoid_label, s);
    // this section is for dispatch on void
    // need to prepare runtime error information
    // see cool-runtime

    // store filename in $a0
    emit_load_string(ACC, stringtable.lookup_string
        (codegen_classtable->curr_class->filename->get_string()), s);
    // this is the line number
    emit_load_imm(T1, this->line_number, s);
    // immediately abort
    emit_jal("_dispatch_abort", s);
    emit_label_def(nonvoid_label, s);


    std::string static_dispatchTable = type_name->get_string()
                        + std::string(DISPTAB_SUFFIX);
    emit_load_address(T1, (char *)static_dispatchTable.c_str(), s);

    // now find the corresponding method in dispatch table (of given type)
    int offset = codegen_classtable->methodOffsetTable[type_name][name];
    emit_load(T1, offset, T1, s); // now $t1 stores the pointer to the method we need
    emit_jalr(T1, s);             // dispatch the method
}
```

8.14 if表达式的代码生成

```
void cond_class::code(ostream &s)
{
    pred->code(s);
    emit_fetch_int(T1, ACC, s); // $t1 now stores the bool val

    int false_label = get_label();
    int end_label = get_label();
    emit_beqz(T1, false_label, s);
    // the section is for true
    then_exp->code(s);
    emit_branch(end_label, s);
    // the section is for false
    emit_label_def(false_label, s);
    else_exp->code(s);

    // end
    emit_label_def(end_label, s);
}
```

8.15 while表达式的代码生成

```
void loop_class::code(ostream &s)
{
    int test_label = get_label();
    int out_label = get_label();

    // test the condition
    emit_label_def(test_label, s);
    pred->code(s);
    emit_fetch_int(T1, ACC, s); // get the boolean value
    emit_beqz(T1, out_label, s);
    // if satisfy the condition
    // do the body code
    body->code(s);
    // back to condition test
    emit_branch(test_label, s);

    // this is the out
    emit_label_def(out_label, s);
    // notice: we need to make $a0 void
    emit_move(ACC, ZERO, s);
}
```

8.16 case表达式的代码生成

```
int get_children_num(CgenNodeP node)
{
    int num = 0;
    for (auto child = node->get_children(); child; child = child->tl())
        num += 1 + get_children_num(child->hd());
    return num;
}

void typcase_class::code(ostream &s)
{
    expr->code(s);
    // check if type is void
    int end_label = get_label();
    int nonvoid_label = get_label();
    emit_bne(ACC, ZERO, nonvoid_label, s);
    // this section is for void handles
    emit_load_string(ACC, stringtable.lookup_string
        (codegen_classtable->curr_class->get_filename()->get_string()), s);
    emit_load_imm(T1, this->get_line_number(), s);
    emit_jal("_case_abort2", s);

    // this section is for nonvoid

    emit_label_def(nonvoid_label, s);
    // now expr object in $a0
    // first we should store it in stack
    emit_store(ACC, currLocalNum, FP, s);
    // fetch its class tag
    emit_load(T2, 0, ACC, s);

    // first sort the branches by class tag
    // then one by one check
    std::vector<std::pair<branch_class *, int>> temp;
    for (int i = cases->first(); cases->more(i); i = cases->next(i))
    {
        auto branch = (branch_class *)cases->nth(i);
        // get classtag
        int tag = codegen_classtable->lookup(branch->type_decl)->get_classtag();
        temp.push_back({branch, tag});
```

```
    }

    // sort by classtag
    // from large to small
    sort(temp.begin(), temp.end(),
    [](const std::pair<branch_class *, int> A, const std::pair<branch_class *, int> B)
         { return A.second > B.second; });

    int next_branch_label;
    // now check each branch
    for (auto it = temp.begin(); it != temp.end(); ++it)
    {
        auto branch = it->first;
        auto tag = it->second;
        if (cgen_debug)
            cerr << branch->name << ' ' << tag << endl;

        int childrenNum = get_children_num(codegen_classtable->tag_to_CgenNode[tag]);

        if (it != temp.begin())
            emit_label_def(next_branch_label, s);

        next_branch_label = get_label();
        // if Class A is Class B's subclass
        // then: suppose class tag of A is a, class tag of B is b
        // A <= B iff b <= a <= b + k, where k is childrenNum of b
        emit_blti(T2, tag, next_branch_label, s);
        emit_bgti(T2, tag + childrenNum, next_branch_label, s);
        // this section is for execting this branch
        objectEnvironment.enterscope();
        objectEnvironment.addid(branch->name, new int(currLocalNum));
        ++currLocalNum;
        branch->expr->code(s);
        --currLocalNum;
        objectEnvironment.exitscope();
        // after choosing one branch,immediately jump to end
        emit_branch(end_label, s);
    }

    // this is for no match error
    emit_label_def(next_branch_label, s);
    emit_jal("_case_abort", s);
```

```
    // end
    emit_label_def(end_label, s);
}
```

8.17 block表达式的代码生成

```
void block_class::code(ostream &s)
{
    for (int i = body->first(); body->more(i); i = body->next(i))
    {
        auto expr = body->nth(i);
        expr->code(s);
    }
    // finally $a0 stores last expression's result
}
```

8.18 let表达式的代码生成

```
void let_class::code(ostream &s)
{
    // if there is no init
    if (init->get_type() == NULL)
    {
        // specially handle Int/Bool/String
        // assign their default value
        if (type_decl == Int)
            emit_load_int(ACC, inttable.lookup_string("0"), s);
        else if (type_decl == Str)
            emit_load_string(ACC, stringtable.lookup_string(""), s);
        else if (type_decl == Bool)
            emit_load_bool(ACC, falsebool, s);
        else
            init->code(s);
    }
    else
        init->code(s); // init object in $a0

    objectEnvironment.enterscope();
    objectEnvironment.addid(identifier, new int(currLocalNum));
    // official implemetation use callee saved registers
    // for our implemetation, it is to be improved

    // save init to corresponding position
    emit_store(ACC, currLocalNum, FP, s);
    ++currLocalNum;
    body->code(s);
    --currLocalNum;
    objectEnvironment.exitscope();
}
```

8.19 算术表达式的代码生成

```
void plus_class::code(ostream &s)
{
    e1->code(s);
    // save result in stack
    emit_store(ACC, currLocalNum, FP, s);
    int e1_offset = currLocalNum;
    ++currLocalNum;
    e2->code(s);
    emit_jal("Object.copy", s);
    emit_load(T1, e1_offset, FP, s); // load previous result from stacks
    emit_fetch_int(T1, T1, s);       // $t1 = M[$s1 + 12]
    emit_add(T1, T1, T2, s);         // $t1 = $t1 + $t2
    emit_store_int(T1, ACC, s);      // M[$a0 + 12] = $t1 (store the value)
    --currLocalNum;
}

void sub_class::code(ostream &s)
{
    e1->code(s);
    // save result in stack
    emit_store(ACC, currLocalNum, FP, s);
    int e1_offset = currLocalNum;
    ++currLocalNum;
    e2->code(s);
    emit_jal("Object.copy", s);
    emit_load(T1, e1_offset, FP, s); // load previous result from stacks
    emit_fetch_int(T1, T1, s);       // $t1 = M[$s1 + 12]
    emit_fetch_int(T2, ACC, s);      // $t2 = M[$a0 + 12]
    emit_sub(T1, T1, T2, s);         // $t1 = $t1 - $t2
    emit_store_int(T1, ACC, s);      // M[$a0 + 12] = $t1 (store the value)
    --currLocalNum;
}

void mul_class::code(ostream &s)
{
    e1->code(s);
    // save result in stack
    emit_store(ACC, currLocalNum, FP, s);
    int e1_offset = currLocalNum;
```

```
    ++currLocalNum;
    e2->code(s);
    emit_jal("Object.copy", s);
    emit_load(T1, e1_offset, FP, s); // load previous result from stacks
    emit_fetch_int(T1, T1, s);       // $t1 = M[$s1 + 12]
    emit_fetch_int(T2, ACC, s);      // $t2 = M[$a0 + 12]
    emit_mul(T1, T1, T2, s);         // $t1 = $t1 * $t2
    emit_store_int(T1, ACC, s);      // M[$a0 + 12] = $t1 (store the value)
    --currLocalNum;
}

// notice don't need to handle divide by zero
void divide_class::code(ostream &s)
{
    e1->code(s);
    // save result in stack
    emit_store(ACC, currLocalNum, FP, s);
    int e1_offset = currLocalNum;
    ++currLocalNum;
    e2->code(s);
    emit_jal("Object.copy", s);
    emit_load(T1, e1_offset, FP, s); // load previous result from stacks
    emit_fetch_int(T1, T1, s);       // $t1 = M[$s1 + 12]
    emit_fetch_int(T2, ACC, s);      // $t2 = M[$a0 + 12]
    emit_div(T1, T1, T2, s);         // $t1 = $t1 / $t2
    emit_store_int(T1, ACC, s);      // M[$a0 + 12] = $t1 (store the value)
    --currLocalNum;
}
```

8.20 比较表达式的代码生成

```
void lt_class::code(ostream &s)
{
    e1->code(s);
    emit_store(ACC, currLocalNum, FP, s);
    int e1_offset = currLocalNum;
    ++currLocalNum;
    e2->code(s);
    emit_load(T1, e1_offset, FP, s); // $t1 = e1 Object
    emit_move(T2, ACC, s);           // $t2 = e2 Object
    emit_fetch_int(T1, T1, s);       // $t1 = e1.val
    emit_fetch_int(T2, T2, s);       // $t2 = e2.val

    int end_label = get_label();
    emit_load_bool(ACC, truebool, s);
    emit_blt(T1, T2, end_label, s);
    emit_load_bool(ACC, falsebool, s);
    emit_label_def(end_label, s);
    --currLocalNum;
}

void leq_class::code(ostream &s)
{
    e1->code(s);
    emit_store(ACC, currLocalNum, FP, s);
    int e1_offset = currLocalNum;
    ++currLocalNum;
    e2->code(s);
    emit_load(T1, e1_offset, FP, s); // $t1 = e1 Object
    emit_move(T2, ACC, s);           // $t2 = e2 Object
    emit_fetch_int(T1, T1, s);       // $t1 = e1.val
    emit_fetch_int(T2, T2, s);       // $t2 = e2.val

    int end_label = get_label();
    emit_load_bool(ACC, truebool, s);
    emit_bleq(T1, T2, end_label, s);
    emit_load_bool(ACC, falsebool, s);
    emit_label_def(end_label, s);
```

```
    --currLocalNum;
}
```

8.21 等于表达式的代码生成

```
void eq_class::code(ostream &s)
{
    e1->code(s);
    emit_store(ACC, currLocalNum, FP, s);
    int e1_offset = currLocalNum;
    ++currLocalNum;
    e2->code(s);
    emit_load(T1, e1_offset, FP, s);
    emit_move(T2, ACC, s);

    // compare the pointer first
    // if pointer the same, then must be the same
    int equal_label = get_label();
    emit_load_bool(ACC, truebool, s);
    emit_beq(T1, T2, equal_label, s);
    emit_load_bool(A1, falsebool, s);
    // else if pointer not the same, and the static type is the Int/Bool/String
    // then we can compare the val
    // here we dispatch function eqaulity_test. it is a builtin function
    // put true in $a0, false in $a1

    emit_jal("equality_test", s);
    emit_label_def(equal_label, s);

    --currLocalNum;
}
```

8.22 取反表达式的代码生成

```
void neg_class::code(ostream &s)
{
    e1->code(s);                // e1 object in $a0
    emit_jal("Object.copy", s); // copy a new Int object of e1 in $a0
    emit_fetch_int(T1, ACC, s); // fetch value attr of Int
    emit_neg(T1, T1, s);        // $t1 = -$t1
    emit_store_int(T1, ACC, s); // store back to $a0
}
```

8.23 取补表达式的代码生成

```
void comp_class::code(ostream &s)
{
    e1->code(s);
    emit_fetch_int(T1, ACC, s); // It is OK to use fetch_int to get the first attr
    int false_label = get_label();
    emit_load_bool(ACC, truebool, s);
    emit_beqz(T1, false_label, s); // if e1.val == false
    emit_load_bool(ACC, falsebool, s);
    emit_label_def(false_label, s);
}
```

8.24 new表达式的代码生成

```
void new__class::code(ostream &s)
{
    std::string object_name = type_name->get_string();
    if (type_name == SELF_TYPE)
    {
        // for SELF_TYPE, we have to find the appropriate class first
        // notice we need emit code to realize this

        // first, get the class tag of curr_class
        emit_load(T1, 0, SELF, s);
        // then get the class_objTab
        emit_load_address(T2, CLASSOBJTAB, s);

        // classtag * 8, this is the mapping from class tag to offset
        emit_sll(T1, T1, 3, s);
        // add the offset and the class_objTab addr, to get the real addr
        // now $t1 is addr of current class's protoObj
        emit_addu(T1, T1, T2, s);
        // get the content of this addr
        // $a0 = M[$T1]. i.e. the protoObj
        emit_load(ACC, 0, T1, s);
        // copy
        emit_jal("Object.copy", s);
        // now need to init this object
        // get the content of init_method

        emit_load(T1, 0, SELF, s);
        emit_load_address(T2, CLASSOBJTAB, s);
        emit_sll(T1, T1, 3, s);
        emit_addu(T1, T1, T2, s);
        emit_load(A1, 1, T1, s); // +4 offset for init_method
        emit_jalr(A1, s);
    }
    // we need to find the corresponding proto_obj, and copy it to $a0
    // how can we find it? we just need to load address...
    // then init the object. i.e. dispatch the init method
    else
    {
        auto protobj_addr = object_name + PROTOBJ_SUFFIX;
```

```
        emit_load_address(ACC, (char *)protobj_addr.c_str(), s);
        emit_jal("Object.copy", s);
        object_name = type_name->get_string();
        auto init_addr = object_name + CLASSINIT_SUFFIX;
        emit_jal((char *)init_addr.c_str(), s);
    }
}
```

8.25 isvoid表达式的代码生成

```
void isvoid_class::code(ostream &s)
{
    e1->code(s);
    emit_move(T1, ACC, s);

    // notice we directly compare the pointer
    // i.e. is the pointer 0?
    int eq_zero_label = get_label();
    emit_load_bool(ACC, truebool, s);
    emit_beqz(T1, eq_zero_label, s);
    emit_load_bool(ACC, falsebool, s);
    emit_label_def(eq_zero_label, s);
}
```

8.26 no_expr表达式的代码生成

```
void no_expr_class::code(ostream &s)
{
    emit_move(ACC, ZERO, s);
}
```

8.27 object表达式的代码生成

```
void object_class::code(ostream &s)
{
    if (name == self)
    {
        emit_move(ACC, SELF, s);
        return;
    }

    // search in objectEnvironment first
    if (objectEnvironment.lookup(name) != NULL)
    {
        // anyway we just need to add the offset and $fp to get the right val
        int offset = *(objectEnvironment.lookup(name));
        emit_load(ACC, offset, FP, s);
        return;
    }

    // then must be attribute
    // (no semantic error!)
    int offset = codegen_classtable->attrOffsetTable
        [codegen_classtable->curr_class->get_name()][name];
    emit_load(ACC, offset, SELF, s);
}
```

8.28 stack.cl的代码

```
class A2I {
     c2i(char : String) : Int {
        if char = "0" then 0 else
        if char = "1" then 1 else
        if char = "2" then 2 else
        if char = "3" then 3 else
        if char = "4" then 4 else
        if char = "5" then 5 else
        if char = "6" then 6 else
        if char = "7" then 7 else
        if char = "8" then 8 else
        if char = "9" then 9 else
        { abort(); 0; }
        fi fi fi fi fi fi fi fi fi fi
     };

     i2c(i : Int) : String {
    if i = 0 then "0" else
    if i = 1 then "1" else
    if i = 2 then "2" else
    if i = 3 then "3" else
    if i = 4 then "4" else
    if i = 5 then "5" else
    if i = 6 then "6" else
    if i = 7 then "7" else
    if i = 8 then "8" else
    if i = 9 then "9" else
    { abort(); ""; }
        fi fi fi fi fi fi fi fi fi fi
     };

     a2i(s : String) : Int {
        if s.length() = 0 then 0 else
    if s.substr(0,1) = "-" then ~a2i_aux(s.substr(1,s.length()-1)) else
        if s.substr(0,1) = "+" then a2i_aux(s.substr(1,s.length()-1)) else
           a2i_aux(s)
        fi fi fi
     };
```

```
     a2i_aux(s : String) : Int {
    (let int : Int <- 0 in
           {
               (let j : Int <- s.length() in
              (let i : Int <- 0 in
            while i < j loop
            {
                int <- int * 10 + c2i(s.substr(i,1));
                i <- i + 1;
            }
            pool
          )
           );
              int;
        }
        )
     };

    i2a(i : Int) : String {
    if i = 0 then "0" else
        if 0 < i then i2a_aux(i) else
          "-".concat(i2a_aux(i * ~1))
        fi fi
    };

    i2a_aux(i : Int) : String {
        if i = 0 then "" else
        (let next : Int <- i / 10 in
        i2a_aux(next).concat(i2c(i - next * 10))
        )
        fi
    };

};

class Main inherits IO
{
   sc : StackCommand <- new StackCommand;
   isStop : Bool <- false;
   main() : Object
   {
      while not isStop loop
```

```
      {
         out_string(">");
         let ch : String <- in_string() in
         (
            if ch = "e" then sc.evaluate()
            else if ch = "x" then isStop <- true
            else if ch = "d" then sc.display()
            else sc.push(ch)
            fi fi fi
         );
      } pool
   };
};

class StackCommand inherits A2I
{
   stack : String;
   io : IO <- new IO;

   push(char : String) : SELF_TYPE
   {
      {
         stack <- stack.concat(char);
         self;
      }
   };

   display(): SELF_TYPE
   {
      {
         let n : Int <- stack.length() - 1 in
         (
            while 0 <= n loop
            {
               io.out_string(stack.substr(n,1));
               io.out_string("\n");
               n <- n - 1;
            } pool
         );
         self;
      }
   };
```

```
    evaluate(): SELF_TYPE
    {
        {
            let n: Int <- stack.length() in
            (
                if n = 0 then {self;}
                else if stack.substr(n - 1, 1) = "+" then
                {
                    let x : Int <- a2i(stack.substr(n - 3, 1)),
                         y: Int <- a2i(stack.substr(n - 2, 1)) in
                    (
                        let ans : String <- i2a(x + y) in
                            stack <- stack.substr(0, n - 3).concat(ans)
                    );
                }
                else if stack.substr(n - 1, 1) = "s" then
                {
                    let x: String <- stack.substr(n - 3, 1),
                         y: String <- stack.substr(n - 2, 1) in
                    (
                        let ans : String <- y.concat(x) in
                            stack <- stack.substr(0, n - 3).concat(ans)
                    );
                }
                else {self;}
                fi fi fi
            );
            self;
        }
    };
};
```

8.29 stack.cl 编译结果和执行结果

部分编译代码：

```
2033        move    $a0 $s0
2034        bne $a0 $zero label149
2035        la  $a0 str_const0
2036        li  $t1 112
2037        jal _dispatch_abort
2038    label149:
2039        lw  $t1 8($a0)
2040        lw  $t1 32($t1)
2041        jalr        $t1
2042        bne $a0 $zero label150
2043        la  $a0 str_const0
2044        li  $t1 112
2045        jal _dispatch_abort
2046    label150:
2047        lw  $t1 8($a0)
2048        lw  $t1 16($t1)
2049        jalr        $t1
2050    label147:
2051        lw  $fp 24($sp)
2052        lw  $s0 20($sp)
2053        lw  $ra 16($sp)
2054        addiu   $sp $sp 28
2055        jr  $ra
2056
2057    # end of generated code
2058
```

SPIM执行结果：

```
breezer@node1:~/CompilerPA/assignments/PA5$ ../../bin/spim -file check_mine.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
>1
>2
>d
2
1
>+
>d
+
2
1
>e
>d
3
>5
>s
>d
s
5
3
>e
>d
3
5
>+
>e
>d
8
>
```

8.30 垃圾回收测试

需通过`./mycool stack.cl -o stack.s -g`的形式来生成代码。然后通过`../../bin/spim -file stack.s`来运行代码。

`-g`生成代码及运行结果：

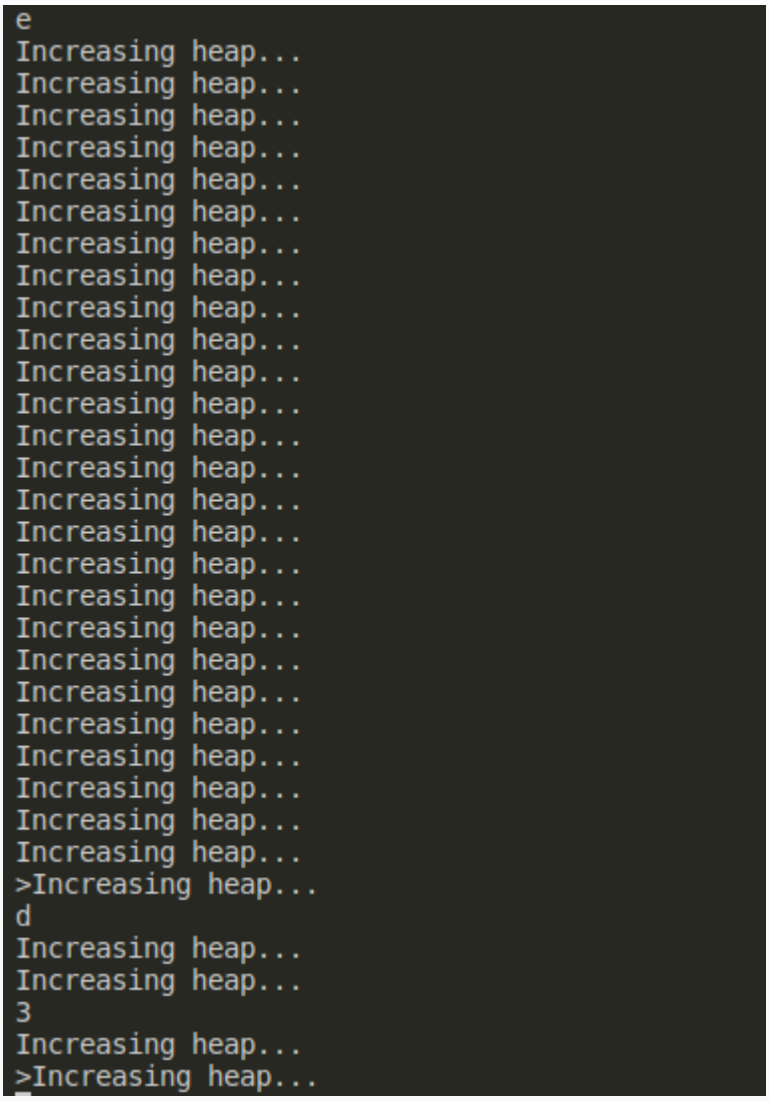
```
    .globl  Int_protObj
    .globl  String_protObj
    .globl  bool_const0
    .globl  bool_const1
    .globl  _int_tag
    .globl  _bool_tag
    .globl  _string_tag
_int_tag:
    .word   3
_bool_tag:
    .word   4
_string_tag:
    .word   5
    .globl  _MemMgr_INITIALIZER
_MemMgr_INITIALIZER:
    .word   GenGC_Init
    .globl  _MemMgr_COLLECTOR
_MemMgr_COLLECTOR:
    .word   GenGC_Collect
    .globl  _MemMgr_TEST
_MemMgr_TEST:
    .word   0
    .word   -1
str_const28:
```

```
breezer@node1:~/CompilerPA/assignments/PA5$ ../../bin/spim -file stack.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
GenGC initialized.
>1
>2
>+
>e
>d
3
>4
>s
>e
>d
3
4
>+
>e
>d
7
>
```

`-t`生成代码及运行结果：

```
    .globl  bool_const1
    .globl  _int_tag
    .globl  _bool_tag
    .globl  _string_tag
_int_tag:
    .word   3
_bool_tag:
    .word   4
_string_tag:
    .word   5
    .globl  _MemMgr_INITIALIZER
_MemMgr_INITIALIZER:
    .word   _NoGC_Init
    .globl  _MemMgr_COLLECTOR
_MemMgr_COLLECTOR:
    .word   _NoGC_Collect
    .globl  _MemMgr_TEST
_MemMgr_TEST:
    .word   1
    .word   -1
```

```
Increasing heap...
Increasing heap...
Increasing heap...
>Increasing heap...
1
Increasing heap...
>Increasing heap...
2
Increasing heap...
>Increasing heap...
d
Increasing heap...
Increasing heap...
2
Increasing heap...
Increasing heap...
1
Increasing heap...
>Increasing heap...
+
Increasing heap...
>Increasing heap...
```



8.31 全部由自己实现的Cool编译器测试

通过`../PA2/lexer stack.cl | ../PA3/parser | ../PA4/semant | ./cgen > allmycoolc.s`的形式，将MIPS汇编代码写入到`allmycoolc.s`，部分结果如图：

```
label11:
    move    $a0 $s0
    lw  $fp 32($sp)
    lw  $s0 28($sp)
    lw  $ra 24($sp)
    addiu   $sp $sp 32
    jr  $ra
Main.main:
    addiu   $sp $sp -20
    sw  $fp 20($sp)
    sw  $s0 16($sp)
    sw  $ra 12($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
label32:
    lw  $a0 16($s0)
    lw  $t1 12($a0)
```

通过`../../bin/spim -file allmycoolc.s`执行代码。结果如图：

```
breezer@node1:~/CompilerPA/assignments/PA5$ ../../bin/spim -file allmycoolc.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
>1
>2
>d
2
1
>+
>d
+
2
1
>e
>d
3
>4
>s
>d
s
4
3
>e
>d
3
4
>+
>e
>d
7
>
```