Lab04-MIPS Code Generation

> 实验名称：NJU-编译原理(2022秋)-Lab03-目标代码生成（Target Code/MIPS Code Generation）
> 实验人员：201220096 钟亚晨
> 完成日期：2023/01/09
> 更多实现细节可见个人博客文章：(pawx2's Blog)【密码：`chcp65001`】

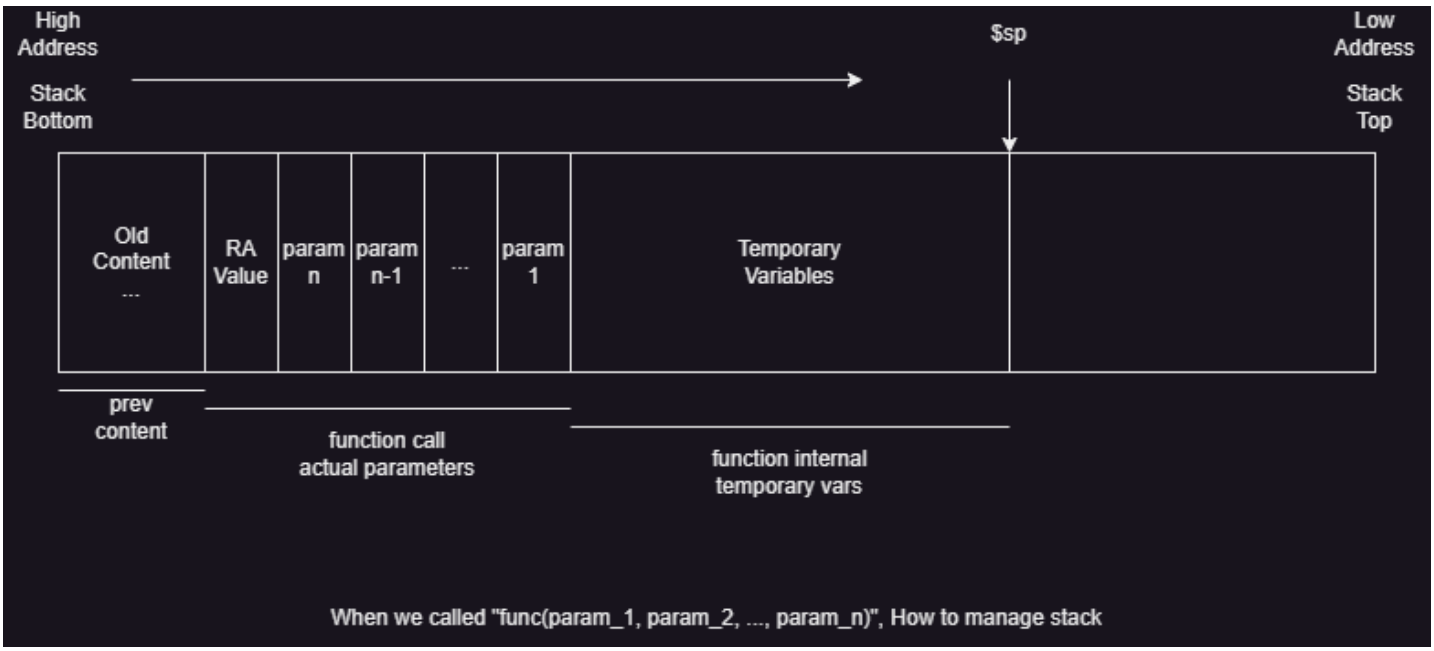
功能实现

- 实现了`实验四--目标代码生成.pdf`中要求的所有**基本内容**（无拓展内容要求）；
- **以90/100的评分通过了OJ平台上的测试用例，而联系助教通过了本地测试框架的所有测试用例**；（推测由于中间代码生成过多，导致目标代码条数或要求的栈空间太大而导致线上SPIM运行时出错）

实现思路

- 本次实验内容较为简单，主要分为三部分：

栈空间的安排



为了使得翻译出的MIPS代码尤其是在进行函数调用、函数嵌套调用时能够正确工作，我依照运行时的栈空间及MIPS的RA、SP寄存器工作流程来设计栈空间的管理，每当调用一个函数func时，进行如下流程：

调用前：①将当前RA寄存器内容压入栈中→②从尾到头向栈中压入实际参数→③依据函数需要的临时变量总size申请栈空间；
利用jal指令调用函数；
调用后：①归还申请的临时变量总size + 实际参数总size的栈空间→②从栈中弹出原RA内容，并存入RA寄存器；

寄存器的分配策略

本次采用最简单、稳定但不考虑效率的朴素寄存器分配方法，所有的临时变量保存至栈中，需要计算时从栈中依次取出，计算后保存到对应左值的栈空间。

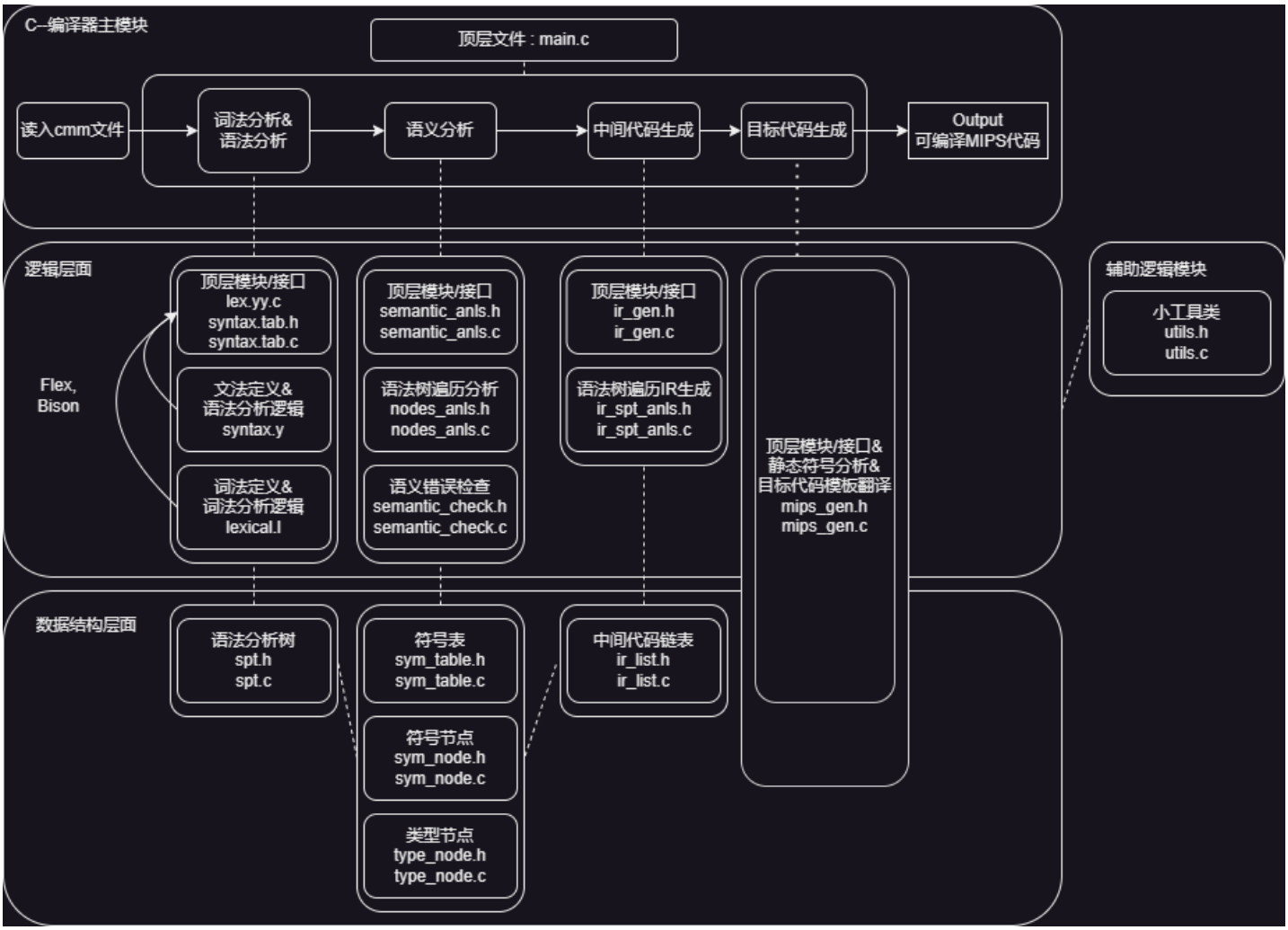
从中间代码到目标代码的翻译

配合寄存器的分配策略使用模板翻译策略，依据实验三中对中间代码的分配可以构造出对应的MIPS指令组合，详细见 `./Code/mips_gen/mips_gen.c`

实现亮点

程序架构图

本次实验在原有系统上追加了模块 `mips_gen`，架构图如下：



在实现过程中的一个重要问题在于：逐条执行MIPS语句中，如何知道其中**符号的位置**？而我的解决方案为：在进行中间代码→目标代码的模板翻译前，对程序进行静态分析获取各个函数符号偏移量，形成"一函数对应一偏移量表"，从而可以获取到一个函数的：①参数数量及size；②内部临时变量的总体size；③各个变量相对于 `$sp` 的偏移量；从而配合前述的栈空间管理，可以实现从中间代码→目标代码的精确、稳定的模板翻译。具体请见 `./Code/mips_gen/mips_gen.c: ir_stmt2mips_static_anls`

实验感想

- 在本次实验中，遇到了不少的问题，其中较为有意思的为：
    1. 在实现过程中提交OJ测试时，遇到了困扰自己很久的几个用例无法通过，最后联系助教老师进行本地测试，发现能够在本地测试中获得满分，推测是中间代码和目标代码的生成过于冗余，**指令条数和栈空间申请较多**，从而导致OJ上的SPIM在运行时出错。**之后通过对中间代码、目标代码生成过程的优化，削减了1/3的代码**（但代码总量和栈空间申请仍然较多），遂能通过OJ上部分原先无法通过的测试用例，但仍有极少的用例无法通过。
    2. 在自己Debug的过程中，发现并解决了函数调用时**无参数函数调用**导致的相关错误，而该部分的测试似乎在OJ上并未得到体现（具体Bug为：RA旧值的保存依靠对ARG语句的首次探测。未检测到参数压入，则在调用无参数函数时未保存RA的旧值），**希望以后能够将无参数函数的调用加入到OJ的测试当中。**
- 在解决上述问题1的过程中，我得到了如下深刻体会：
    - 最终生成的目标代码的效率，很大程度上取决于**中间代码的生成**，并且由于目标代码与机器相关，而硬件具备”固定“/”稳定“的特点，因此在对中间代码和目标代码的优化过程中，我的直观感受为：**中间代码层面的优化相比于目标代码的优化具备更多的空间和可能性。**
    - 也正是因为上述提及的原因，我也感受到编译器这一翻译系统中，**引入中间代码层将前端和后端分离**的重要性，一方面使得前后端的修改、更新得以分离，另一方面，又使得机器无关的代码优化（也是最能提升目标代码效率的手段）可以脱离机器指令抽象，优化统一的中间代码便能部署到各种机器达到优化效果。
    - 而**各种各样的前端→中间代码→各种各样的后端**这样一个呈现**沙漏**的结构，事实上并不仅仅在编译器系统的设计中有所体现，在计算机网络中的**七层协议模型里IP层**也有体现。因而计算机学科当中，具备着很多的共通之处，沙漏型结构和**沙漏颈**的意义就在于：将两组集合通过一个统一的**沙漏颈**实现抽象、转换和联系。
- 另外，至此C--编译器的整体系统可集成部分也都完成并通过了测试。在一个一个实验进行的过程中，一步一步地填充架构图上的各个模块，事实上多多少少会有小时候**搭积木**的感觉，而当这样一个代码量的项目从0开始而不依靠已有大框架在自己手中完成时，一方面是完成了一个更大体量项目的成就感，另一方面则是通过实现小型编译器，深刻体会编译器系统的各个设计部分和结构的**踏实感**，不会像以往计算机网络，各个协议层次最终都只觉得”纸上得来终觉浅“，学得很虚。
- NJU的编译原理课程与实验都设计得很好，尤其是不依赖任何大框架，从0开始”造“一个微型系统的经历，真的对于深刻体会编译器系统设计和结构有巨大的帮助。尽管工作量和学分不太匹配（吐个槽😛），**但确实是收获很大，毫无疑问是一门硬课、一门好课！**💪