优化文章

首先声明，虽然实现了两个版本的mips生成代码，但是最终提交的版本为优化前版本，优化后版本在函数调用的数组访问出现了一些无法解决的bug。

下面分享整体优化过程

寄存器分配

未优化的版本几乎只使用了`$t0, $t1`，因此实现了一个寄存器分配方法。

每个函数都有自己的一个符号表，用于对应新创建的mips符号和原本的llvm虚拟寄存器，由于要进行寄存器的分配，为每个函数的mips符号表额外对应一张寄存器分配表。寄存器表提供接口来分配临时，全局等寄存器

出现的主要问题在寄存器回收，如何确立良好的寄存器回收机制？

由于维护了所有符号和寄存器的对应关系和使用关系，我为符号设计了一个`isUsed`字段，用来说明是否被使用过，使用过的符号的寄存器是可以被回收的，这里的回收机制本质是在调用寄存器表的分配寄存器接口时，查找所有空闲的寄存器，包括`isUsed`被使用过的符号的寄存器，实现一个覆盖机制，从而实现寄存器的回收

然而寄存器的数量是有限的，在有些特殊的情况下，所有可分配寄存器都被临时变量占用，此时需要引入内存读写机制，在寄存器无法分配时，将一些指定的，比如我设计了一个变量维护最旧被使用的寄存器，将最旧使用的寄存器的值写入内存，更新新的最旧寄存器，释放写入内存的寄存器。寄存器分配和内存读写相配合来适应所有情况。

建立了寄存器写回机制，在函数调用前，跳转指令前，将所有的已分配的寄存器写回到内存中。

常量优化

对于中间代码中所有的常量，全部使用`li`指令分配常量值，而不使用内存访问

全局变量的分配优化

在全局变量分配时提前计算好其内存位置，使用指令时全部转换为一条`li`指令和`sw`指令，不使用过多的偏移量计算

srem模操作的优化

对于mod操作，在mips中执行非常慢，因此转换为`div`和`mfhi`指令加速计算

putstr函数的设计优化

在本人的llvm中，并没有设计字符串常量，也就是字符串的输出。所有的字符串输出都使用单个的`putch()`进行输出，因此输出的llvm代码非常的冗余，正常转换到mips效率也低(多次系统调用)，因此对于所有连续的字符串常量，将多个`putch`函数优化为`putstr`，将字符串常量加入到全局声明

alloca指令优化

alloca指令不实际分配内存，即不产生相关指令，只是创建符号并加入符号表，将所需内存分配换到store和load指令上执行，减少内存访问