编译原理实验四：目标代码生成

——杨嘉兴+21311054+2807593076@qq.com

一、 底层数据结构

本次实验的大部分代码老师均已给出，在目标代码生成的代码实现中，采用的底层数据是**寄存器描述符**、**变量描述符**和**栈帧描述符**。它们提供了一个基础的数据结构框架，用于在目标代码生成过程中**管理寄存器和变量的状态**，以便生成有效且正确的**目标代码**。它们帮助编译器跟踪寄存器和变量的分配情况，并提供必要的信息来生成与目标平台兼容的代码。

1. 寄存器描述符

该寄存器描述符结构体主要用于描述寄存器的**状态**和**属性**，可以标记寄存器是否可用，以便协调寄存器的分配，这样可以**避免**多个变量同时占用同一个寄存器；还可以**记录距离上次访问的间隔**，用于寄存器选择算法，帮助决定最适合用于存储某个变量的寄存器；以及**存储寄存器的别名**，方便在生成目标代码时标识寄存器；同时该结构体还可以**存储在该寄存器中的变量的描述符**，以便在代码生成过程中可以根据寄存器快速访问相应的变量信息。

2. 变量描述符

该变量描述符主要用于描述变量的属性和存储位置，可以存储变量相对于当前**栈帧底部的偏移量**，以便在生成目标代码时可以准确地定位变量在栈帧中的存储位置；还可以**描述变量的操作数信息**，例如**类型**、**名称**等，在生成目标代码时使用这些信息来正确处理变量的操作；同时该结构体还可以链接下一个变量描述符，以构建变量描述符的链表，方便**遍历**和**管理**栈帧中的所有变量。

3. 栈帧描述符

该栈帧描述符主要用于**描述函数的栈帧和其中的变量**，可以存储**该栈帧对应函数的名称**，方便在生成目标代码时可以快速查找函数的名称；还可以存储栈帧中预定存放对应函数的所有变量和临时变量的**变量描述符**；以及通过预先安排变量描述符的次序，可以方便地定位变量在栈帧中的存储位置；同时该结构体还可以链接下一个**栈帧描述符**，以构建栈帧描述符的链表，方便遍历和管理所有栈帧。

二、 功能实现

本次实验完成了所有的实验内容，成功地将实验三中得到的中间代码经过与具体体系结构相关的指令选择、**寄存器选择**以及**栈管理**之后，转换为 **MIPS32 汇编代码**。

1. 寄存器分配

- **确定可用寄存器集合：**首先，需要确定目标体系结构中可用的寄存器集合。**MIPS32** 架构中通常有一组通用寄存器（**如：$t0-$t9**）和一组特殊寄存器（**如：$sp、$fp、$ra 等**）。

- **建立寄存器分配策略：**根据编译器的设计和优化目标，选择适当的寄存器分配策略。常见的策略包括贪婪策略、图染色算法等。
- **分析变量的生命周期：**对中间代码进行分析，确定每个变量的生命周期，即变量在程序执行过程中的有效范围。
- **构建活跃变量分析：**基于变量的生命周期，构建活跃变量分析。该分析确定在每个程序点上哪些变量是活跃的，即在该点之后仍然被使用或写入的变量。
- **进行寄存器分配：**根据活跃变量分析和寄存器分配策略，为每个变量选择寄存器。通常，对于活跃的变量，优先选择可用寄存器而不是内存位置。
- **生成目标代码：**根据寄存器选择的结果，将中间代码转换为目标体系结构（**MIPS32**）的汇编代码。在生成目标代码时，将寄存器分配的结果反映在相应的指令中。

2. 栈管理

- **确定栈指针：**在 **MIPS32** 架构中，栈通常由栈指针（**$sp**）表示。栈指针指向栈顶的位置，即最新的数据存储位置。
- **分析函数调用：**对中间代码进行分析，识别函数的调用和返回。确定函数调用时需要保存的信息，如返回地址、调用者保存的寄存器等。
- **确定栈帧大小：**对于每个函数，需要确定其栈帧的大小。栈帧大小由函数的局部变量、参数和其他需要存储的数据的大小决定。
- **生成栈指令：**根据栈帧大小和函数调用的要求，生成相应的栈指令。常见的栈指令包括将栈指针移动到新的栈帧位置（扩展栈）、将数据压入栈、从栈中弹出数据等。
- **局部变量和参数的访问：**根据栈指针的位置和栈帧大小，计算局部变量和参数在栈中的偏移量。通过相对于栈指针的偏移量，可以在目标代码中访问局部变量和参数。
- **栈的动态管理：**在生成目标代码时，需要动态管理栈的使用。这包括栈的扩展和收缩，以适应函数调用和返回的需求。

三、 挖空部分的代码填充

1. 存在空闲寄存器时

首先，判断该寄存器是否为可用的空闲寄存器，如果当前寄存器是空闲的，将该寄存器与当前变量关联起来，然后更新寄存器的 **interval** 值。如果 **load** 的值为 **1**，表示需要将值加载到寄存器中，则根据变量的类型进行相应的装载操作。如果变量的操作类型是一个常量，将常量的值加载到寄存器中，使用 **MIPS** 汇编指令 **li**。如果变量的操作类型为 **VARIABLE_OP** 或 **TEMP_VAR_OP**，表示该变量存储在栈中，将栈中存储的变量值加载到寄存器中，使用 **MIPS** 汇编指令 **lw**。

2. 如果是获取地址操作时

首先为操作的 **opr** 字段获取一个可用的寄存器并返回该寄存器的编号，然后查找当前的

栈帧，随后基于操作的**opr**字段和当前栈帧创建一个变量描述对象，生成**MIPS**汇编指令**addi**，将栈帧指针**$fp** 加上变量的偏移量，结果存储到寄存器中。最后返回寄存器的编号即可。

3. 将 ASSIGN_IR 的中间代码翻译为目标代码并写入文件中

首先，获取目标操作数和源操作数，处理源操作数并返回源操作数所在的寄存器编号，然后根据目标操作数的类型进行不同的处理。如果目标操作数的类型是一个变量或临时变量，调用函数返回目标操作数所在的寄存器编号，生成 **MIPS** 汇编指令 **move**，将源寄存器的值移动到目标寄存器中，实现赋值操作。如果目标操作数的类型是一个获取值操作，调用函数返回目标操作数所在的寄存器编号。生成 **MIPS** 汇编指令 **sw**，将源寄存器的值存储到目标寄存器所指示的内存位置中，实现赋值操作。

4. 将 SUB_IR 的中间代码翻译为目标代码并写入文件中

首先，获取**左操作数**、**右操作数 1** 和**右操作数 2**，然后分别处理**右操作数 1** 和**右操作数 2**，返回它们所在的寄存器编号，根据左操作数的类型进行不同的处理。如果**左操作数**的类型是一个变量或临时变量，获取**左操作数**所在的寄存器编号，生成 **MIPS** 汇编指令 **sub**，将**右操作数 1** 寄存器减去**右操作数 2** 寄存器的值，并将结果存储到**左操作数**寄存器中，然后将使用完的**左操作数**所在的寄存器的值保存到栈中。而如果**左操作数**的类型是一个获取值操作，调用函数获取操作的第 1 个操作数所在的寄存器编号，生成 **MIPS** 汇编指令 **sub**，将**右操作数 1** 寄存器减去**右操作数 2** 寄存器的值，并将结果存储到操作的**第 1 个**操作数所在的寄存器中，获取操作的**第 1 个**操作数所在的寄存器编号，然后生成 **MIPS** 汇编指令 **sw**，将操作的**第 1 个**操作数所在的寄存器的值存储到**第 2 个**操作数所在的寄存器指向的内存位置中。

5. 将 DIV_IR 的中间代码翻译为目标代码并写入文件中

首先，获取**左操作数**、**右操作数 1** 和**右操作数 2**，然后分别处理**右操作数 1** 和**右操作数 2**，返回它们所在的寄存器编号，然后根据**左操作数**的类型进行不同的处理。如果**左操作数**的类型是一个变量或临时变量，获取**左操作数**所在的寄存器编号，生成 **MIPS** 汇编指令 **mflo**，将除法操作的结果存储在**左操作数**寄存器中，同时将使用完的**左操作数**所在的寄存器的值保存到栈中。如果**左操作数**的类型是一个获取值操作，获取操作的**第 1 个**操作数所在的寄存器编号，生成 **MIPS** 汇编指令 **mflo**，将除法操作的结果存储在操作的**第 1 个**操作数所在的寄存器中，然后获取操作的**第 2 个**操作数所在的寄存器编号。然后生成 **MIPS** 汇编指令 **sw**，将操作的**第 1 个**操作数所在的寄存器的值存储到**第 2 个**操作数所在的寄存器指向的内存位置中。

四、 代码编译与运行

1. **代码编译：**进入 **Code** 目录下执行 **make** 命令，会在该目录下产生相应的可执行文件 **parser**。
2. **代码测试：**编译得到 **parser** 文件后，通过**./parser ../Test/test1.cmm ../Result/out1.s** 对相应的测试文件进行测试，随后可在 **Result** 文件夹中得到该测试文件对应的目标代码生成。随后可以进入到 **Result** 目录中，然后通过命令 **spim -file out1.s** 来执行生成的汇编代码。