实验四

171860572 侯策 171860572@smail.nju.edu.cn

171860588 史文泰 171860588@smail.nju.edu.cn

中间代码分块

- 寄存器分配算法选择**局部寄存器分配算法**，因此，需要将中间代码分块。
- **函数块**：
  - **记录变量及其偏移**：对于每个函数块，记录其中出现的所有变量及其在该函数栈帧中的偏移量，以及这些变量的总偏移量（用来形成栈帧空间）。依次遍历中间代码，记录中间代码中出现的所有**v**和**t**的变量，这样的实现会产生两个效果：
    - 中间代码中使用到的参数`(v)`，实际变量`(v)`以及临时变量`(t)`都会在内存中对应空间。
    - 分配空间的顺序为变量在该函数中间代码中出现的先后顺序，而非在实际的`.cmm`文件中的定义顺序。
  - 针对每个变量，记录其在当前函数块中出现的位置的行号，为后续寄存器分配提供依据。
  - **函数块的划分**：
    - 函数块开始：当前句类型为`FUNCTION f :`，则当前句作为新的函数块开始。
    - 函数块结束：下一句类型为`FUNCTION f :`，或者当前句为最后一句中间代码，则当前句作为当前函数块结尾。
- **基本块**：
  - **基本块开始**：当前句为`LABEL`，`FUNCTION`，或上一句为`CALL`，`RETURN`，`GOTO`，`IFGOTO`，则当前句作为新的基本块开始。
  - **基本块结束**：当前句类型为：`CALL`，`RETURN`，`GOTO`，`IFGOTO`，或下一句为`LABEL`或`FUNCTION`，或者当前句为最后一句中间代码，则当前句为当前基本块结尾。

寄存器分配算法：

- 采用局部寄存器分配算法。
  - 数据结构：采用课上的数据结构：**寄存器描述符**以及**地址描述符**。数据结构的更新与上课PPT上讲解一致，每次做出变动之后，都需同时检查两数据结构相应位置是否需要更新。
  - 分配策略：采用实验手册PDF上讲解的方法：如果当前代码中有变量需要使用寄存器，就从当前空闲的寄存器中选一个分配出去；如果没有空闲的寄存器，不得不将某个寄存器中的内容写回内存（该操作称为溢出或`spilling`）时，则选择那个包含本**基本块内**将来用不到或最久以后才用到的变量的寄存器。
- 寄存器回收时机：
  - 时机一：在翻译每句中间代码时，会记录下来当前句所涉及到的全部寄存器，在翻译好当前句之后，会一一检查该句用到的寄存器，如果寄存器中的是立即数，则此时将该寄存器释放。如果该寄存器中所存储的所有变量在**本基本块中**均不会再次被使用，则将该寄存器溢出会内存，并将其释放。
  - 时机二：由于采用了局部寄存器分配算法，因此`$t0`~`$t9`和`s0`~`s7`没有本质的区别。这些寄存器是能够自由分配的寄存器。在当前基本块结束时，需要将`$t0`~`$t9`和`s0`~`s7`寄存器中被使用的变量溢出栈，并将相应寄存器释放。

指令选择

- 在指令选择时，遍历每个`FunctionBlock`，对于每个`FunctionBlock`，遍历其所有`BasicBlock`，以`BasicBlock`为单位，将`BasicBlock`内的每条中间代码翻译为对应的`MIPS32`代码即可。

栈管理

- 若函数**A**调用函数**B**，则**函数调用栈帧图**如下：

$ra of A	**A函数的返回地址**
$fp of A	**EBP旧值**
Parameters of B	**B函数的参数**
Local Variable of B	**B函数的局部变量**

- 由上表可知，本次实验**将`$30`寄存器用作`$fp`栈帧指针**，函数的参数和局部变量均由`$fp`索引得到。在函数中**将参数和局部变量同等对待**，**均存放在栈中**。由前述可知每个变量按其使用顺序存储（参数在前，局部变量在后），其`offset`域即可代表在栈中的偏移量。
- 在函数调用时，由于参数位于被调用者栈帧起始偏移位置，可以提前确定，故**无需使用`$a0`~`$a3`寄存器保存函数参数**，而是在`CALL`语句时向前遍历其参数，将对应参数提前压入栈中即可。
- 在函数参数压栈后，在预留位置中压入`$ra`，跳转执行。需要注意的是，不同于`x86`体系中压入`$eip`也即下一条指令作为返回地址，`$ra`**代表的是调用者的返回地址**，由硬件自动填写，我们仅需保存即可。进入被调用者后，将`$fp`压栈，保存`Old EBP`，调整新的`$fp`位置，而后移动`$sp`指向栈顶，即可开始后续代码。
- 函数返回时，将临时变量（包括参数）的栈回收，恢复`$fp`，并根据`$ra`直接跳转即可。
- 返回调用者后，需要从`$v0`寄存器中获得返回值，并**恢复`$ra`寄存器**。

编译方法

- 实验严格按照提供的`makefile`文件进行编译与执行。命令如下：

  ```
  make clean
  make parser
  make test
  ```

  若需要添加新的测试文件,则在`Test`文件夹中添加文件,在`makefile`中修改对应的指令即可。