HW9 Report

胥昊天

2025-05-28 17:39:35

一、实验目标

本实验的目标是：**根据给定的四元式（Quad）中间代码和寄存器分配（Color）结果，生成符合 RPI（ARM）平台的汇编代码**。输入为 prepared quad 和 color 文件，输出为 RPI 汇编文件，要求正确处理寄存器分配、溢出变量（spills）、指令合并与拆分等问题。

二、实现思路

1. 总体流程
   - 读取 prepared quad 文件，解析为 QuadProgram。
   - 读取 color 文件，解析为 ColorMap，获得每个函数的寄存器分配与溢出信息。
   - 对每个函数，调用 `convert(QuadFuncDecl* func, DataFlowInfo *dfi, Color *color, int indent)`，将四元式转换为 RPI 汇编代码。
   - 汇编代码输出到指定文件。
2. 关键数据结构
   - **Color**：记录每个临时变量分配到的物理寄存器编号，以及溢出变量及其在栈帧中的偏移。
   - **ColorMap**：管理所有函数的 Color 信息，便于多函数程序的寄存器分配查询。
   - **DataFlowInfo**：提供活跃变量分析结果，辅助指令合并优化。
3. 代码生成核心逻辑
   - 遍历每个四元式指令，根据类型生成对应的 ARM 汇编指令。
   - 临时变量（temp）用分配的物理寄存器（r0~r10）替换。
   - 对于溢出变量（spill），使用 r9/r10 作为中转寄存器，每次用到前都从栈中加载（`ldr`），作为目标时用后写回栈（`str`）。

- 支持部分指令合并优化，如连续的加法和 load/store 可合并为带偏移的 ldr/str 指令。
- 标签、跳转、条件跳转等均加上函数名前缀，保证全局唯一性。

三、关键实现细节

1. 溢出变量处理
   - 每次用到溢出变量时，调用 `loadSrcReg` 生成 `ldr r9/r10, [fp, #-offset]` 指令，保证 r9/r10 的值总是最新。
   - 每次溢出变量作为目标写入时，调用 `storeDstReg` 生成 `str r10, [fp, #-offset]` 指令。
   - 不假设 r9/r10 的值会一直有效，严格按照标准输出每次都重新加载。
2. 指令合并与拆分
   - 检查连续的加法和 load/store，若满足条件则合并为 ldr/str 带偏移指令，减少冗余指令。
   - 跳转指令（Jump/CJump）根据后继语句是否为目标标签，省略不必要的跳转。
3. 代码风格与可读性
   - 所有汇编指令统一缩进，便于对比和调试。
   - 生成的标签、全局符号等均规范化处理，避免命名冲突。

四、遇到的问题与解决方法

1. 溢出变量多次用到时的加载问题

   最初实现时，未能保证每次用到溢出变量都插入 `ldr` 指令，导致 r9/r10 的值被覆盖后未及时更新，和标准输出不一致。通过分析标准输出，调整生成逻辑，**每次用到溢出变量都插入一次加载指令**，保证行为一致。

2. 行数不一致与空行问题

   在自动化测试时，发现生成的汇编文件与标准输出行数不一致。通过 diff 工具定位到具体差异，发现有时多/少生成了一条指令或空行。通过严格对齐每条指令的生成时机，解决了该问题。

3. 指令合并优化的边界处理

   在合并加法和 load/store 指令时，需确保合并不会影响语义，特别是变量活跃性分析的正确使用。通过 DataFlowInfo 的 liveout 信息，判断变量是否只在下一条指令中使用，保证合并安全。