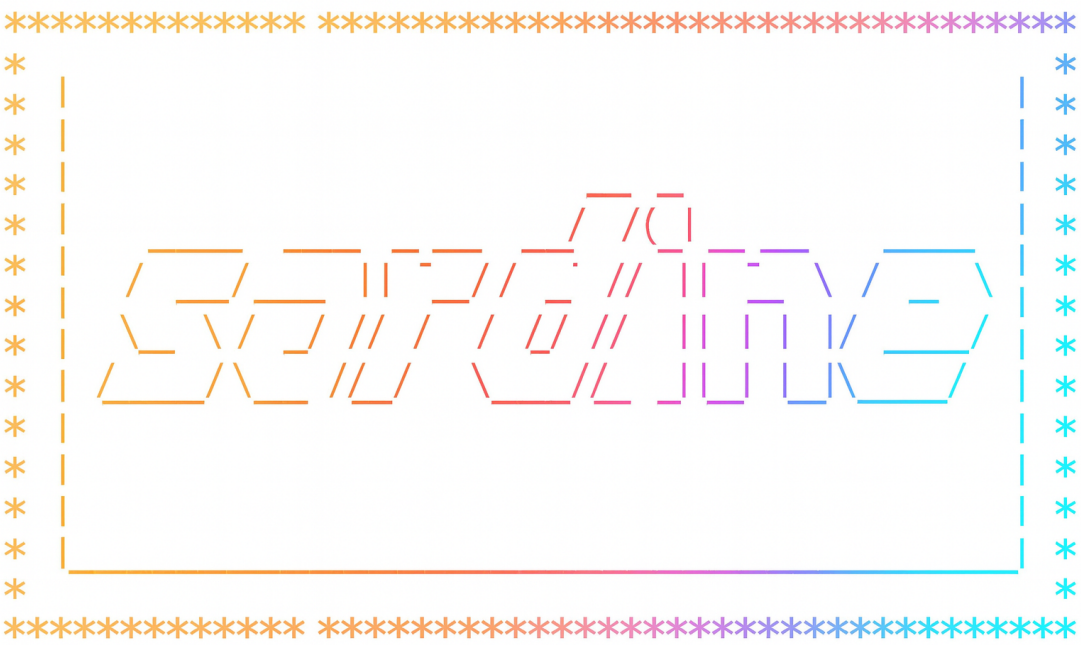
Sardine 编译器技术文档



1 项目概述

Sardine 编译器是为 2025 年编译系统设计赛而开发的编译器。编译器支持 SysY2022 语言，这是一种简化的 C 语言子集，语言特性包括基本的数据类型、控制流语句、函数和数组操作等。编译器将 SysY2022 源代码（.sy 文件）编译成 ARMv8-a 指令集的目标代码（.s 文件），并且能够优化生成的代码，以充分利用 ARM 架构的特点，提高程序的运行效率。

2 系统架构

项目使用 Java 语言开发，采用模块化设计。项目大体上分为前端和后端。前端负责词法分析、语法分析和语义分析，生成类 LLVM 的中间表示（IR）。后端通过遍历中

间代码 IR，最终生成 ARMv8-a 指令集的目标代码（.s 文件）。此外，在 IR 层面也将作出相应的优化 Pass，以提高生成的汇编代码的执行效率。

3 详细设计

Sardine 编译器项目的目标是构建一个高效、可扩展的编译器，以支持 SysY2022 语言并生成适用于 ARMv8-a 架构的目标代码。编译器的架构设计关键在于清晰地划分各个处理阶段，以确保源代码能够高效转换成目标代码。以下是编译器的主要架构组成部分及其详细设计：

3.1 前端（Frontend）

前端负责解析源代码，检查语法错误，并构建抽象语法树（AST）。这一阶段包括以下关键子系统：

- 词法分析器（Lexer）：将源代码字符串转换成一系列标记（tokens）。这些标记是构建语法树的基本元素，包括关键字、运算符、标识符等。词法分析器支持 SysY2022 语言的所有词法单元，包括：
  - 关键字：`const, int, float, break, continue, if, else, while, return, void`
  - 运算符：算术运算符、比较运算符、逻辑运算符
  - 标识符和字面量：整数、浮点数、字符串常量
  - 分隔符：括号、分号、逗号等
- 语法分析器（Parser）：解析标记序列，检查源程序的语法结构是否符合 SysY2022 的语法规则，并构建 AST。错误处理机制将在此阶段捕获并报告语法错误。语法分析器采用递归下降的方式，支持：
  - 变量声明和定义
  - 函数定义和调用
  - 控制流语句（if-else, while, break, continue）
  - 表达式和语句
  - 数组操作
- 抽象语法树（AST）：表示源代码的语法结构，为后续的语义分析和 IR 生成提供基础数据结构。

3.2 中间表示（IR）

AST 经过语义分析后，将转换成一种更接近机器语言的中间表示（IR）。IR 提供了一个与具体机器无关的代码表示方式，便于实现代码优化和目标代码生成。我们所采用的 IR 是类 LLVM 的、SSA 形式的 IR。

- IR 生成：将 AST 转换为我们的 IR，为接下来的优化阶段提供一个更加规范和易于操作的数据结构。IR 生成过程包括：
  - 符号表管理：维护变量和函数的作用域信息
  - 类型系统：支持整数、浮点数、指针等类型
  - 控制流图构建：将程序转换为基本块和边的形式
  - SSA 形式转换：消除变量重定义，提高优化效果
- IR 指令集：支持多种类型的指令，包括：
  - 算术运算指令：`add`，`sub`，`mul`，`div`，`mod` 等
  - 比较指令：`eq`，`ne`，`lt`，`le`，`gt`，`ge` 等
  - 内存操作指令：`load`，`store`，`alloca` 等
  - 控制流指令：`br`，`call`，`ret` 等
  - 类型转换指令：`zext`，`ftoi`，`itof` 等
- Pass 优化：在 IR 层面进行各种优化处理，如死代码消除、循环优化等，以提高代码效率和减少最终生成的代码量。优化 Pass 包括：
  - 常量折叠（Constant Folding）
  - 死代码消除（Dead Code Elimination）
  - 循环优化（Loop Optimization）
  - 寄存器分配优化

3.3 后端（Backend）

后端负责将优化后的 IR 转换成目标机器 ARMv8-a 的汇编代码。这一过程涉及到多个重要的子阶段：

- 指令选择：将 IR 指令转换为特定架构 ARMv8-a 的机器指令。支持：
  - 整数运算指令
  - 浮点运算指令

  - 内存访问指令
  - 分支和跳转指令
  - 函数调用约定

- 寄存器分配：分析程序的变量使用情况，将变量分配到寄存器中。由于寄存器数量有限，这一步骤还包括必要的溢出处理，即将一些变量存储在内存中。寄存器分配策略包括：
  - 朴素寄存器分配算法
  - 活跃变量分析
  - 寄存器溢出处理
  - 栈帧管理

- 代码输出：将最终的汇编代码输出到目标文件（.s 文件）。输出格式符合 GNU 汇编器语法，包括：
  - 数据段和代码段的组织
  - 全局变量和局部变量的处理
  - 函数调用约定的实现
  - 运行时库的链接

3.4 工具和环境支持

- 开发环境：
  - 使用 Java 作为主要开发语言，提供良好的面向对象特性和丰富的标准库
  - 采用模块化设计，便于维护和扩展
  - 支持多种编译选项，如调试模式、优化级别等

- 测试和调试：
  - 编译器开发中包含广泛的单元测试和集成测试，以确保每个构建阶段的正确性和性能
  - IR 有相应的类似与 LLVM 的文本打印的形式，便于及时检查调试。最终测试时，使用交叉编译器 + QEMU 模拟器，模拟指定硬件的运行，得到测试结果

- 构建系统：
  - 提供 PowerShell 构建脚本，支持 Windows 环境下的自动化编译

  - 支持清理、详细输出等构建选项
  - 自动检测 Java 环境并进行编译

4 未来计划

- 增加优化 Pass：根据性能分析结果，增加更多的优化 Pass，如循环向量化、函数内联等
- 性能优化：进一步优化编译速度和生成代码的执行效率
- 工具链完善：提供更完善的调试和性能分析工具