



PARALLEL COMPILER CONSTRUCTION

SysY 编译器课程实验

前端 · 中端优化 · 后端代码生成

从 SysY 语言到 **AArch64 汇编** 的完整编译器实现，涵盖**语法分析**、**IR 生成**、**标量优化**、**寄存器分配与循环优化**六大实验模块。

Lab1-Lab6 全部完成

21 项回归测试通过

全量测试 217.293s

实验验收汇报

小组成员：**程景愉 · 舒钰权 · 杨力嘉**

仓库地址：**<https://git.nudt.space/CompileThreeMaggot/nudt-compiler-cpp.git>**



项目概述与实验目标

完整的编译器流水线

实现从 **SysY 源程序** 到 **AArch64 汇编** 的完整编译流程，打通前端解析、中端优化与后端代码生成的全部环节。

六阶段渐进式实验

按 Lab1–Lab6 逐步完成语法树构建、IR 生成、指令选择、标量优化、寄存器分配与循环优化，形成层次清晰的编译器架构。

工程化协作实践

基于 Git 分支协作、CMake 构建系统、自动化测试脚本与 QEMU 模拟验证，完整复现工业级编译器开发流程。

Lab1
语法树

Lab2
IR 生成

Lab3
汇编生成

Lab4
标量优化

Lab5
寄存器分配

Lab6
循环优化



技术栈总览

编译器架构

- 前端** ANTLR4 + Visitor 模式，词法/语法分析 → 语法树
- 语义分析** 符号表 + 作用域栈 + 类型检查 + 名称绑定
- 中端 IR** LLVM 风格 SSA IR，含完整 use-def 链与 CFG
- 中端优化** Mem2Reg · ConstFold/Prop · CSE · Load CSE · DCE · CFGSimplify · LICM
- 后端 MIR** 机器级中间表示 → 虚拟寄存器 → 物理寄存器
- 后端优化** 窥孔优化 · 冗余消除 · 栈帧压缩 · SP 直接寻址 · 寄存器别名感知

工具链与验证

- 构建** CMake + C++17，parse-only / 全量两种模式
- IR 验证** LLVM 工具链 (llc + clang) 编译运行比对
- 汇编验证** AArch64 交叉编译 + QEMU 用户态模拟
- 目标平台** ARM64 / AArch64 (gcc-aarch64-linux-gnu)
- 运行库** 自研 sylib.c，含 I/O、计时与浮点十六进制支持
- 测试** verify_ir.sh / verify_asm.sh 自动化回归脚本



Lab1: 语法树构建 — 前端基石

核心工作

- 基于 ANTLR4 扩展 `SysY.g4` 文法，覆盖完整 SysY 语言规范
- 实现**控制流语句** (if-else / while / break / continue)
- 支持**表达式优先级**、浮点数字面量、数组类型与函数参数
- 通过 Visitor 模式遍历语法树，输出结构化语法树打印

关键产出

- 完整的 SysY 词法/语法规则定义
- C++ Lexer/Parser 自动生成流程
- `SyntaxTreePrinter` 语法树可视化
- parse-only 构建模式，快速迭代验证

技术要点

ANTLR4 的 labeled alternative 写法直接影响 `SysYParser::*Context` 类型名与访问接口，为后续 sem/irgen 的 Visitor 适配奠定基础。扩展文法后必须同步调整语义分析与 IR 生成的 `visit*` 逻辑。



Lab2: 中间表示生成 — IR 语义全覆盖

IR 类型系统与指令扩展

- 扩展 IR 类型系统: `i32` / `float` / 数组 / 指针
- 覆盖完整指令集: 算术、比较、**GEP 地址计算**、类型转换 (`zext` / `sitofp` / `fptosi`)
- 支持 **短路求值** (`&&` / `||`) 与控制流 (`if-else` / `while`)
- 函数定义、调用与参数传递的 IR 生成

关键难点突破

- **编译期常量求值**与运行时 IR 生成严格分离
- 数组对象、数组形参 (指针退化)、数组元素地址明确区分
- 多维数组花括号初始化的聚合布局
- IR 打印格式对齐 LLVM 规范 (SSA 命名、GEP、比较结果类型)

成果: 支持 **int/float 常量表达式**、**多维数组**、**函数调用**、**短路求值**、**控制流**, 生成的 IR 通过 `llc` 编译与 `clang` 链接运行验证。



Lab3: 指令选择与汇编生成 — AArch64 后端

AArch64 指令覆盖

完整实现**算术、比较、条件选择、分支跳转、函数调用、内存访存、浮点转换**等核心指令子集，采用高可靠栈槽模型保证变量活跃期正确性。

ABI 调用约定

完整实现前 8 个整型/指针参数及前 8 个浮点参数通过寄存器传递，返回值通过 `w0/x0/s0` 返回，支持多函数多基本块控制流。

浮点位精确

浮点常量以 `.word <bits>` 二进制字面量输出，保证**编译-汇编-运行全生命周期 100% 位一致**，消除精度丢失问题。

关键难点突破

解决**双向迭代器/指针失效**（vector 重配 → 野指针）、**大栈帧寻址越界**（ldur/stur 超出 [-256,255] 范围 → 寄存器偏移寻址回退）、**参数指针二级间接**等底层问题。重写 `sylib/sylib.c` 运行库，补齐全部 I/O 与十六进制浮点（`%a`）支持。



Lab4: 基本标量优化 — SSA 中端核心

支配树分析与 Mem2Reg

- 实现 **Cooper-Harvey-Kennedy 迭代支配树算法**
- 计算支配边界，在控制流汇合点插入 Phi 节点
- 沿支配树 DFS 完成**变量重命名**，构建 SSA 形式
- 消除 alloca/load/store 冗余内存访问

优化 Pass 管线

- **ConstFold**: 算术/比较/类型转换深度折叠与代数简化
- **ConstProp**: 常量传播 + 条件分支死目标剪枝
- **CSE**: 块内局部公共子表达式消除
- **DCE**: Mark-and-Sweep 死代码删除
- **CFGSimplify**: 合并线性块、清理不可达代码

Phi 节点降低到汇编

在栈槽后端正确处理 Phi 生命周期：控制流分叉块末尾生成**条件拷贝 (Condition Copy-Store)**，函数头部预分配 Phi 槽位。修复指针截断 (64→32 位)、GEP 参数二级指针解引用、ConstProp 后 Phi 残留清理等隐蔽缺陷。



Lab5: 寄存器分配与后端优化 — 窥孔优化

窥孔优化核心能力

- **同名寄存器自移动消除**: 识别并删除 `mov w8, w8` 等无意义指令
- **冗余 Load-after-Store 消除**: 栈槽写入后紧跟同寄存器读取 → 直接复用
- **寄存器尺寸匹配**: W/X 寄存器间 move 时动态适配尺寸

寄存器别名感知

- 实现 **NormalizeReg**: $X0-X28 \rightarrow W0-W28$ 归一化映射
- 所有别名冲突检测与消除基于归一化寄存器进行
- **隐式写寄存器追踪**: 浮点 MovImm 对 x8/w8 的副作用主动失效

技术挑战

浮点常数加载 (`adrp + ldr`) 隐式占用通用寄存器 x8/w8, 若窥孔器未感知会导致**寄存器污染**, 引发浮点比较错误。解决方案: 识别 MovImm 目标为浮点寄存器时, 主动擦除 slot_to_reg 追踪中的 x8/w8 条目。



Lab6: 并行与循环优化 — LICM

自然循环识别

- 基于**支配树**扫描 CFG 回边 (Back-edge)
- 回边 $B \rightarrow H$ 若 H 支配 B , H 为循环头 (Header)
- 沿前驱方向 BFS 收集循环体全部基本块
- 判定 Preheader (循环头在循环体外的唯一前驱)

循环不变式外提 (LICM)

- 不变性判定: 操作数为**常量、循环体外定义、或已判定为不变**
- 覆盖算术、比较、类型转换 (ZExt/SIToFP/FPTToSI)、GEP 地址计算
- 按数据流依赖**保序外提**到 Preheader 末尾 (Terminator 之前)

关键修复

支配边界计算死循环漏洞: DCE/CFGSimplify 后可能产生从 Entry 不可达的死块, 其 idom 为空或自环, 导致 ComputeDF 的 while 循环无限挂起。修复方案: 增加 `runner == nullptr` 与 `next_runner == runner` 的优雅阻断分支。



近期攻坚：运算符优先级修正与后端内存优化

运算符优先级与结合性修正

修正了 `SysY.g4` 中同级运算符分行写导致结合性错误的 ANTLR4 缺陷。将其合并为 `addSubExp` / `mulDivModExp` 等统一规则，并重构 `Sema` 与 `IRGen` 遍历逻辑，彻底解决了 `fft0.sy` 等复杂表达式的计算 Bug。

大数组零初始化 memset 优化

针对大局部数组初始化生成的几十万条冗余 store 指令进行了优化。在中端 IR 生成阶段，改用运行时 `memset` 函数调用，消除了汇编代码膨胀，使编译时间缩短 99% 以上。

后端死栈槽消除优化

在后端 `Peephole` 阶段新增了死栈槽分析。静态扫描发现从未被 load 或取地址的冗余 Store 槽位，直接予以删除，进一步缩减了栈帧空间并精简了指令流。

通过这几次的系统性优化与缺陷修复，编译器在**语义正确性、编译效率、以及生成代码的精简度**上均得到了极大的提升，实现了 21 个回归测试的 100% 完美通过。

性能优化专项：无硬编码的通用提速

IR 层：基本块内 Load CSE

在 `CSE` Pass 中增加同一基本块内的重复 `load` 消除：相同指针地址若未被 `store/call` 破坏，后续读取直接复用已有结果。该优化对 `A[i][j] * A[i][j]` 等循环密集模式收益明显，同时通过内存写入失效保证语义安全。

MIR 层：死栈槽删除 + 栈帧压缩

后端扫描所有 `FrameIndex` 使用，删除从未被读取或取地址的临时栈槽写入；随后重新紧凑布局仍然活跃的栈槽，缩小 frame size，减少无效栈空间和大偏移访存。

汇编层：SP 直接寻址

原先大偏移栈访问只能退化为 `ldr x10, =offset` + 寄存器偏移访存。优化后优先尝试 `[sp, #imm]` 正偏移寻址，大幅减少 literal load 和临时寄存器占用。

重点样例收益

`2025-MY0-20.sy` 单测运行时间由约 **130.8s** 降至约 **90.2s**；生成汇编中大偏移栈访问 literal load 由 **24** 降至 **0**。

栈访问优化收益

`if-combine3.sy` 中 `ldr x10, =offset` 由 **208** 降至 **0**，汇编行数由约 **923** 行降至约 **715** 行，单测约 **25s** 完成。

全量测试结果

取消所有 benchmark 特化和硬编码后，完整脚本 `./scripts/run_all_tests_verbose.sh` 从约 **279.6s** 降至 **217.293s**，21 项测试全部通过。



关键技术难点与突破

编译期/运行期分离

全局常量初始化走纯编译期求值，绝不生成 IR 指令。**避免依赖 Runtime IRBuilder 插入点。**

数组语义三层拆分

标量 alloca、聚合数组基址、数组形参指针退化**三种语义严格区分**，避免 Load/GEP 类型错乱。

浮点精度保全链路

常量折叠→IEEE 754 二进制位→`.word` 原样输出，确保**全链路位精确一致**。

SSA 一致性维护

ConstProp 后显式清理 Phi dead incoming 边；
CFGSimplify 正确替换 Phi uses；Mem2Reg 沿支配树 DFS 栈式管理版本。

后端指针安全

Vector 预分配容量避免迭代器失效；64 位指针强制 X 寄存器加载；参数 alloca 栈槽通过静态扫描提升至 8 字节。

支配树鲁棒性

ComputeDF 对不可达节点与自环做显式阻断；迭代 IDom 兼容非连通图与临时死块，保证收敛。



功能测试验证结果

序号	测试用例	测试内容	结果
1	simple_add.sy	简单加法，基础 IR/汇编验证	✓
2	05_arr_defn4.sy	多维数组定义与初始化	✓
3	09_func_defn.sy	函数定义与调用	✓
4	11_add2.sy	多变量算术表达式	✓
5	13_sub2.sy	减法与混合运算	✓
6	15_graph_coloring.sy	递归图着色（指针/数组）	✓
7	22_matrix_multiply.sy	矩阵乘法（多维数组+循环）	✓
8	25_scope3.sy	嵌套作用域与变量遮蔽	✓
9	29_break.sy	break/continue 控制流	✓
10	36_op_priority2.sy	运算符优先级综合测试	✓
11	95_float.sy	浮点 I/O / 类型转换 / 逻辑短路	✓



功能测试验证结果

序号	测试用例	测试内容	结果
12	01_mm2.sy	矩阵乘法（性能）	✓
13	02_mv3.sy	矩阵向量乘法（性能）	✓
14	03_sort1.sy	排序算法（性能）	✓
15	2025-MY0-20.sy	综合性能测试（循环/数组）	✓
16	fft0.sy	快速傅里叶变换（性能）	✓
17	gameoflife-oscillator.sy	生命游戏振荡器（性能）	✓
18	if-combine3.sy	条件组合与分支密集（性能）	✓
19	large_loop_array_2.sy	大循环数组访问（性能）	✓
20	transpose0.sy	矩阵转置（性能）	✓
21	vector_mul3.sy	向量乘法（性能）	✓



人员分工

组长 / 全栈核心

程景愉

- 负责各 Lab 中端 IR 与优化 Pass 实现
- 完成 Mem2Reg / ConstFold / CSE / DCE / LICM
- 支配树分析、循环识别与优化框架搭建
- Phi 节点降低到汇编的核心方案设计
- 作为组长统筹进度、文档与汇报材料

后端与系统

舒钰权

- 负责 Lab1 语法树构建与 ANTLR 文法扩展
- 负责 Lab3 后端指令选择与 AArch64 汇编生成
- 实现浮点位精确、大栈帧寻址回退机制
- 重写 sylib.c 运行库 (I/O / 计时 / %a 浮点)
- 修复指针截断、参数 GEP 越界等后端缺陷

优化与测试

杨力嘉

- 负责 Lab5 窥孔优化 (冗余 move / Load-after-Store)
- 实现寄存器别名感知 (W/X 归一化)
- 浮点隐式写寄存器追踪与失效处理
- 全量功能测试用例的回归验证
- 批量测试脚本维护与 CI 流程协调

分工遵循"**组长主抓中端优化核心 + 成员按前后端专长协作推进**"的模式, 通过 Git 分支 + PR 评审完成协作。



实验总结与展望

已完成的核心能力

- ☒ 完整 SysY 前端解析 (ANTLR4 + Visitor)
- ☒ LLVM 风格 SSA IR 生成与打印
- ☒ AArch64 后端指令选择与汇编输出
- ☒ Mem2Reg + 五大标量优化 + LICM
- ☒ Load CSE + 死栈槽删除 + 栈帧压缩
- ☒ SP 直接寻址与寄存器别名感知窥孔优化
- ☒ 21 项完整回归测试全部通过

可继续深入的方向

- ☐ 图着色 / 线性扫描寄存器分配
- ☐ 循环展开、强度削弱与并行化
- ☐ 过程间优化 (Inlining / IPO)
- ☐ GVN / PRE 等高级中端优化
- ☐ 更完整的 AArch64 指令调度

本项目已构建起一个**结构清晰、可扩展、语义正确**的 SysY 编译器框架，并在不引入测例硬编码的前提下，将完整回归测试耗时优化至 **217.293 秒**，为后续继续深入编译器优化与并行化研究提供了坚实基础。



CONCLUSION

谢谢聆听

从语法树到 AArch64 汇编，从 SSA 优化到循环不变式外提

我们构建了一个**完整、正确、可扩展**的 SysY 编译器

Q & A

程景愉 · 舒钰权 · 杨力嘉 | 并行编译优化课程实验