

《SysY 编译器课程实验》验收汇报配套讲稿

8 分钟精简版 | 约 1800 字逐字稿 + 答辩 FAQ

8 分钟汇报时间分配

- **页面时间**: 封面 10s / 概述 25s / 技术栈 15s / Lab1 20s / Lab2 45s / Lab3 30s / Lab4 45s / Lab5 25s / Lab6 25s / 近期攻坚 30s / 性能优化专项 45s / 难点 25s / 测试 20s / 分工 15s / 总结 20s / 致谢 5s。总计约 420 秒 ≈ 7 分钟演讲 + 1 分钟缓冲。
- **精讲原则**: 每页只讲 1-2 个核心技术点, 不展开细节。六个必讲亮点: 编译期/运行期分离、支配树+Mem2Reg、浮点位精确、寄存器别名、LICM、无硬编码性能优化。
- **语速**: 中文约 260 字/分钟, 本稿演讲正文约 2300 字。

逐页逐字稿 (8 分钟精简版)

第 1 页: 封面页 (10 秒)

【逐字演讲稿】各位老师、同学们, 下午好! 我是舒钰权。今天代表我们小组——程景愉、舒钰权、杨力嘉, 汇报 SysY 编译器课程实验成果。我们实现了从 SysY 到 AArch64 汇编的完整编译器, 六个实验全部完成, 21 项完整回归测试通过, 并将全量测试耗时优化到 217.293 秒。

【演讲技巧】站姿挺拔, 声音洪亮。一句话自我介绍 + 一句话项目概括。

第 2 页: 项目概述与实验目标 (25 秒)

【逐字演讲稿】项目定位: 从 SysY 源程序到 AArch64 汇编的完整编译器。六个实验呈递进关系——Lab1 语法树、Lab2 IR 生成、Lab3 汇编生成、Lab4 标量优化、Lab5 寄存器分配与窥孔优化、Lab6 循环优化。工程上实践了 Git 分支协作、CMake 构建、QEMU 模拟验证的完整流程。六个实验环环相扣, 语义正确性是我们的第一原则。

【演讲技巧】沿 flow-box 从左到右划过, 强调“递进关系”。

第 3 页: 技术栈总览 (15 秒)

【逐字演讲稿】快速一览技术栈。前端 ANTLR4 + Visitor, 中端自研 SSA IR 含完整 use-def 链, 中端优化实现了 Mem2Reg、五个标量 Pass、Load CSE 及 LICM, 后端 MIR 到 AArch64 汇编, 并加入栈帧压缩和 SP 直接寻址等后端优化。LLVM 工具链验证 IR, AArch64 交叉编译 + QEMU 验证汇编, 全程自动化。

【演讲技巧】快速全景扫描, 15 秒带过。

第 4 页: Lab1 语法树构建 — 前端基石 (20 秒)

【逐字演讲稿】Lab1 由我负责。核心是扩展 ANTLR4 文法, 覆盖完整 SysY——控制流、表达式、浮点、数组、函数参数。关键认知: ANTLR 文法的 rule 命名直接决定生成 C++ 类的类型名——意味着 sem 和 irgen 所有 visit* 函数必须与文法精确匹配。文法一变, 下游全部同步适配。这是前端工程“牵一发而动全身”的典型体现。

【演讲技巧】强调文法与下游耦合关系。快速带过, 不展开细节。

第 5 页: Lab2 中间表示生成 — IR 语义全覆盖 (45 秒)

【逐字演讲稿】Lab2 工作量最大, 由程景愉负责。扩展了 IR 类型系统和指令集, 实现了短路求值、控制流、函数调用与多维数组的 IR 翻译。

讲两个最核心的难点。第一, “编译期/运行期路径混用”——原 EvalConstExpr 内部调用了需要插入点的 IRBuilder, 全局初始化时直接崩溃。解决方案: 彻底分离, 常量路径只返回 ConstantInt/ConstantFloat, 绝不碰 IRBuilder。这是编译器设计的基本原则, 但初学者极易违反。

第二, “数组语义混乱”——标量 alloca、聚合数组基址、数组形参指针退化被混为一谈, 导致 Load/GEP 类型错误。我们做了三层严格拆分, 这是 IR 生成中最重要的设计决策。

【演讲技巧】放慢语速讲“编译期/运行期分离”, 全篇最核心的设计原则。

第 6 页: Lab3 指令选择与汇编生成 — AArch64 后端 (30 秒)

【逐字演讲稿】Lab3 由我负责。采用高可靠栈槽模型——每个 IR Value 分配专属栈槽，100% 保证变量活跃期正确性。攻克四个底层难题：vector 扩容指针失效——预分配容量；栈帧超 256 字节 ldur/stur 立即数越界——自适应回退寄存器寻址；浮点精度丢失——memcpy 取 IEEE 754 位、.word 原样输出，全链路位精确；重写 sylib.c 补齐 I/O。

【演讲技巧】“浮点位精确”和“大栈帧自适应寻址”最体现工程严谨性。

第 7 页: Lab4 基本标量优化 — SSA 中端核心 (45 秒)

【逐字演讲稿】Lab4 理论深度最高，由程景愉负责。先实现迭代支配树算法、计算支配边界，然后完成 Mem2Reg——汇合点插 Phi、沿支配树 DFS 变量重命名，将内存形式 IR 提升为 SSA。

在此基础上实现五个优化 Pass。ConstFold 做编译期计算与代数简化。ConstProp 传播常量并简化条件分支——这里有个极易遗漏的细节：简化后必须显式清理 Phi 的 dead incoming 边，否则后续 Pass 会基于脏数据做错误替换。CSE 做块内公共子表达式消除，DCE 用 Mark-and-Sweep，CFGSimplify 合并线性块。

Phi 降低到汇编的方案：控制流分叉块末尾生成条件拷贝，函数头部预分配槽位。同时修复了 64 位指针截断、GEP 二级指针解引用等缺陷。

【演讲技巧】支配树和 Phi 清理是两大亮点。讲 Phi 清理时加重语气。

第 8 页: Lab5 寄存器分配与后端优化 — 窥孔优化 (25 秒)

【逐字演讲稿】Lab5 由杨力嘉负责，聚焦后端窥孔优化。三类优化：消除同名寄存器自移动、冗余 Load-after-Store、寄存器尺寸动态适配。核心挑战是 AArch64 寄存器别名——Wn 和 Xn 共享物理寄存器，简单字符串比对会漏优化甚至做错。我们实现 NormalizeReg 归一化，X0-X28 映射到 W0-W28 再做冲突检测。另一个隐蔽问题：浮点 MovImm 底层翻译 adrp+ldr 时隐式占用 x8/w8，窥孔器必须感知并主动失效追踪。

【演讲技巧】强调 W/X 别名是后端开发者必知的核心知识点。

第 9 页: Lab6 并行与循环优化 — LICM (25 秒)

【逐字演讲稿】Lab6 由程景愉负责，实现循环不变式外提。三步：基于支配树识别回边——B→H 且 H 支配 B，H 为循环头，BFS 收集循环体；检查 Preheader 唯一性确保安全；worklist 迭代判定不变指令，覆盖 GEP 和类型转换，按拓扑序保序外提。

修复了一个隐蔽的死循环漏洞：DCE 后可能留下 idom 为空或自环的不可达死块，ComputeDF 的 while 循环永不收敛、编译器卡死。定位两小时，修复只需两行阻断代码。

【演讲技巧】死循环漏洞是精彩的调试故事。

第 9.5 页: 近期攻坚: 运算符优先级修正与后端内存优化 (35 秒)

【逐字演讲稿】接着汇报我们在回归测试与回归攻坚阶段完成的几项关键优化与修复。

第一，我们修正了前端 SysY.g4 文法中的运算符优先级和左结合性缺陷。原本文法把加减、乘除等同级运算符写在不同行，在 ANTLR4 中这会导致右结合或优先级错乱，使得 fft0.sy 等用例计算出脏数据。我们将其重构成合并为 addSubExp 等统一规则，并重写了相应的 Sema 和 IRegen AST 遍历逻辑，解决了这个隐蔽的语法解析 Bug。

第二，针对局部零初始化大数组，中端从直接生成几十万条 store 指令重构为生成运行时 memset 调用，彻底消除了代码膨胀与编译超时。

第三，后端在 Peephole 阶段新增了死栈槽优化，自动识别并删除了那些从未被 load 或取地址的冗余 store，进一步压缩了物理栈空间。

【演讲技巧】语速放慢，条理清晰地讲出“文法结合性”、“memset 大数组”和“死栈槽消除”三个近期攻坚点。

第 10 页: 性能优化专项: 无硬编码的通用提速 (45 秒)

【逐字演讲稿】这里重点汇报基础六个 Lab 之外，我们最后针对性能测试做的通用优化。要求是不能硬编码测试名、文件名或输出常量，所以我们只保留可以解释为编译器正常优化的方案。

第一是 IR 层 Load CSE：同一基本块内，如果两次 load 来自同一个指针，并且中间没有 store 或 call 破坏内存，就直接复用第一次 load 的结果。这个优化对 $A[i][j] * A[i][j]$ 这类循环密集表达式非常有效。

第二是 MIR 层死栈槽删除和栈帧压缩。删除从未被读取的临时栈槽后，重新紧凑布局活跃 frame slot，减少大负偏移访存。

第三是汇编层 SP 直接寻址。原先大偏移访问会生成 `ldr x10, =offset` 再访存；优化后能用 `[sp, #imm]` 就直接编码。效果上，2025-MY0-20.sy 单测从约 130.8 秒降到约 90.2 秒，if-combine3.sy 的大偏移 literal load 从 208 次降为 0。完整脚本从约 279.6 秒降到 217.293 秒，21 项测试全部通过。

【演讲技巧】 这一页是性能亮点，强调“无硬编码”和“可解释为通用优化”。数字要讲清楚。

第 11 页：关键技术难点与突破（25 秒）

【逐字演讲稿】 六大技术挑战总结。编译期/运行期分离——常量求值绝不碰 IRBuilder。数组语义三层拆分——标量、聚合、指针退化严格区分。浮点精度保全——从常量折叠到 .word 汇编全链路位精确。SSA 一致性——每个改变 CFG 的 Pass 必须同步维护 Phi 边。后端指针安全——预分配容量、64 位强制 X 寄存器、栈槽静态扫描。支配树鲁棒性——不可达节点和自环必须优雅阻断。这六点是优化开启后仍保持语义正确的基石。

【演讲技巧】 快速过六个要点，手指逐一指向卡片。

第 12 页：测试验证结果（20 秒）

【逐字演讲稿】 全部 11 项功能测试与 10 项性能测试在优化全开条件下通过，21 个用例输出与退出码 100% 匹配。当前无硬编码优化版本完整脚本耗时 217.293 秒。覆盖从 simple_add 到递归图着色、95_float 浮点综合测试，再到 2025-MY0-20 等性能测试。特别强调：这是在 Mem2Reg、五个 Pass、LICM、Load CSE 和后端栈优化全部开启下通过的——优化管线在提升性能的同时保证了语义正确。验证链路：SysY 源码 → IR → 优化 → AArch64 汇编 → QEMU 模拟 → 输出比对。

【演讲技巧】 强调“优化全开”、“21/21”和“217.293 秒”。

第 13 页：人员分工（15 秒）

【逐字演讲稿】 三人分工。程景愉负责中端优化——Lab2 IR 生成、Lab4 支配树与全部 Pass、Lab6 LICM。我负责 Lab1 文法扩展和 Lab3 AArch64 后端，攻克了浮点位精确等底层难题。杨力嘉负责 Lab5 窥孔优化与全量测试回归，在寄存器别名感知方面做出关键贡献。通过 Git 分支 + MR + Code Review 完成协作。

【演讲技巧】 真诚肯定组员贡献。

第 14 页：实验总结与展望（20 秒）

【逐字演讲稿】 核心成果：构建了一个结构清晰、语义正确、可扩展的 SysY 编译器框架。六个实验覆盖前端到后端全环节，在支配树、SSA 构建、Phi 降低、浮点位精确、寄存器别名、LICM 等关键技术上了做了深入实现；同时额外完成了 Load CSE、栈帧压缩和 SP 直接寻址等通用性能优化，把完整回归测试稳定压到 217.293 秒。可继续方向：寄存器分配升级为图着色/线性扫描，循环优化扩展到强度削弱和展开，中端引入 GVN/PRE。

【演讲技巧】 直视评委，展示热情和清晰规划。

第 15 页：致谢与 Q&A

【逐字演讲稿】 感谢各位老师和同学的聆听！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。接下来是答辩与提问环节，敬请批评指正！谢谢！

防答辩提问防线策略 (Q&A 环节 FAQ)

一、评委老师专业提问

问题一：Mem2Reg 中支配边界的作用是什么？如何计算？

应答：支配边界确定 Phi 插入位置——块 A 中定义的变量，在 A 的支配边界中的每个块都需要 Phi 汇合不同前驱的版本。我们使用经典迭代算法：对每个块，沿其前驱的支配链向上攀登直到遇到当前块的 idom，路径上所有块加入其支配边界。

问题二：LICM 如何处理指令间的传递依赖？

应答：采用 worklist 迭代判定。每轮遍历循环体，操作数全部满足不变条件则标记。传递依赖通过多轮自然解出——依赖循环外定义的先被标记，依赖它的再下一轮标记。外提按拓扑序移动，保证操作数可用。

问题三：栈槽模型冗余访存，为什么不做真正寄存器分配？

应答：窥孔优化约消除 30-40% 冗余。但要达到高质量代码，必须依赖完整的寄存器分配器——这是展望中列出的首要后续方向。当前策略是先保证语义正确、打通全链路，再替换为更优方案。

问题四：IR 的 use-def 如何维护？指令移动时如何保证一致性？

应答：Value 维护 Users 列表，Use 含双向指针。替换用 `replaceAllUsesWith` 遍历更新；删除时 `dropAllReferences` 清理操作数；跨块移动通过 `ilist splice` 不改变 use-def 关系。每个 Pass 后验证 Use 双向指针一致。

问题五：ConstProp 简化分支后为什么必须清理 Phi？

应答：ConstProp 将 `br il 0` 简化为无条件 `br` 后，被跳过块的 Phi 仍保留已删除前驱的 incoming 引用。CFGSimplify 合并时可能将残留值误作唯一值替换，导致语义错误。修复：简化时遍历死前驱后继块，显式调用 `removeIncomingBlock`。

二、现场同学互动提问**问题六：编译器能编译什么？有什么限制？**

应答：支持标准 SysY 完整语法——整数/浮点运算、控制流、多维数组、递归函数、I/O。不支持指针运算、结构体、动态内存分配。未实现循环展开和自动向量化。能正确编译 SysY 范围内所有程序，不能编译 C 程序。

问题七：为什么自己设计 IR 而不是用 LLVM API？

应答：课程教学需要亲手实现才能深入理解 SSA 本质；轻量自研 IR 可自由添加定制化分析和优化，不受第三方 API 约束。

问题八：遇到的最难 bug？

应答：Lab6 支配树死循环——编译器在 `95_float` 上完全卡死，GDB attach 发现 ComputeDF 中不可达死块 idom 自环导致永不收敛。定位两三个小时，修复只需两行阻断代码。教训：静态分析必须对 CFG “脏数据” 做防御性处理。

问题九：架构上会做什么不同的选择？

应答：一是在 Lab1/2 之间引入独立 AST 层解耦文法与下游；二是在 Lab3 就用虚拟寄存器，避免后续从栈槽模型重构。

问题十：性能怎么样？做了哪些非基础 Lab 优化？

应答：教学编译器仍然以语义正确优先，但我们额外做了三类通用性能优化：IR 层基本块内 Load CSE，MIR 层死栈槽删除与栈帧压缩，汇编层 SP 直接寻址。取消所有测例硬编码后，完整脚本从约 279.6 秒降到 217.293 秒；其中 2025-MYO-20 单测从约 130.8 秒降到约 90.2 秒，if-combine3 的大偏移 literal load 从 208 次降到 0。

问题十一：能在真机上运行吗？

应答：可以。汇编遵循标准 AArch64 指令集和 Linux ABI，ELF 是标准 ARM64 格式。可在树莓派、Apple Silicon Linux VM 等真机执行，无 QEMU 特有依赖。