mem2reg优化指南

mem2reg 的目标：把满足条件的 `alloca`（只经由 load/store 使用的标量局部变量）提升到 SSA 形式，
减少大量冗余的 load/store，通常收益非常明显。

1.1 策略

实现时按经典两阶段：

1. **插入 φ（phi）**：
   - 构建 CFG
   - 计算 dominator / idom / dominator tree
   - 计算 dominance frontier
   - 对每个 promotable alloca，在 DF 上插入 phi
2. **重命名**：
   - 在支配树上 DFS
   - 对每个 alloca 维护一个栈，模拟 reaching definition
   - 用栈顶值替换 load 的结果
   - store 则 push 新定义
   - 填充后继块 phi 的 incoming

1.2 困难点 1：promotable alloca 的判定

现象

如果把数组/指针相关的 alloca 也提升，会破坏内存语义（例如 GEP、按地址传参）。

解决方案

promotable 判定采用"保守正确"为原则：

- alloca 指向的类型必须是标量（例如 i32）
- alloca 的 use 必须只来自：
  - `load`（且 load 的指针操作数就是这个 alloca）
  - `store`（且 store 的指针操作数就是这个 alloca）

数组、指针算术、传址等一律不提升。

1.3 困难点 2：项目原本没有 phi 指令实现

现象

IR 枚举里可能预留了 `PHI` 类型，但没有对应类与打印逻辑。

解决方案

- 新增 `PhiInstr`：维护 (value, fromBlock) 的 incoming 列表
- 补齐打印格式：
  - `%x = phi i32 [ v1, %bb1 ], [ v2, %bb2 ]`

这样可以：

- `llvm_ir_after.txt` 中直接看到 mem2reg 效果（phi 出现、load/store 减少）
- 也为后端 lowering 提供结构化信息

1.4 困难点 3：后端不支持 phi，必须进行 lowering

现象

MIPS 没有 phi 指令，若直接忽略，会导致运行时值选择错误。

解决方案（工程上可行且实现量适中）

采用"**边上拷贝（edge copy）**"的方式：

- 对于块 `B` 开头的 phi：
  - `x = phi [v1, P1], [v2, P2] ...`
- 在每条前驱边 `Pi -> B` 上插入：
  - `x <- vi` 的拷贝（在 MIPS 中表现为把 vi 写回到 x 的栈槽）

为了能把"边上执行"落到线性 MIPS 文本，我们引入了 edge label：

- `Pi` 的 BR/JUMP 不直接跳 `B`，而是跳到一个新标签 `Pi_to_B_phi_edge_k`
- 在该 label 下输出 phi copies，再 `j B`

这种方案等价于"关键边分割 + 并行拷贝"的一种实现方式，
能保证语义正确，同时不必在 IR 层先完整实现 phi 消除 pass。

1.5 困难点 4：IR 中存在"终结指令后仍有指令"

这是本次优化里最隐蔽、也最影响正确性的坑。

问题表现（真实发生过）

在 testcase6 中，评测机报错：

> We got "13" when we expected "15" at line 8.

同一份源码：

- **关闭优化**（`--no-opt`）跑出来是对的：第 8 行是 15
- **开启 mem2reg** 后跑出来变成 13

这类"优化后运行结果变了"的问题，优先怀疑：

- CFG/支配关系算错
- phi incoming 填错
- 存在未定义值（use-before-def）

根因定位

IR 构建阶段对 `break/continue` 的 lowering，可能会产生这种块：

- 先插入一个 `jump next`
- 后面仍残留 `jump somethingElse`

也就是说：**同一个基本块里，出现了"terminator（BR/JUMP/RET）之后还有指令"。**

而 mem2reg 在构建 CFG 时，如果简单用 `bb->instructions.back()` 当 terminator，就会把 CFG 连错（把“最后一个冗余 jump”当成真正终结），导致：

- 可达块集合不完整
- dominator/DF 错误
- 一部分变量被提升到 SSA，一部分仍停留在内存形式
- 最终在某些循环/短路路径上读到了错误的 `cnt` 值

这正好解释了“只有优化开启才错”的现象。

解决方案

在 mem2reg pass 开始前，先做一次 IR 清理：

- 对每个基本块：
  - 找到第一条 terminator（BR/JUMP/RET）
  - **删除其后的所有指令**（它们在语义上不可达）

这样 CFG 就与真实控制流一致，支配关系计算也稳定。

这类修复属于“优化前置 IR 规范化”，收益很大：

- 不只 mem2reg，后续任何 CFG/数据流分析 pass 都会更可靠