

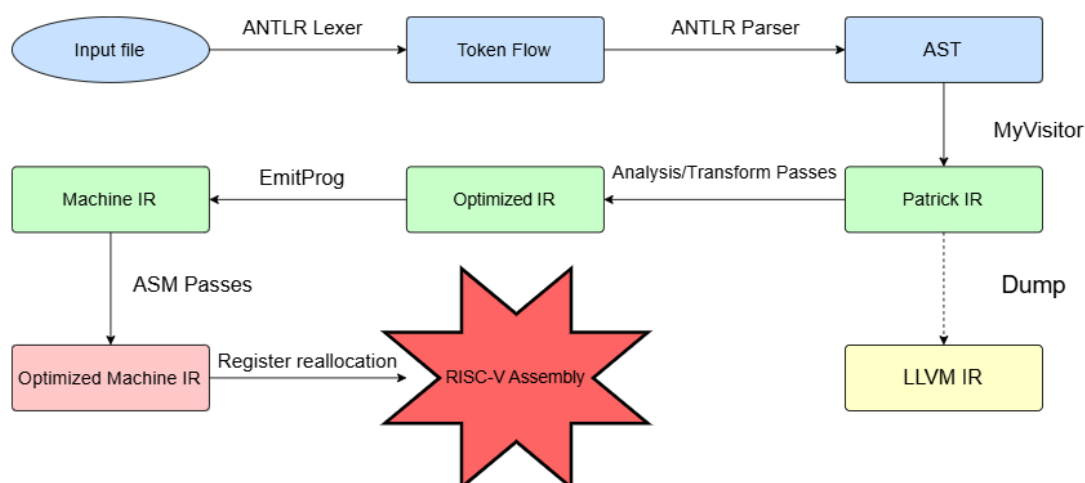
Patrick-CC 编译系统设计文档

引言

Patrick-CC 是一个支持 SysY2022 高级程序设计语言，面向 RISC-V 硬件平台，支持中端和后端编译优化的综合编译系统。该编译系统由前端词法语法分析器，中端 IR 生成器，后端目标代码生成器三个主要模块组成。

编译过程

编译过程示意图



编译系统设计方案

1. 编译器前端

1. 词法分析和语法分析

Patrick-CC 的词法分析器和语法分析器基于 ANTLR 4.13.1 构建。词法分析器负责对输入文件进行词法分析，并输出 token 流。随后，语法分析器读取这些 token 流，进行语法分析，从而构建抽象语法树（AST）。

2. IR 构建

myVisitor 是一个继承自 ANTLR 生成的基础访问器类的访问者类，可以遍历语法分析器输出的 AST，并执行自定义逻辑。

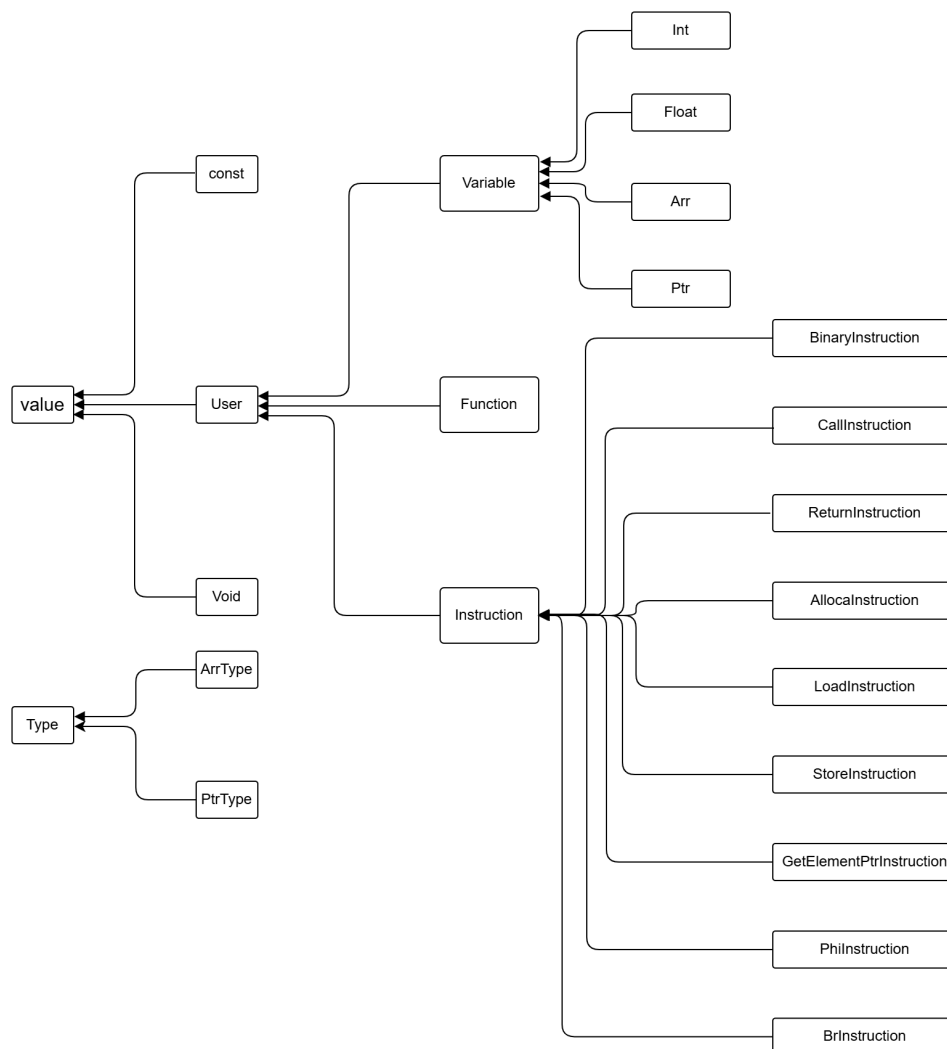
IRbuilder 是一个高度集成的 IR 生成器，提供了安全高效的 IR 操作接口，用于方便地构建和插入指令到一个基本块中。

在 IR 构建的过程中，myVisitor 遍历 AST，使用 IRbuilder 生成 Patrick IR

2. 编译器中端

1. Patrick IR

继承关系图



概述

Patrick IR 是一个与源语言，目标平台无关的中间表示中间代码形式，它位于源代码和目标代码（如机器代码）之间，旨在提供一种中立的、易于操作的表示形式，使得编译器可以进行各种优化和转换，而不依赖于源代码的具体语言或目标机器的具体架构。Patrick IR 的设计灵感来源于 LLVM IR，并针对 SysY2022 语言和目标 RISC-V 硬件平台的特性，进行了特殊设计。

关键数据结构

1. Value

Value 是 Patrick IR 中的一个核心概念,用于表示程序中的各种数据和计算结果。

它本身不直接表示具体的操作，而是作为其他更具体的类型的基类。

2. 基本块 (Basic Blocks)

定义：基本块是Patrick IR 的基本组成单位，它是一组顺序执行的指令，以标签标识，并且没有控制流分支（即，基本块内的指令是连续执行的）。

作用：基本块用于组织和管理控制流。每个基本块的开始处有一个标签，基本块的结束通常有条件跳转或无条件跳转指令。

3. 指令 (Instructions)

定义：指令是Patrick IR 中的操作单元，执行具体的计算、数据传输或控制流操作。指令包括算术运算、逻辑运算、内存访问、函数调用等。

作用：指令实现了计算和数据处理的基本功能。

常见的指令包括：

算术及逻辑指令：BinaryInstructions。

内存指令：如 LoadInstruction , StoreInstruction 等。

控制流指令：如 BrInstruction , CallInstruction , ReturnInstruction 等。

4. 函数 (Functions)

定义：函数是由一系列基本块和指令组成的代码块，代表一个可以被调用的程序单元。

作用：函数用于封装代码逻辑，实现模块化编程。每个函数由函数签名、参数和返回类型定义，以及其包含的基本块。

5. 类型系统 (Type)

定义：Patrick IR 具有丰富的类型系统，包括基本类型（如整数、浮点数）、数组类型和指针类型。

作用：类型系统提供了对数据的精确描述，支持类型安全和优化。指令和操作数的类型定义了如何处理数据以及如何优化。

6. 全局变量及函数参数 (Variables)

定义：Variables 是定义在函数外部的变量

作用：用于在整个模块中共享数据。它们可以在程序的任何地方被访问和修改。

7. 模块 (Module)

定义：模块是 Patrick IR 的顶层容器，包含了多个函数和全局变量。

作用：模块用于组织和管理 Patrick IR 代码的整体结构，辅助 IR 构建。一个模块对应一个编译单元，包含编译后的所有函数和数据。

2. 中端优化

Patrick-CC 支持添加 Analysis/Transform Pass 对 IR 进行优化。

3. 编译器后端

Patrick-CC 编译器的后端部分包括以下几个关键组件：

1. 数据结构

操作数 (MOperand)：用于表示中间表示中的操作数，如整数寄存器、浮点寄存器、立即数等

指令 (MInst)：用于表示中间表示中的指令，如二元运算、分支、比较、加载/存储等。

基本块 (MBlock)：用于表示函数中的基本块，包含基本块的定义和操作。

函数 (MFunc)：用于表示函数，包含函数的定义和操作。

程序 (MProj)：用于表示整个程序，包含程序的定义和操作。

使用 (MIUse)：用于追踪操作数的使用情况。

2. 构建 MIR，将中间表示 IR 转换为 MIR。

包括建立中端基本块与后端基本块的映射关系、前驱后继关系

将函数参数转换为 MIR 指令

将各种类型的 IR 指令转为 MIR 表示。

3. 翻译 MIR 到 RISC-V，将 MIR 转换为 RISC-V 汇编语言。

包括翻译寄存器名称

翻译指令类型

翻译全局变量等。

代码引用

1. 中端：

- a. 支配树分析算法 (src/analysis/domTreeAnalysis.h, src/analysis/domTreeAnalysis.cpp)

<https://github.com/dtcxzyw/cmmc/blob/main/cmmc/Analysis/DominateAnalysis.cpp>

- b. mem2reg算法 (src/transform/mem2reg.cpp,src/transform/mem2reg.h)

<https://gitlab.eduxiji.net/educg-group-17291-1894922/202310558201558-3109/-/blob/main/src/opt/mem2reg.cpp>

2. 后端：

- a. 寄存器分配算法 (src/backEnd/asm/registerAllocation.h)

https://gitlab.eduxiji.net/educg-group-17291-1894922/202310558201558-3109/-/blob/main/src/backEnd/asm/register_allocation.h

- b. 部分后端pass

(src/backEnd/asm/algebraicIdentity.h,src/backEnd/asm/condify.h,src/backEnd/asm/removeIdenticalMoves.h,src/backEnd/asm/removeUnusedInstruction.h,src/backEnd/asm/stackRa.h,src/backEnd/asm/simplifyAsm.h)

<https://gitlab.eduxiji.net/202310558201558/yat-cc/-/tree/riscv/src/backEnd/asm>