计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200400018

CompCert 编译器目标代码生成机制分析

杨 萍[1] 王生原[2]
1 北京语言大学信息科学学院 北京 100083
2 清华大学计算机系 北京 100084
(yangp@blcu.edu.cn)

摘 要 CompCert是著名的C语言可信编译器,是经过形式化验证的编译器的杰出代表,近年来被广泛应用于学术界和工业界的许多研发工作中。CompCert编译器的当前版本支持多种目标机结构。文中对CompCert编译器目标代码生成机制进行剖析,主要介绍其设计逻辑、翻译过程、语义保持性以及代码结构,并给出了CompCert编译器重定向设计的要点。文中工作有助于实现CompCert重定向,比如实现面向重要国产处理器的后端。

关键词:CompCert;形式化验证的编译器;目标代码生成;编译器重定向
中图法分类号 TP314

Analysis of Target Code Generation Mechanism of CompCert Compiler

YANG Ping[1] and WANG Sheng-yuan[2]
1 School of Information Science,Beijing Language and Culture University,Beijing 100083,China
2 Department of Computer Science and Technology,Tsinghua University,Beijing 100084,China

Abstract CompCert is a well-known C-language trustworthy compiler,which is one of the outstanding representatives among the formally verified compilers. In recent years,CompCert has been widely used in many research and development work in academia and industry. The current version of the CompCert compiler supports a variety of target architectures. The target code generation mechanism of CompCert compiler is analyzed,by mainly introducing the design logic,the translation,the semantic preseving and the code structure. Finally,as a summary,the key points for retargeting the CompCert compiler are given. The paper is helpful to retarget the Compcert compiler,for example,we can construct a back-end for some important domestic processor.

Keywords CompCert,Formally Verified compilers,Target code generation,Compiler retargeting

1 引言

航空、航天、高速铁路、核电能源和医疗卫生等领域的安全攸关系统(Safety-Critical System[1])一旦失效,将给人类的生命财产、社会生产和生活环境带来巨大的破坏。现代计算机技术的发展中,软件系统的安全一直是计算机系统安全性中的薄弱环节。为安全攸关系统构造一个基础的安全软件环境是需要解决的首要问题,尤其是对操作系统、编译器等基础软件来说。

编译器是产生代码的工具。对于安全攸关系统而言,必须考虑编译器引入的错误,否则高成本的源程序级验证工作可能在目标程序级失效。实际上,如航空领域的RTCA DO-178B/C标准,编译器属于需要鉴定的工具类软件,需要按照机载软件的要求一样对待。

为了保障编译器的正确性,传统的方法是用大量的测例程序进行测试。然而,如果测试用例的覆盖范围不够广泛,则可能会遗漏编译器中的错误;即便通过测试发现了错误并且做了修改,也无法保证编译器自身的正确性。对编译器的正确性进行验证是解决问题的根本途径。最严格的验证手段莫过于采用形式化方法。

形式化方法近年来发展迅猛,在系统开发方面取得了一系列重要的突破。以编译器和操作系统内核验证为代表的诸多优秀工作(如CompCert[2],seL4[3]以及CertiKOS[4]等),充分展示了基于交互式定理证明器实现大型软件形式化验证的可行性。无疑,这预示着一种技术的变革,对未来软件业,特别是安全攸关领域的软件开发,将产生深刻的影响。

CompCert[2,5]编译器是经过形式化验证的可信编译器的杰出代表。该编译器将C的一个重要子集Clight[6]翻译为汇编代码,其编译过程划分为多个阶段,词法分析和一些预处理过程之后,各个阶段的翻译或确认过程的正确性都借助证明辅助工具Coq[7-8]进行了证明,且这些证明可由独立的证明检查器检查,这是迄今最强的形式化验证手段,达到了人们所期

到稿日期:2020-04-07 返修日期:2020-07-31 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家核高基重大专项(2017ZX01030-301-003)
This work was supported by the National Science and Technology Major Project of the Ministry of Science and Technology of China (2017ZX01030-301-003).
通信作者:王生原(wwssyy@tsinghua.edu.cn)

望的最高可信程度[9]。Yang 等关于 Csmith[10]的研究工作表明:CompCert 在正确性方面的表现明显优于常用的开源或商用 C 编译器。因 CompCert 编译器的杰出成就,其代表性成果[11]的作者 Leroy 获得了 2016 年度的"Most Influential POPL Paper Award"。

CompCert 编译器几乎完全支持 C 语言标准 ISO C99,同时性能优于 GCC(-O1),能够满足多数嵌入式软件开发的需求,已经被工业界应用于航空等安全攸关领域。在学术界,近年来 CompCert 编译器更是被广泛用于构建可信软件的基础平台,如编译优化过程翻译确认程序的验证[12-13]、OS 内核的验证[4]、静态分析程序的验证[14]、以 CompCert 为后端的模型语言可信编译器[15-18],以及并发程序分开编译过程的验证[19],等等。

CompCert 编译器最初仅支持 PowerPC 处理器,后来又扩展了 IA32 和 ARM 后端,目前已扩展至可支持 64 位处理器以及开源的 RISC V 体系结构。

本文从设计逻辑、翻译过程、语义保持性以及代码结构等方面对 CompCert 编译器后端实现进行剖析,并总结了 CompCert 编译器重定向设计的要点。

近年来,我国大力发展自主处理器芯片越来越深入人心,已有不少国产处理器被用于重要的军事和民用领域。若 CompCert 可以重定向到重要的国产处理器,将对我国安全攸关领域的发展非常有益。这是本文的主要动机。

本文第 2 节对 CompCert 编译器的设计框架及后端结构进行简要介绍;第 3 节对目标汇编代码生成机制中面向处理器描述的相关模块及其代码结构进行分析;第 4 节着重介绍生成目标汇编代码的最后一个环节;第 5 节对 CompCert 编译器重定向设计的要点进行讨论和总结。

2 CompCert 编译器设计框架及后端结构

CompCert 编译器的源语言为 CompCert C,是 C 语言的大子集(几乎是完整的 ISO C99)。经过前端解析(含词法和语法分析)、符合性检查(各种静态检查)以及规范化(剔除表达式计算中的副作用,明确求值顺序等 ISO C99 标准中一些模糊的内容),编译器将 CompCert C 变换为一种简化的中间表示 Clight,后者已消除表达式的副作用,满足语义确定性。从 Clight 开始,编译器经历另外 7 个中间表示以及 10 多遍(近 20 遍)变换,最终生成目标汇编代码。当前版本支持 PowerPC,ARM,RISC-V 和 x86 等目标处理器(含 32 位和 64 位处理器)。

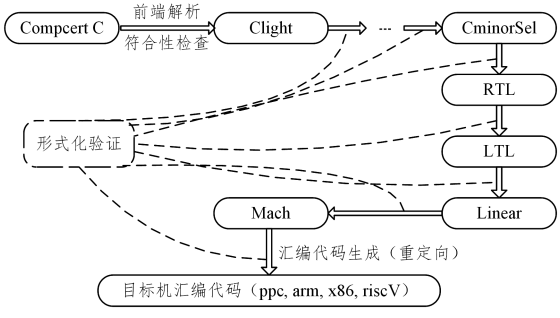

图 1 CompCert 编译器前后端示意

Fig. 1 Front and back end sketch of the CompCert compiler

CompCert 最突出的特点就是其大部分实现是在证明辅助器 Coq 环境中完成的,并且除词法分析和某些预处理过程之外,各个翻译阶段均在 Coq 中实现了正确性证明。生成 Clight 之前,语法分析过程的验证是通过对生成器所生成的 LR(1) 自动机进行确认的确认程序进行验证[20]来实现的。类似地,类型检查程序通过确认满足类型系统定义来验证其正确性。从 Clight 开始到目标汇编代码的生成,是 CompCert 编译器实现及其形式化验证最核心的部分,证明了所生成汇编代码保持了 Clight 源代码的语义。在 CompCert 中,所证明的语义保持性质可描述为一种正向的行为模拟等价关系[2,11,21]:

$$\forall B \notin Wrong, S \Downarrow B \Rightarrow C \Downarrow B$$

其中,S 和 C 分别代表当前翻译过程的源程序和目标程序,表示满足安全行为的语义。上式所描述的正向行为模拟等价关系可以刻画翻译过程的语义保持行为(通常被认为是一种反向的行为模拟等价关系),是建立在"Clight 及其后续中间语言的语义均满足确定性"这一前提之上的。

从 Clight 至汇编代码生成的多数变换,均是直接证明翻译过程本身满足语义保持性质,个别过程是通过翻译确认的方法[22]实现的,如从 RTL 翻译至 LTL 的寄存器分配过程。翻译确认的方法不是直接验证翻译程序,而是通过确认程序对翻译前后代码的语义保持性进行确认,在 CompCert 中进一步证明了确认程序的正确性。

CompCert 编译器从 Clight 层中间表示开始逐步增加低级特性。首先是细化访存操作及部分局部量的显式栈空间分配,体现于中间表示 Csharpminor 和 Cminor(二者在图 1 中被省略)。从 Cminor 到 CminorSel,生成了面向特定机器的算术/逻辑运算、布尔条件以及寻址方式。至中间表示 RTL,程序已转换为基于三地址码、伪寄存器(不限量的中间变量)的控制流图结构。从 RTL 到 LTL 的变换,程序变换为以三地址码基本块为节点的控制流图,且通过图着色寄存器分配将伪寄存器访问转换为有限个数的物理寄存器访问(并未指定具体处理器)。接着,从 LTL 到 Linear 的变换,控制流图被替换为含显式的分支和标号的线性指令序列。然后,从 Linear 到 Mach 的变换,使得活动记录的访问操作更加具体化,使用了实际偏移量而非抽象的栈帧单元的位置,同时也显式区分开了 caller 和 callee 的栈帧空间。最后,在 Mach 基础上的目标汇编代码生成,将编译器重定向到各种目标处理器结构。

除编译主体的各阶段翻译过程,CompCert 编译器也包含了某些层次的中间代码优化工作。

上面提到的各级中间语言在编译器中的关联关系可参见图 1 的前后端示意图。

3 面向处理器描述的相关模块及其代码结构

CompCert 编译器各阶段所生成的中间代码,从 CminorSel 层(见图 1)便开始涉及到了目标处理器特定的运算、布尔条件以及寻址方式,在后续的各个层次也用到了目标处理器特定的其他信息。尽管如此,CompCert 的后端组织将面向目标处理器结构的相关模块独立出来,这有益于编译器的重定向开发。

图 2 给出了含目标处理器结构信息的主要模块，这些模块针对不同处理器结构均有不同的副本。CompCert 编译器的当前版本支持 PowerPC，ARM，RISC-V and x86 等目标处理器，包括 32 位和 64 位结构。与这些模块相关的其他各类辅助模块、证明模块以及打印输出模块均未列出。

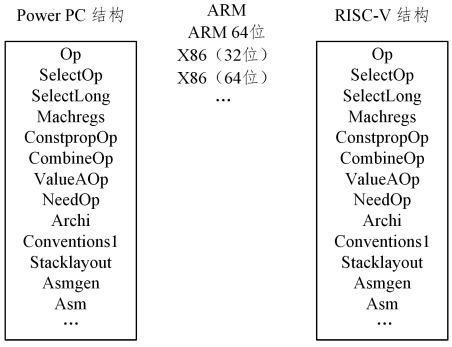

图 2　含目标处理器结构信息的主要模块

Fig. 2　Main modules with processor architecture specific information

本节从目标代码生成的若干重要环节出发，对目标处理器结构相关模块的内容和作用进行简要分析与介绍。

3.1　指令选择

CompCert 编译器的指令选择发生在从中间表示 Cminor 至 CminorSel 的转换过程中，涉及到的目标处理器相关模块包括 Op，SelectOp，SelectLong，Machregs。

模块 Op 刻画的信息包括 3 个方面：1）条件分支中的布尔条件，抽象为类型 condition；2）算术与逻辑运算，抽象为类型 operation；3）load 和 store 操作的寻址方式，抽象为类型 addressing。模块 Op 中围绕这 3 方面信息定义了相应的抽象语法、动态语义及相关内容，这些内容将在中间语言 CminorSel，RTL，LTL，Mach 及其翻译过程的相关定义和证明中用到（通过 condition，operation 和 addressing）。

指令选择过程中，除了 Op 模块外，还用到另外两个面向目标处理器的模块 SelectOp 和 SelectLong。前者使得识别出某些操作或寻址方式的组合成为可能，可以使用特定目标机器的惯用指令进行替换。后者主要用于处理 64 位整数操作的指令选择工作。在实现除法和取余操作的指令选择时，这两个模块也发挥了重要作用。

指令选择过程基于上述面向机器的模块 Op，SelectOp 和 SelectLong，返回 CminorSel 表达式和语句，再经过后续变换（生成 RTL，LTL 和 Mach 的翻译模块），最终通过面向特定目标处理器的翻译模块 Asmgen，得到该处理器的目标汇编代码。

指令选择过程还涉及另外两个机器描述模块：Archi 和 Machregs。

模块 Archi 中描述了诸如大/小端、64/32 位、对齐方式以及是否生成位置无关码等一些体系结构信息。在选择操作数时，需要考虑是 64 位还是 32 位结构；在选择寻址方式时，需要考虑是否生成位置无关码。

模块 Machregs 用于描述机器寄存器信息，如用户程序可访问的或者可用于分配的（整数/浮点）寄存器有哪些（这些寄存器不包括专用或者机器保留寄存器），指定其中某些寄存器的特殊用途，以及内部函数的参数限定等。指令选择阶段用到了内部函数参数限定的信息。

3.2　涉及处理器特性的中间代码优化

在 RTL 层中间语言，CompCert 实现了多种优化相关工作。其中，常量传播（constant propagation）、公共子表达式删除（common subexpression elimination）以及冗余代码删除（redundancy elimination）过程涉及到由目标处理器相关模块提供的信息。

常量传播优化过程用到的目标处理器相关模块（CminorSel 之后各中间语言及其相关的翻译过程均涉及到模块 Op，后面叙述时统一略去）包括 ConstpropOp，ValueAOp，以及 Archi 和 Machregs。ConstpropOp 模块用于描述强度削弱规则，尽可能使用更廉价的运算和条件，充分利用特定机器自身的优势。ValueAOp 用于静态求值分析。此外，该阶段用到了 Machregs 中描述的内部函数参数限定，以及 Archi 中描述的是否为 64 位结构和是否生成位置无关码等信息。

公共子表达式删除优化过程用到的目标处理器相关模块包括 ValueAOp 和 CombineOp。ValueAOp 的作用与前面一样，用于静态求值分析。CombineOp 刻画了在这一优化过程中识别可组合操作的方式、寻址方式和条件。

冗余代码删除优化过程用到了目标处理器相关模块 NeedOp，该模块提供了针对操作和条件的冗余性分析函数。

3.3　寄存器分配

从 RTL 到 LTL 的变换过程中，寄存器分配模块（Allocation）借助外部函数来实现寄存器的分配，并采用翻译确认的方案验证正确性[22]，即进行翻译后的确认（posteriori validation），并对确认程序进行证明。

在目前的 CompCert 版本中，与寄存器分配相关的外部函数包括 3 个 OCaml 模块：Regalloc，IRC，和 XTL。Regalloc 通过干扰图上的图着色（coloring of an interference graph）算法分配将伪寄存器访问转换为有限个数的物理寄存器访问。具体算法（George 和 Appel 的 Iterated Register Coalescing 图着色算法）在 IRC 模块中实现。XTL 定义了这一寄存器分配过程中用到的一种中间语言。

此外，在模块 Locations 中，根据寄存器分配的结果对 RTL 伪寄存器进行了精化。

在模块 Regalloc，IRC，XTL 以及 Locations 中，均涉及到访问面向特定目标处理器的 Machregs 模块的信息，如可分配的寄存器列表、是否为专用或保留寄存器等。

模块 Regalloc 和 IRC 均使用了面向特定目标处理器的 Conventions1 模块的信息。模块 Conventions1 描述了特定机器 ABI 的函数调用约定（function calling conventions）以及使用机器寄存器和栈帧单元的其他约定信息。

模块 Allocation 还用到了模块 Archi 中面向特定目标机结构的描述，包括是否为 64 位机器以及大小端信息等。

3.4　调试信息综合

模块 Debugvar 作用于中间表示 Linear，用来计算携带调

试信息的局部变量的活跃范围，涉及 Machregs 机器描述中的寄存器使用约定信息。

3.5 生成活动记录操作

从 Linear 到 Mach 的变换过程中，模块 Stacking 负责活动记录的具体布局(layout of activation records)，其中目标处理器以及 ABI 相关的方面在模块 Stacklayout 中描述。

Stacklayout 模块中需要清楚定义活动记录相关的环境信息，给出活动记录的结构(各子部分的内容、次序和大小)。例如，对于 ARM 和 RISC-V 结构，从栈帧的底部(偏移量最小的位置)到顶部所存放的内容依次是：要返回结果的函数调用参数，动态链，返回地址，局部量栈帧单元，callee 负责保存的寄存器值存放单元，栈式分配的临时数据单元。各个子部分的数据大小，即函数调用所需栈帧资源上界的计算，在 Linear 层由模块 Bounds 实现。

Bounds 模块在计算资源上界时需要访问模块 Machregs 中刻画的与寄存器使用密切相关的机器描述信息。关于模块 Machregs 的简单介绍，参见 3.1 节。

3.6 生成目标汇编代码

CompCert 编译器翻译过程的最后一步，即生成目标汇编代码，在面向目标处理器的模块 Asmgen 中实现。Asmgen 中所定义的翻译过程，将最后一层中间语言 Mach 代码翻译至对应处理器的汇编语言代码。目标处理器的汇编语言在模块 Asm 中定义。

遵循由多个机器描述模块 Op, Machregs, SelectOp, SelectLong, Conventions1, Archi, Stacklayout 等的定义和限定，以及模块 ValueAOp, ConstpropOp, CombineOp 和 NeedOp 等面向优化的机器描述信息，Mach 代码已经充分体现了目标处理器的低级代码特性。然而，最终编译器还是要产生符合模块 Asm 定义的目标汇编代码。

3.7 体现目标机结构的中间代码和目标代码的格式化输出

中间代码和目标代码的格式化输出在编译器的设计、实现、代码追溯以及调试过程中发挥着重要作用。

某些中间代码的格式化输出与目标机结构信息相关，如打印模块 PrintLTL, PrintLTLin 和 PrintXTL 用到了依赖于目标机结构的打印模块 PrintOp，而打印模块 PrintMach 用到了由模块 Machregsaux 定义的机器寄存器的相关函数。

模块 PrintOp 提供了目标机结构相关的运算、条件和寻址方式的格式化打印函数。

目标代码输出模块 PrintAsm 用到了模块 TargetPrinter 中定义的目标处理器汇编代码的格式化输出函数。

4 目标汇编代码的生成

本节进一步介绍从 Mach 到 Asm 的翻译以及汇编代码输出的相关信息。在涉及到 Asm 中间语言以及汇编代码格式化输出的内容时，借助 RISC-V 结构进行说明。

首先是关于 Mach 和 RISC-V Asm 的抽象语法概述，然后是从 Mach 到 RISC-V Asm 的翻译以及相应证明模块的简介。

4.1 Mach 语言抽象语法

相比 Linear，Mach 中对活动记录的访问使用了实际偏移量，而非抽象的栈帧单元的位置，同时也显式区分了 caller 和 callee 的栈帧空间。在此基础上，通过依赖于机器的模块生成面向处理器的汇编代码。

以下列举了 Mach 语言抽象语法中指令定义的片断。

$i ::=$ setstack(r,τ,δ) // 从寄存器写入栈
| getstack (δ,τ,r) // 从栈读出到寄存器
| op (op, r^*, r) // 算术/逻辑运算指令
| call $(sig,(r \mid id))$ // 函数调用指令
…
| goto (l) // 无条件转移指令
| label (l) // 标号指令
| return // 函数返回指令

其中，Mach 指令的名称与文献[21]中的表达相同，容易对应到实际代码(如 Msetstack, Mgetstack, …)。

这里忽略了其他一些 Mach 指令，如 load, store, tailcall, …。

setstack(r,τ,δ)表示将寄存器 r 的内容以 τ 描述的数据类型写入当前栈中偏移量为 δ 的单元。getstack(δ,τ,r)表示从栈中偏移量为 δ 的单元读出类型为 τ 的值到寄存器 r。算术/逻辑运算指令 op(op, r^*, r)中，op 是具体的运算，另两个参数分别表示输入和输出参数寄存器。call(sig,id)和 call(sig,r)表示直接调用和间接调用。其他类型的指令具有通常的含义。

基于指令定义 i，Mach 代码(指令序列)c 可抽象描述为：

$c ::= i^*$

另外，Mach 函数(同文献[21])可抽象定义为：

$F ::=$ { sig$=sig$; // 函数 signature
code$=c$; // 指令列表
stacksize n; // 栈帧大小
link$=\delta$; // 动态链
retaddr$=\delta$} // 返回地址

4.2 RISC-V ASM 语言简介

CompCert 中，ASM 是面向目标处理器的中间语言。如图 2 所示，每个目标处理器结构对应各自的 ASM 模块以及其他相关模块。各个机器的相关模块，结构上相似，但内容方面各有特性。我们仅以 RISC-V 结构的 ASM 语言为例进行简述。

RISC-V 的 ASM 模块中定义了 32 个整数寄存器：

$r_i ::= X1 \mid X2 \mid \cdots \mid X31$

包括这 31 个整数寄存，再加上一个值恒为 0 的 X0 寄存器(零寄存器)。

ASM 中还定义了 32 个浮点寄存器：

$r_f ::= F0 \mid F1 \mid \cdots \mid F31$

此外，ASM 中还包括一个作为程序计数器的特殊寄存器 PC。

在 32 个整数寄存器中，$X5, X6, \cdots, X30$ 为可分配给用户程序的寄存器(可参见 RISC-V 的 Machregs 模块所描述的信息)。

进一步,从模块 ASM 和 Machregs 的定义中可知,RISC-V 约定返回地址寄存器(RA)为 $X1$,栈指针寄存器(SP)为 $X2$,全局指针寄存器(GP)为 $X3$,以及线程指针寄存器(TP)为 $X4$。同时,寄存器 $X31$ 保留为临时工作寄存器(存放临时量)。

ASM 模块中,将 RISC-V 指令集分为 32 位整数、64 位整数、乘除指令、单精度浮点以及多精度浮点等若干子集。

对于 32 位或 64 位整数算逻指令,CompCert 会在 ASM 的抽象语法层面通过特定的指令(加 W/L 扩展的名字)将其约束至 32/64 位结果。比如,若 ADD 是满位数(32 位或 64 位)指令,则 ADDW 用作 32 位结果的指令,而 ADDL 用作 64 位结果的指令。在实际的格式输出(见模块 TargetPrinter)时,对于 32 位模式,ADDW 会打印成 ADD,而 ADDL 不合法;对于 64 位模式,ADDW 会照旧打印成 ADDW,而 ADDL 会打印成 ADD。

RISC-V 指令中对立即数的各种限制没有体现在描述的抽象语法中,而是在目标汇编代码生成过程(见模块 Asmgen)中进行检查。

ASM 模块中所描述的伪指令,有一些(如栈帧分配 allocframe 和栈帧释放 freeframe 等伪指令)会在目标汇编代码生成附加模块 Asmexpand 中被转换成其他抽象指令/伪指令,然后再利用模块 TargetPrinter 中定义的格式输出为实际的 RISC-V 指令/伪指令序列。还有一些伪指令(如标号 label、装入符号地址 loadsymbol 以及装入 64 位立即数 loadli 等),则会利用模块 TargetPrinter 中的相应函数直接输出为实际的 RISC-V 指令/伪指令序列。

4.3　Mach 翻译至 ASM

Mach 翻译至 RISC-V ASM 的过程,在对应于 RISC-V 的 Asmgen 模块中实现。整体翻译函数(transf_program)可分解为程序各个子部分的翻译函数,直至每条 Mach 指令的翻译。下面仅举几个 4.1 节中列举的 Mach 指令的翻译例子。

"寄存器写入栈"指令 setstack(r,τ,δ)经 Asmgen 模块被翻译至"sw(r,SP,δ)"(ASM 模块的实际代码中,sw 需要加前缀"P",即 Psw,以下各指令类似,不再赘述)。然后,在模块 TargetPrinter 中,"sw(r,SP,δ)"被格式输出为实际的 RISC-V 指令"sw r,δ(sp)"。这里,SP 或 sp 代表栈指针寄存器。

"栈读出到寄存器"指令 getstack(δ,τ,r)经 Asmgen 模块被翻译至"lw(r,SP,δ)",后者最终被格式化输出为"lw r,δ(sp)"。

"直接调用"指令 call(sig,id),经 Asmgen 模块被翻译至"jal_s(id,sig)",将被格式化输出为 RISC-V 指令"call id"。其中,sig 会在上下文中进行处理,下同。

"间接调用"指令 call(sig,r),经 Asmgen 模块被翻译至"jal_r(r,sig)",将被格式化输出为 RISC-V 指令"jalr r"。

加法指令 op(add,$r1,r2,r$),经 Asmgen 模块被翻译至"addw $r,r1,r2$",将被格式化输出为 RISC-V 指令"addw r,$r1$,r2"(最终"addw"会根据 32 位或 64 位机器有不同的输出,参见 4.2 节)。

无条件转移指令 goto(l),经 Asmgen 模块被翻译至"j_l(l)",将被格式化输出为"j l"。

标号指令 label(l),经 Asmgen 模块翻译后无本质改变,最终被格式化输出为"l:"。

函数返回指令 return,经 *Asmgen* 模块将会生成 *epilogue* 代码序列,为简化讨论,这里略去。

4.4　Mach 至 ASM 的语义保持性

为了证明从 Mach 至 ASM 翻译过程(Asmgen. transf_program)的正确性(语义保持性),在模块 Mach 和 ASM 分别给出了 Mach 语言和 ASM 语言的操作语义(operational semantics)定义,在模块 Asmgenproof 中给出了完整的证明代码。

模块 Asmgenproof 的实现用到了另外两个证明模块:Asmgenproof0 和 Asmgenproof1。模块 Asmgenproof0 提供了汇编代码生成过程中机器无关部分的证明,而模块 Asmgenproof1 则是一些机器相关特性的证明。

限于篇幅,本文不对 Mach 和 ASM 的操作语义进行定义,亦不对 Asmgen 翻译过程的语义保持性证明展开讨论。

5　CompCert 后端重定向的设计要点

CompCert 编译器基于定理证明方法,实现了 C 语言到 ASM 语言的可信翻译,保证了翻译过程的安全可靠。因此,CompCert 编译器的翻译转换过程是一个自始而终的完备整体,且每一步转换都经过证明。其前端主要处理 C 语言特性,而后端从中间语言 Cminor 到 CminorSel 的指令选择过程开始,又逐步实现寄存器分配、堆栈操作、机器结构描述、汇编语言定义等多项功能,翻译形成最终的汇编语言。

CompCert 编译器的后端重定向,需要结合整体架构的前端解析、前端 C 语言翻译、后端转换控制及机器架构指令系统的统筹调整来设计。在充分利用 CompCert 编译器原有处理流程结构及形式化验证模块的基础上,筛选目标体系结构及 CompCert 现有指令集的相似部分能直接借鉴使用,差别较大的部分需要进行结构调整及多重手段控制。下面主要从寄存器分配、指令集筛选调整、堆栈策略调整及 ASM 文本输出 4 个方面进行介绍。

5.1　寄存器分配策略

CompCert 编译器的寄存器分配算法以干扰图上的图着色算法为基础。该算法是 CompCert 的核心工作之一,通常能够满足寄存器分配的需求,在重定向时可直接沿用该算法实施寄存器分配。由于 CompCert 的这个环节采用了基于翻译确认[22]的形式化验证方法,因此证明过程具有良好的可重用性,根据需要修改或改进分配算法不会影响到确认过程及其证明。

寄存器使用策略严格按照 CompCert 编译器对寄存器的要求,根据具体机器特性和约定实施分类。寄存器类型主要包括通用寄存器、浮点运算寄存器、控制和专用寄存器。

CompCert 目前支持 32 位和 64 位架构,对于机器架构不支持的超长类型(如 128 位浮点型),应该在 C 语言处理前端

实施判断控制，严格限制或特别标识超长类型的使用。如机器架构支持超长类型寄存器，应该设置超长寄存器，用以处理超长数据类型。如指令集支持寄存器联合处理超长类型，可对处理过程进行调整，联合分配调用寄存器，或直接实施寄存器捆绑，严格控制寄存器分组，以实现超长类型数据的处理。上述超长类型寄存器分配策略的示意图如图 3 所示。

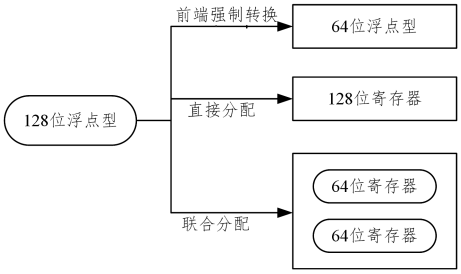

图 3 超长类型寄存器的分配策略

Fig. 3 Register allocation of very long data type

5.2 指令集筛选调整

目前，Compcert 后端支持 powerPC，ARM，X86 和 riscV 等架构，目标指令集与现有指令集语义甚至语法大概率存在相似部分。因此，重定向时需要大量对比二者之间的相似程度及可复用范围，对于语义、语法相同的指令，保留其原有结构或者进行微调，以适应目标指令集的要求。其中主要保留的部分包括现有指令集的语义和语法、翻译转换手段及形式化证明的相关部分。

相对于现有指令集，目标指令集的差异化、个性化指令将逐条筛选定义。根据需要调整整体翻译架构及形式化证明过程，增加新指令的语法、语义定义。对于语义相近的指令，合理论证并调整语法定义。对于副作用少的指令块的使用，也要进行合理化论证调整，然后将其作为整体定义输出，以确保输出代码的正确性、安全性。全部定义及翻译过程应严格按照 Compcert 现有框架实施。对于那些与 Compcert 框架背离的极特殊情况，要用相似指令或者功能相同的指令块代替，或者通过前端 C 语言解析翻译限制。

对于目标机器的 ABI 函数，将按照目标机器定义进行调整，重新分配定义 callee 寄存器及 caller 寄存器等。另外，在 SelectOp 模块增加目标机器的惯用指令，提高翻译效率，提升生成代码的高效性、安全性。

目标机器指令的生成过程如图 4 所示。

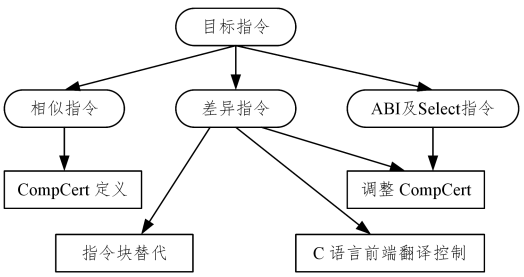

图 4 目标机器指令生成方法

Fig. 4 Code generation of target machine instructions

5.3 堆栈策略调整

Compcert 编译器的主体使用推展结构，从栈帧的底部到顶部，通常存放的内容主要包括：要返回结果的函数调用参数，动态链，返回地址，局部量栈帧单元，callee 负责保存的寄存器值存放单元，栈式分配的临时数据单元，等等。相应顺序根据各机器架构的不同而有所调整。

目标架构可能存在的不同堆栈内容和布局，将根据实际情况进行结构调整，并完善相关内容的形式化验证。

5.4 ASM 文本输出

ASM 文本输出内容以 Compcert 编译器现有架构输出样式为基础模板，针对目标机器及目标链接器输出定制格式和定制样式的标准化 ASM 文本。常用指令应按照固定模式输出；而对于特定功能的指令模块，将依据设计要求及形式化验证的语法和语义要求进行设计，统一标准化输出。

此外，还需要在总控流程内添加目标机器架构的编译流程，该流程只生成目标机器可信编译后的可执行文件；同时应支持针对目标机器的 ASM 打印，打通从 C 语言至目标机器架构的可信编译。

以上仅提到 CompCert 后端重定向设计的基本要点，而在实际开发中需要考虑的细节还有许多。第 3 节和第 4 节所提到的各个环节和模块中的各种相关定义、翻译过程以及证明过程，均需要有序地修改/扩充，先重定向最基本的翻译逻辑，然后根据需要补充优化等环节，必要时（比如上面提到的 128 位超长类型的支持）可能会涉及到众多模块的修改和调整。

结束语 本文对 CompCert 后端结构，特别是与目标处理器描述相关的模块及其代码结构进行了梳理，分析了 CompCert 编译器重定向的基本方面和设计要点。本文工作的出发点是将 CompCert 重定向到重要的国产处理器，这有益于我国安全攸关领域的发展。随着具体工作的深入和积累，本文部分内容有待进一步充实或者纠偏。

第 2 节对 CompCert 翻译过程的正确性证明进行了基本描述，但限于篇幅，本文未对后端相关语言的操作语义定义以及代码生成过程的语义保持性证明展开讨论。

参 考 文 献

[1] KNIGHT J C. Safety critical systems: challenges and directions [C]// Proceedings of the 24th International Conference on Software Engineering (ICSE 2002). 2002: 547-550.

[2] LEROY X. Formal verification of a realistic compiler[J]. Communications of The ACM, 2009, 52(7): 107-115.

[3] KLEIN G, ELPHINSTONE K, HEISTER G, et al. seL4: formal verification of an OS kernel [C]// Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09). 2009: 207-220.

[4] GU R H, SHAO Z, CHEN H, et al. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels [C]// Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16). 2016: 653-669.

[5] CompCert Main Page [EB/OL]. http://compcert.inria.fr/.

[6] BLAZY S,LEROY X. Mechanized semantics for the Clight subset of the C language [J]. Journal of Automated Reasoning, 2009,43(3):263-288.

[7] BLAZY S,LEROY X. Mechanized semantics for the Clight subset of the C language [J]. Journal of Automated Reasoning, 2009,43(3):263-288.

[8] BERTOT Y,CASTRAN P. Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions [M]// Texts in Theoretical Computer Science. An EATCS Series,Springer Verlag,2004.

[9] MORRISETT G. Technical Perspective:A Compiler's Story [J]. Communications of the ACM,2009,52(7):106-106.

[10] YANG X,CHEN Y,EIDE E,REGEHR J. Finding and understanding bugs in C compilers [C]// Proceedings of the 2011 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011). 2011:283-294.

[11] LEROY X. Formal certification of a compiler back-end or:programming a compiler with a proof assistant [J]. Proceedings of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages,2006,41(1):42-54.

[12] TRISTAN J B,LEROY X. A simple,verified validator for software pipelining [C]// POPL. 2010:83-92.

[13] BARTHE G,DEMANGE D,PICHARDIE D. A formally verified SSA-based middle-end - Static Single Assignment meets CompCert [C]// Programming Languages and Systems(ESOP 2012). 2012:47-66.

[14] JOURDAN J H,LAPORTE V,BLAZY S,et al. A formally-verified c static analyzer [C]// Proceedings of POPL'15. Mumbai, India,ACM Press. 2015:247-259.

[15] BOURTE T,BRUN L,et al. A Formally Verified Compiler for Lustre [C]// Proceedings of the Programming Language Design and Implementation. 2017:586-601.

[16] SHANG S,GAN Y K,et al. Key Translations of the Trustworthy Compiler L2C and Its Design and Implementation[J]. Journal of Software,2017,28(5):1233-1246.

[17] KANG Y X,GAN Y K,WANG S Y. Key Analysis and Comparison of two Trustworthy Compilers Vélus and L2C for Synchronous Languages [J]. Journal of Software, 2019, 30(7): 2003-2017.

[18] BOURTE T,BRUN L,POUZET M. Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset [C]// Proceedings of the 47th ACM SIGPLAN Symposium on Principles of Programming Languages. New Orleans, LA, USA,2020.

[19] JIANG H,LIANG H,et al. Towards Certified Separate Compilation for Concurrent Programs [C]// Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2019:111-125.

[20] JOURDAN J H,POTTIER F,LEROY X. Validating LR(1) parsers [C]// Programming Languages and Systems -- 21st European Symposium on Programming(ESOP 2012). 7211 of Lecture Notes in Computer Science. 2012:397-416.

[21] LEROY X. A formally verified compiler back-end [J]. Journal of Automated Reasoning,2009,43(4):363-446.

[22] PNUELI A,SIEGEL M and SINERMAN E. Translation Validation [C]// Proceedings of TACAS'98. 1998:151-166.

YANG Ping, born in 1964, Ms.D, associate professor. Her main research interestes include programming languages and systems, artificial languages, compilers and artificial intelligence.

WANG Sheng-yuan, born in 1964, Ph.D, associate professor. His main research interestes include programming languages and systems, compilers and formal methods.