基于指令生成约束的RISC-V测试序列生成方法

刘 鹏[①] 胡文超[①] 刘德启[②] 韩晓霞*[①] 刘扬帆[②]

[①](浙江大学信息与电子工程学院 杭州 310027)

[②](合芯科技有限公司 广州 510725)

摘 要：为了避免处理器受到指令缺陷的威胁，该文提出基于指令生成约束的RISC-V测试序列生成方法，构建测试指令序列生成框架，实现测试指令生成及指令缺陷检测，解决现有测试指令序列生成方法约束定义困难和收敛速度慢的问题。在该方法中，首先，根据指令集架构规范和指令验证需求定义指令生成约束，包括指令格式约束、通用功能覆盖约束和特殊功能覆盖约束，以解决随着指令数量增多约束定义的困难，提高可复用性；然后，定义启发式搜索策略，通过统计覆盖信息，加快覆盖率收敛速度；最后，基于启发式搜索策略构造求解算法，实现满足指令生成约束的测试序列生成。实验结果表明，与现有方法相比，在覆盖所有指令验证需求的前提下，结构覆盖率和数值覆盖率的收敛时间分别减少了85.62%和57.64%。利用该框架对开源处理器进行检测，可以定位到在处理器译码和执行阶段引入的指令缺陷，为处理器指令缺陷检测提供了有效的方法。

关键词：处理器；RISC-V；指令缺陷检测；约束指令生成

中图分类号：TN407 **文献标识码**：A **文章编号**：1009-5896(2023)09-3141-09

DOI: 10.11999/JEIT230480

A RISC-V Test Sequences Generation Method Based on Instruction Generation Constraints

LIU Peng[①] HU Wenchao[①] LIU Deqi[②] HAN Xiaoxia[①] LIU Yangfan[②]

[①](*College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou* 310027, *China*)

[②](*HEXIN Technologies Co., Ltd., Guangzhou* 510725, *China*)

Abstract: In order to avoid the threat of instruction defects to the processor, this paper proposes a RISC-V test sequence generation method based on instruction generation constraints. A test instruction sequence generation framework is constructed based on this method to achieve test instruction generation and instruction defect detection, while addressing the challenges in defining constraints and slow convergence speed in existing test instruction sequence generation methods. Firstly, the instruction generation constraints are defined according to the instruction set architecture specification and instruction verification requirements. These constraints include instruction format constraints, general coverage constraints, and particular coverage constraints, aiming to solve the challenges in defining constraints as the number of instructions increases and improve reusability. Then, a heuristic search strategy is applied to accelerating the convergence rate of coverage by utilizing statistical coverage information. Finally, a solving algorithm is constructed based on the heuristic search strategies to generate test sequences that satisfy the instruction generation constraints. The experimental results show that, compared with the state-of-the-art methods, the convergence time of structural coverage is reduced by 85.62% and numerical coverage is reduced by 57.64%, under the premise of covering all instruction verification requirements. By using this framework to detect open-source processor, instruction defects introduced in the processor decoding and execution stages can be located, providing an efficient method for detecting processor instruction defects.

Key words: Processor; RISC-V; Instruction defect detection; Constrained instruction generation

收稿日期：2023-05-24；改回日期：2023-08-23；网络出版：2023-08-31

*通信作者： 韩晓霞 hanxx@zju.edu.cn

1 引言

处理器被广泛应用在信息系统中，处理器硬件安全是信息系统能被安全使用的前提。隐藏在处理器中的功能性缺陷会导致处理器功能异常、信息被篡改和泄露，严重威胁处理器的硬件安全。处理器的功能性缺陷泛指因设计规范或硬件描述语言的局限性，或者设计人员考虑不周等情况，导致集成电路内部部分电路出现错误[1]。在处理器众多功能性缺陷中，基于指令集架构的指令缺陷是其中最为关键的缺陷之一。指令缺陷是指处理器硬件实现中存在的设计缺陷，其导致指令集架构规范中明确定义行为的指令被错误的执行。指令缺陷会给处理器系统带来严重的安全威胁，甚至造成巨大的经济损失[2]。为了使处理器免受指令缺陷的威胁，需要一种方法和工具辅助设计人员快速准确地检查处理器中可能存在的指令缺陷。由于RISC-V(Reduced Instruction Set Computer-Five)开源、简洁以及模块化的特性，RISC-V指令集架构成为自主设计处理器的新方向[3]。因此，本文以RISC-V处理器为例阐述测试序列生成方法用于检测指令缺陷，其他指令集架构的处理器指令缺陷检测可借鉴RISC-V的方案实现。

模拟仿真是检测指令缺陷的主要方式，由验证人员向处理器注入测试指令序列，将处理器执行结果与预期结果进行对比，对不一致的结果进行分析，检测存在的指令缺陷[4]。因此，检测指令缺陷的关键就是高效获得高覆盖率的测试指令序列[5]，以快速触发指令缺陷。目前存在基于指令格式的约束随机生成和基于指令格式的覆盖率驱动随机生成的两类测试指令序列生成方法，指令格式保证生成的指令符合指令集架构规范[6]。

根据不同指令验证需求自定义各种指令的功能覆盖约束是基于指令格式的约束随机生成方法的一个研究方向， Herdt等人[7]提出了基于约束规范的随机生成方法，在指令格式约束基础上遍历预定义的各种功能覆盖约束情况得到测试指令序列。Chupilko等人[8]提出基于执行路径的约束生成模型，通过预定义指令的生成顺序以获取测试指令序列。谭坚等人[9]预先对结果操作数、操作数之间、指令内部以及浮点操作数类型进行功能覆盖约束定义，再对定义的每种约束情况进行求解，生成测试指令序列。上述方法都需要依赖预定义的功能覆盖约束，然后对每种约束情况进行遍历求解，以获得测试指令序列，虽然可以对已定义的约束实现全覆盖，但是缺少对功能覆盖约束的归纳，随着指令数量的增多，预定义各种指令的功能覆盖约束愈加困难，同时人为定义各种约束易造成约束遗漏，而且缺少已生成指令的覆盖统计，存在指令覆盖率收敛速度慢的问题。

基于指令格式的约束随机生成方法的另一个研究方向就是问题映射，IBM公司设计的针对POWER处理器的Genesys-Pro随机指令生成器[10]将测试指令序列生成问题映射为约束满足问题(Constraint Satisfaction Problem, CSP)，然后根据指令验证需求，通过维持弧相容(Maintaining Arc Consistency, MAC)算法进行约束求解。该方法依赖于指令验证需求生成测试指令序列，在不了解指令验证需求情况下，无法自动生成高覆盖率的测试指令序列，同时也存在由于缺乏覆盖统计导致收敛速度慢的问题。

基于指令格式的覆盖率驱动随机生成方法不需要定义功能覆盖约束，而是随机生成一段测试指令序列，直接送入待验证处理器中测试执行，再利用机器学习等技术对待验证处理器的执行结果进行分析学习，根据反馈得到的已覆盖指令信息对未覆盖到的指令验证需求进行检测，以指导后续指令生成。Braun等人[11]采用贝叶斯网络、Pfeifer等人[12]采用强化学习以及Herdt等人[13]采用模糊测试，实现了测试指令序列的在线生成。这类方法可以针对性地提高某一款具体处理器的指令覆盖率，但是无法保证对指令验证需求的全覆盖，同时由于需要依赖待验证处理器的执行结果，无法独立使用。

针对上述方法的不足，本文以RISC-V指令集架构为研究对象，提出了基于指令生成约束的RISC-V测试序列生成方法，并构建了测试指令序列生成框架，能够自动生成高覆盖率的测试指令序列，用于RISC-V处理器的指令缺陷检测。本文主要贡献如下：

(1)定义了指令生成约束，通过对指令集架构规范和指令验证需求进行归纳定义指令生成约束，具体包括指令格式约束和功能覆盖约束。其中功能覆盖约束又分为通用覆盖约束和特殊覆盖约束。通用和特殊覆盖约束的组合不仅克服了随着指令增多预定义指令约束的困难，避免了约束遗漏，还增强了约束的可复用性。

(2)提出了基于指令生成约束的求解算法，实现测试指令序列的高效生成。该算法的核心是启发式搜索策略，利用集成的指令集模拟器收集已覆盖的指令生成约束的信息，反馈引导后续指令生成，以实现对指令生成约束进行全覆盖，同时加快覆盖率的收敛速度，提升测试指令序列生成的效率。

(3)构建了测试指令序列生成框架，实现测试指令序列自动生成到处理器指令缺陷检测的完整过程，从而实现对处理器中可能存在的指令缺陷进行高效准确定位。

2　研究背景和基础知识

2.1　RISC-V指令集

加州大学伯克利分校推出的第五代开源指令集架构RISC-V的核心是RV32I基础指令集[14]，其余功能可以通过模块化的标准扩展或自定义扩展实现。RISC-V指令集架构定义的模块化扩展指令集包括M(整数乘/除法指令)，A(内存原子操作与加载保留/条件存储指令)，F(单精度浮点指令)，D(双精度浮点指令)，C(压缩指令)等。RV32I是固定不变的，处理器设计者可以自由选择硬件设计包含的模块化扩展指令集。RV32I支持32位地址空间，包括6种(R, I, S, B, U, J)指令格式，具有32个通用整数寄存器(X0～X31)，其中X0为硬连线零[14]。RV32I中的指令分为运算指令、访存指令、分支指令以及系统指令。前面3种指令通过访问寄存器和立即数实现指令功能，系统指令用于获取处理器当前的系统状态并对系统状态进行控制(包括中断、异常处理等)。

2.2　CSP模型定义

CSP模型，即约束满足问题模型，用3元组(V, D , C)进行表示[15]，$V=\{v_1,v_2,\cdots,v_n\}$表示变量的集合，$D=\{d_1,d_2,\cdots,d_n\}$表示每个变量对应的值域，$C=\{c_1,c_2,\cdots,c_m\}$表示变量之间的约束集合，作用于变量从而限制变量的取值。例如v_1变量对应值域d_1，c_1可以表示为限制v_1取值为零的取值约束。利用CSP模型可以将测试指令序列生成问题用一个标准的数学模型进行表示，便于指令生成。

2.3　指令CSP模型定义

基于CSP模型定义，将其与测试指令序列生成问题进行结合，定义指令CSP模型。

定义1：单条指令CSP模型。每条指令构造一个CSP模型，V表示单条指令下寄存器变量和立即数变量的集合；D表示寄存器变量和立即数变量的值域；C表示寄存器变量和立即数变量的取值约束。例如：生成一条add加法指令需要完成两个加数相加，并保存加法结果。构造的加法指令CSP模型(V, D, C)中$V=\{\mathrm{rs1},\mathrm{rs2},\mathrm{rd}\}$，寄存器变量rs1, rs2分别表示源操作数1和2，rd表示目的寄存器用于保存加法结果。$D=\{d_1,d_2,d_3\}(d_1=d_2=d_3\in\{0,31\})$表示3个寄存器变量对应的值域，在32个整数寄存器中进行选择。$C=\{\varnothing\}$表示该加法指令没有约束限制，设计者亦可增加取值约束，生成所需的加法指令，例如限制 rd的取值不为零。

定义2：指令序列CSP模型。生成多条指令的CSP模型，V包括各指令操作码变量、寄存器变量和立即数变量；D表示上述3个变量的值域；C表示3个变量的取值约束。

3　测试指令序列生成框架

为了检测处理器硬件实现中的指令缺陷，解决测试指令序列自动生成的问题，本文提出了测试指令序列生成通用框架。如图1所示，一个完整的测试指令序列生成周期分为输入、生成、输出和检测4个阶段。

输入阶段，验证人员自定义测试模板，初始化寄存器值，指定生成的指令及数量，同时可以修改配置文件和指令宏文件。其中配置文件定义存储空间和指令地址空间范围，指令宏文件定义一系列指令操作码组合。在生成阶段，解析测试模板及相应文件，调用基于指令生成约束的求解算法，保证生成符合指令集架构规范和指令验证需求的测试指令序列。在生成过程中，集成指令集模拟器对生成的指令进行实时模拟，记录指令模拟的结果，反馈给求解算法引导后续指令生成。最后，在输出和检测阶段，将生成的测试指令序列送入待验证处理器中测试执行，获取执行结果，并和指令集模拟器的模拟结果进行对比，通过确定缺陷范围，定位缺陷指令，确定设计缺陷3个步骤，逐步缩小范围直到定位到处理器中的指令缺陷。

4　基于指令生成约束的求解算法

为了高效生成高覆盖率的测试指令序列，本文提出基于指令生成约束的求解算法，实现测试指令

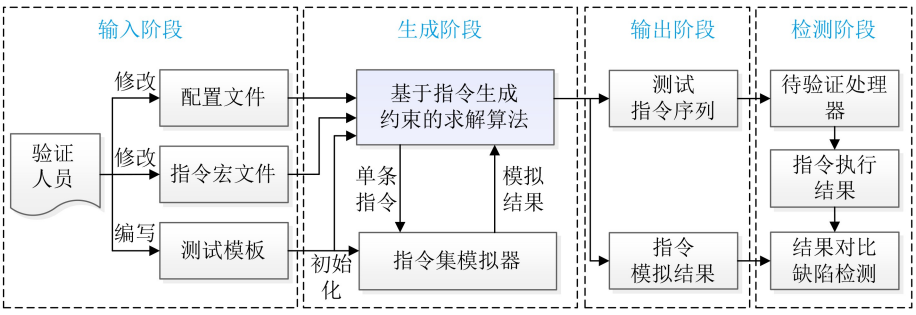


图 1 测试指令序列生成通用框架

序列自动生成。指令生成约束包括指令格式约束和功能覆盖约束，用于构造CSP模型的变量集合V、值域D和约束集合C。通过对指令验证需求进行分析，将功能覆盖约束分为通用覆盖约束和特殊覆盖约束，以增强指令生成约束的可复用性。通用覆盖约束是所有指令生成时的共性约束，特殊覆盖约束是对个别指令(如访存、分支指令等)生成时在遵守通用覆盖约束基础上，还需额外遵守的特殊约束。在提出的启发式搜索策略基础上进行指令约束求解，利用集成的指令集模拟器收集已覆盖指令生成约束的信息，反馈引导后续指令生成，实现测试指令序列的自动生成。所提出求解算法具有以下优势：第一，明确需要覆盖的指令生成约束，避免遗漏，实现全覆盖；第二，根据覆盖率信息反馈引导生成未覆盖指令，加快收敛速度；第三，定义通用和特殊功能覆盖约束可以复用到其余扩展指令约束定义中。

4.1 指令格式约束

指令格式约束用于保证对CSP模型求解生成的指令符合指令集架构规范，表1列出了RV32I的指令格式约束内容，用于构造单条指令或指令序列CSP模型。表1中的指令操作码、寄存器及立即数变量的数值约束用于构造变量集合V和值域D，不同访存粒度的约束用于构造寄存器变量和立即数变量之间的约束集合C。其中指令操作码值域仅考虑用户级指令，由于系统指令随机生成时会影响处理器正常状态，所以本文暂时不考虑RV32I中的系统指令。

4.2 功能覆盖约束

以RISC-V为例，功能覆盖约束用于构造CSP模型的约束集合C，定义如下：

(1)通用覆盖约束

(a)指令操作码覆盖约束，覆盖所有指令操作码以及指令操作码的依赖情况。约束表达式如式(1)所示：

$$\left.\begin{array}{l}\min(\mathrm{cnt}(\mathrm{op}_j))\geq 1, \mathrm{op}_j\in\{\mathrm{op}_1,\mathrm{op}_2,\cdots,\mathrm{op}_Q\}\\ \mathrm{Instr}_i.\mathrm{op}=\mathrm{Instr}_{i+1}.\mathrm{op}||\mathrm{Instr}_i.\mathrm{op}\neq\mathrm{Instr}_{i+1}.\mathrm{op}\end{array}\right\} \tag{1}$$

其中，Instr_i是指令序列中任意一条指令。$\min(\mathrm{cnt}(\mathrm{op_i}))$用于统计指令操作码数量，下标$Q$是指令操作码总数，通过统计最小值大于等于1表明是否覆盖所有指令操作码；Instr_i和Instr_{i+1}是相邻的指令，通过选择相邻指令操作码相同或相异，覆盖指令操作码的依赖结构。

表 1 RV32I指令格式约束

约束变量	数值约束
指令操作码	add, sub等37种已定义操作码
寄存器	X0 ～ X31，32个整数通用寄存器
立即数	根据不同比特数确定，如5-bit无符号数立即数范围0 ～ 31
访存指令	访存粒度为字节/半字/字，偏移地址约束为1/2/4的倍数

(b)寄存器变量覆盖约束。约束表达式如式(2)所示，其中 reg表示寄存器，下标M是数量，Read和Write表示寄存器读和写数量，下标N是指令数量，约束读写的最小值表明覆盖所有寄存器的读写。rd是目的寄存器，rs1, rs2分别是源寄存器1和2，$\mathrm{I}_1.\mathrm{r}_1$ sign $\mathrm{I}_2.\mathrm{r}_2$用于约束指令序列间寄存器的依赖结构，通过选择不同的I_1, I_2, r_1, r_2 覆盖单条指令内源寄存器和目的寄存器相同或相异，相邻指令或指令序列之间寄存器写后写、写后读、读后写、读后读4种寄存器依赖结构。

$$\left.\begin{array}{l}\min(\mathrm{Read}(\mathrm{reg_k}))\geq 1\ \&\&\ \min(\mathrm{Write}(\mathrm{reg_k}))\geq 1\\ \mathrm{I}_1.\mathrm{r}_1\ \mathrm{sign}\ \mathrm{I}_2.\mathrm{r}_2,\ \mathrm{sign}\in\{=,\neq\}\\ \mathrm{I}_1,\mathrm{I}_2\in\{\mathrm{Instr_i},\mathrm{Instr_{i+1}},\mathrm{Instr_j}\}\\ \mathrm{r}_1,\mathrm{r}_2\in\{\mathrm{rd},\mathrm{rs1},\mathrm{rs2}\}\\ \mathrm{reg_k},\mathrm{rd},\mathrm{rs1},\mathrm{rs2}\in\{\mathrm{reg}_1,\mathrm{reg}_2,\cdots,\mathrm{reg}_M\}\\ \mathrm{Instr}_i,\mathrm{Instr}_{i+1},\mathrm{Instr}_j\in\{\mathrm{Instr}_1,\mathrm{Instr}_2,\cdots,\mathrm{Instr}_N\}\\ j\neq i, j\neq i+1\end{array}\right\} \tag{2}$$

(c)立即数变量覆盖约束。立即数变量需要覆盖特殊数值，即最大值，最小值，–1, 0, 1，同时需要均匀覆盖取值空间。约束表达式如式(3)所示，其中，Imm.min 和 Imm.max分别是立即数变量的最小值和最大值，U(Imm.min, Imm.max)表明立即数在最小值和最大值之间服从均匀分布。

$$\begin{array}{c}\mathrm{Imm}\in\{\mathrm{Imm.max},\mathrm{Imm.min},0,1,-1\}\ \&\&\\ \mathrm{Imm}\sim\mathrm{U}(\mathrm{Imm.min},\mathrm{Imm.max})\end{array} \tag{3}$$

(2)特殊覆盖约束

(a)访存指令覆盖约束。访存指令不仅需要遵守通用覆盖约束，还需要覆盖对同一地址连续访存、连续地址进行访存、均匀访存存储空间的特殊约束。约束表达式如式(4)所示，其中，memAccess.addr是访存地址，下标i和i+1是当前访存指令及下一条访存指令的访存地址，min和max分别表示访存空间范围的最小值和最大值，与实际处理器设计相关，根据实际处理器的设计进行修改。S表示访存粒度，选择不同的S覆盖连续地址的访存。

$$\left.\begin{array}{l}\mathrm{memAccess.addr}_i=\mathrm{memAccess.addr}_{i+1}\ \&\&\\ \mathrm{memAccess.addr}_i+\mathrm{S},\mathrm{S}\in\{8,16,32\}\\ \mathrm{memAccess.addr}\sim\mathrm{U}(\mathrm{memAccess.addr.min},\\ \mathrm{memAccess.addr.max})\end{array}\right\} \tag{4}$$

(b)分支指令覆盖约束。分支指令需要覆盖分

支跳转的各种情况，同时应避免在指令序列中出现死循环的分支跳转。其约束表达式如式(5)所示，其中，Instr_i.pc表示当前指令的程序计数器(Program Counter, PC)地址。通过控制下一条指令的PC地址范围可以覆盖跳转的各种情况，同时保证下一条指令的PC地址与已生成指令的PC地址不同，以避免出现死循环。

$$\left.\begin{array}{l}\text{Instr}_{i+1}.\text{pc} > \text{Instr}_i.\text{pc} || \text{Instr}_{i+1}.\text{pc} < \text{Instr}_i.\text{pc} \\ \text{Instr}_{i+1}.\text{pc} \notin \{\text{Instr}_1.\text{pc}, \text{Instr}_2.\text{pc}, \cdots, \text{Instr}_i.\text{pc}\}\end{array}\right\} \tag{5}$$

(c)扩展指令或自定义指令。验证人员可以基于上述约束定义对扩展指令或自定义指令定义特殊覆盖约束。

4.3 启发式搜索策略

为了实现对上述约束进行全覆盖并加快收敛速度，本文提出启发式搜索策略，通过统计已生成指令的覆盖信息，引导后续指令生成，具体包括指令操作码搜索策略、寄存器搜索策略和立即数搜索策略。

根据式(6)定义的指令操作码搜索策略生成满足指令操作码覆盖约束定义的指令操作码。

$$\left.\begin{array}{l}\text{op}_i = \begin{cases} P_1 \times \text{RAND}(\text{UInstr}) \\ P_2 \times \text{op}_{i-1} \\ P_3 \times \text{RAND}(\text{WInstr} - \text{op}_{i-1}) \end{cases} \\ \displaystyle\sum_{i=1}^{i=3} P_i = 1 \end{array}\right\} \tag{6}$$

指令操作码搜索策略包括3种搜索路径。其中，RAND(UInstr)表示在未覆盖到的指令操作码集合UInstr中利用RAND随机函数随机生成指令操作码；op_i和op_{i-1}分别表示当前指令操作码和上一条指令操作码；WInstr 表示所有指令操作码集合，$\text{WInstr} - \text{op}_{i-1}$表示与上一条指令操作码相异的集合。$P_1$, P_2及P_3表示3种搜索路径的概率，满足概率之和等于1。生成第1条指令时，P_2及P_3的取值为零，仅P_1为1；生成第1条指令后且UInstr值域不为空时，P_1, P_2及P_3的取值相同，因为3种情况均需要被覆盖到，也可以根据具体验证需求调整概率取值；当UInstr为空时，P_1的取值为零，P_2及P_3的取值各为0.5。寄存器搜索策略和立即数搜索策略中的搜索路径概率的选择类似，可根据实际需求进行调整。

式(7)为满足寄存器变量覆盖约束定义下的寄存器搜索策略，包括4种搜索路径。其中，Ureg是未覆盖到的寄存器集合；op_i.reg是当前指令已覆盖寄存器集合；op_{i-1}.reg是上一条指令覆盖的寄存器集合；Hreg 是生成多条指令后已覆盖寄存器集合。与指令操作码搜索类似，在生成第1条指令的第1个寄存器变量时，只在Ureg搜索，即P_1为1；在生成第1条指令的第1个寄存器变量后，在Ureg和op_i.reg中进行搜索，即P_1及P_2各为0.5；在生成第1条指令后，在Ureg, op_i.reg和op_{i-1}.reg搜索，即P_1, P_2及P_3的取值相同；在生成多条指令后，P_1, P_2, P_3及P_4的取值为0.25。当任一集合为空时，相应搜索路径概率为零。

$$\text{reg} = \begin{cases} P_1 \times \text{RAND}(\text{Ureg}) \\ P_2 \times \text{RAND}(\text{op}_i.\text{reg}) \\ P_3 \times \text{RAND}(\text{op}_{i-1}.\text{reg}) \\ P_4 \times \text{RAND}(\text{Hreg}) \end{cases} \tag{7}$$

根据立即数变量覆盖约束和特殊覆盖约束定义立即数搜索策略，包括运算指令立即数、访存指令立即数和分支指令立即数搜索策略。根据3种不同指令类型的立即数验证要求，可以进一步定义搜索策略，即运算指令立即数搜索策略如式(8)所示，包括2种搜索策略，其中Uniform是均匀分布函数，Special为特殊数值集合，包括最大值，最小值，–1, 0, 1，也可根据验证需求添加其他数值。

$$\text{Imm} = \begin{cases} P_1 \times \text{Uniform}(\text{Imm.min}, \text{Imm.max}) \\ P_2 \times \text{RAND}(\text{Special}) \end{cases} \tag{8}$$

访存指令和分支指令的立即数搜索策略与式(8)类似，其余扩展指令或自定义指令的立即数搜索策略也可在式(8)的基础上进行扩展。

4.4 求解算法流程

为了能够自动生成满足验证需求的测试指令序列，提出了基于指令生成约束的求解算法，执行流程如图2所示。首先，根据指令生成需求构建指令序列CSP模型；然后，调用指令操作码搜索策略构建单条指令CSP模型，再调用寄存器和立即数搜索策略，完成单条指令的生成；之后，对生成的指令进行格式约束检查，若不符合则丢弃，若符合则送入指令集模拟器进行模拟，记录覆盖信息和模拟结果，用于引导后续指令生成；最后，根据指令生成数量判断是否终止算法，输出测试指令序列及模拟结果。算法伪代码如图3所示。

图4(a)为测试模板中自定义指令生成需求及数量的示例，Macro为指令宏标志，RV32I_Alu表示RV32I指令集中属于运算类型的一系列指令，Num表示需要生成的指令数量，图4(b)是利用本文所提出的求解算法按照测试模板要求生成的测试指令序列。

5 实验结果与分析

5.1 实验内容和评价指标

以RV32I为目标指令集，官方的RISC-V Tests[16]

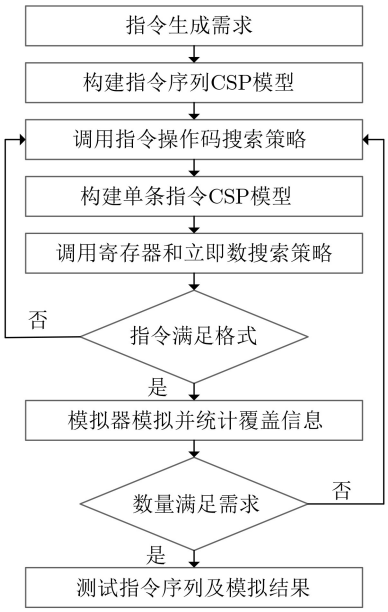


图 2 指令约束求解流程

```
输入: 测试模板指令生成需求及数量
1. build(CSP)
/* 构造指令序列CSP模型 */
2. for n = 1 to N do
    /* 循环生成所需数量的指令 */
      3. OP = OpGener()
    /* 指令操作码搜索策略 */
      4. build(OP.CSP)
      /* 构造单条指令CSP模型 */
      5. for i = 0 to OP.CSP.V do
          /* 逐个求解CSP模型变量 */
          6. if OP.CSP.V_i = Reg
          7. OP.CSP.D_i=RegGener()
            /* 寄存器搜索策略 */
          8. if OP.CSP.V_i = Imm
          9. OP.CSP.D_i = ImmGener()
            /* 立即数搜索策略 */
      10. end for
      11. Check()
      12. Simulate()
      13. updateHistory()
14. end for
输出: 测试指令序列
```

图 3 指令约束求解算法伪代码

```
Variable: reg
Register: X1
Reg_Value: 0xffff_ffff
Register: X2
Reg_Value: 0x0000_0001
Macro:
Mac_Name ←— RV32I_Alu
Num ←— 1000
```

(a) 测试模板示例

```
Register Initial Value:
X0 = 0x0, X1 = 0xffffffff, ……
Instructions:
1: add X3,X1,X2
2: sub X3,X3,X5
……
999: addi X14,X17,0x0
1000: or X8,X14,X14
```

(b) 生成的测试指令序列示例

图 4 测试模板及生成的测试指令序列示例

测试集作为基准，选择维持弧相容算法[17]、Herdt等人[7]提出的基于约束规范随机生成方式与本文提出的基于指令生成约束的求解算法进行测试指令序列生成的比较。采用Herdt等人提出的结构覆盖率和数值覆盖率进行评估[18]，结构覆盖率是指令操作码和寄存器变量的依赖结构，此外还需要增加目的寄存器等于和不等于X0寄存器的两种情况，对RV32I而言共174种情况。数值覆盖率是指指令操作码、寄存器变量和立即数变量的覆盖情况，立即数仅覆盖最大值，最小值，1, 0, –1，对RV32I而言共2684种情况。用已覆盖情况与所有覆盖情况之比评估生成的测试指令序列质量。

5.2 实验分析

5.2.1 指令覆盖率数量统计

图5给出了RISC-V Tests官方测试集、维持弧相容算法、基于约束规范随机生成方式和基于指令生成约束的求解算法在生成相同数量的测试指令序列下，结构覆盖率(a)和数值覆盖率(b)的比较。官方测试集由加州伯克利定向编写，指令内容固定，所以结构和数值覆盖率不随数量的增加而增大。维持弧相容算法对变量随机赋值缺乏相应的约束条件，虽然可以获得较高的覆盖率，但是无法保证对所有情况进行覆盖。基于约束规范随机生成方式，预定义各种约束情况进行遍历求解，覆盖率随数量的增加而增大，可以实现对所有情况的覆盖，但是该方法由于在对每种情况求解中会得到所有满足约束的指令，即某些覆盖点存在冗余生成指令。实验结果表明，基于指令生成约束的求解算法相较于其他生成方法，在生成相同指令数量下可以得到更高的覆盖率。在生成30 000条指令时，基于指令生成约束的求解算法相较于官方测试集结构覆盖率提升了13.64%，数值覆盖率提升了85.06%；相较于维持弧相容算法结构覆盖率提升了8.45%，数值覆盖率提升了6.90%；相较于约束规范随机生成方式结构覆盖率一致，数值覆盖率多覆盖到了JALR指令中RS1＝X0的情况，这是因为指令约束求解算法对生成指令实时模拟，可以实时获取PC地址，通过反馈控制PC地址可以覆盖JALR指令中RS1＝X0的情况。此外，基于约束规范随机生成和基于指令生成约束的求解算法在数值覆盖率统计时，未覆盖到

的情况是访存指令LW, LH, LB, LBU, LHU, SB, SH, SW的基址寄存器等于X0的情况，这是因为X0寄存器恒为零值。

5.2.2　指令覆盖率收敛时间

图6给出了维持弧相容算法、基于约束规范随机生成方式以及基于指令生成约束的求解算法在固定时间内的结构覆盖率(a)和数值覆盖率(b)的收敛时间统计。

维持弧相容算法缺乏相应约束无法保证在有限时间内对所有情况进行覆盖。基于约束规范随机生成方式必须对预定义的约束情况进行遍历求解，缺少覆盖率统计所以无法快速收敛。实验结果表明，基于指令生成约束的求解算法的收敛时间明显优于其他两种算法，相较于基于约束规范随机生成方式结构覆盖率收敛时间减少85.62%，数值覆盖率收敛时间减少57.64%，加快收敛速度，提高测试指令序列生成的效率。

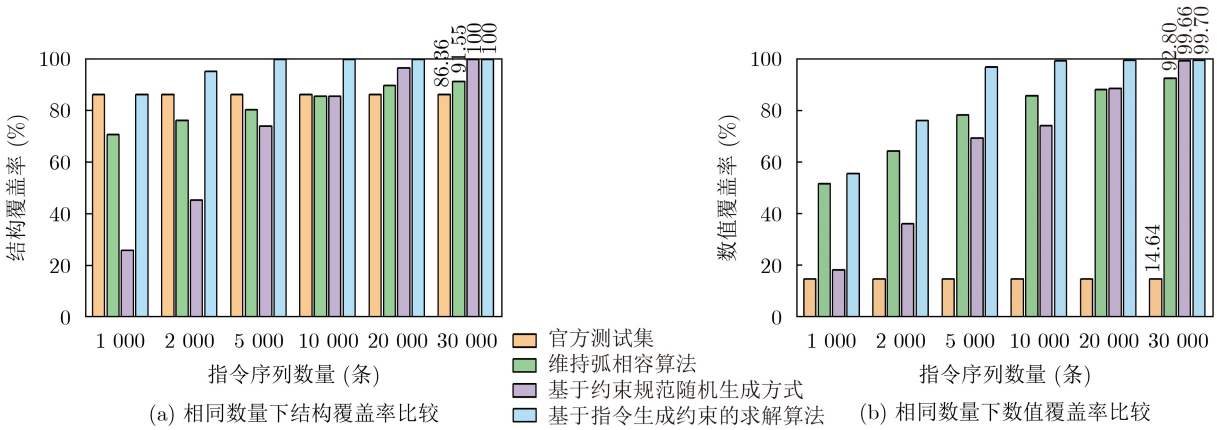
5.2.3　指令类型和寄存器数量分布

图7进一步给出了基于指令生成约束的求解算法生成30 000条指令时指令类型数量分布(a)和寄存器数量分布(b)。实验结果表明，利用该算法可以均匀覆盖RV32I中所有的37条指令类型(图7(a)中横坐标仅列部分类型名称)，并可以均匀访问X0～X31所有寄存器。

6　结论

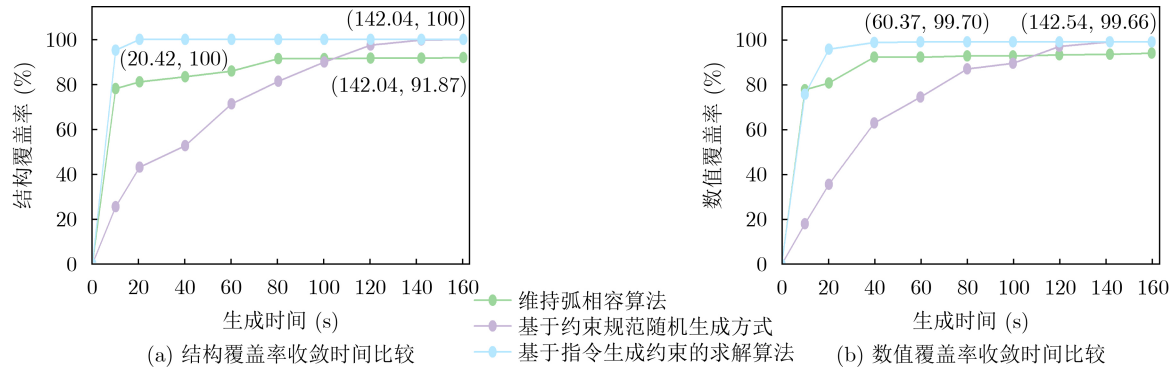
为了避免处理器受到指令缺陷的威胁，本文以RISC-V指令为例，提出基于指令生成约束的测试序列生成方法，构建测试指令序列生成框架，实现对RISC-V处理器的指令缺陷检测，本文方法也适用于其他指令集架构的处理器。首先，结合所需的指令集架构，定义指令生成约束，区分通用和特殊功能覆盖约束，可以增强约束定义的可复用性；然后，构造启发式搜索策略，完成对指令生成约束的求解，以实现测试指令序列的自动生成。实验结果表明，该方法在满足所有覆盖要求的前提下，可以降低覆盖率的收敛速度，实现RISC-V测试指令序列的高效生成。

基于所提出的测试指令序列生成框架，在Linux环境下开发了一款指令生成工具GTIR(Generate Test Instructions for RISC-V)，利用开发的GTIR工具对第三方引入指令缺陷后的开源RISC-V处理器SweRV-EH2[19]进行检测。实验表明，该工具可以定位到7条指令(SLT, XORI, LW, SW,



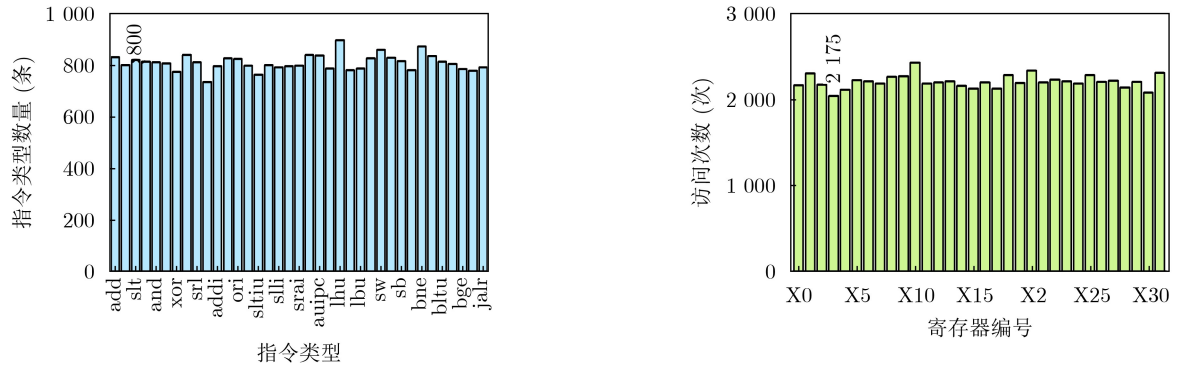
(a) 相同数量下结构覆盖率比较　　(b) 相同数量下数值覆盖率比较

图 5 相同数量下结构和数值覆盖率比较



(a) 结构覆盖率收敛时间比较　　(b) 数值覆盖率收敛时间比较

图 6 结构和数值覆盖率收敛时间比较



(a) 指令类型数量分布　　(b) 寄存器数量分布

图 7 指令类型和寄存器生成分布

BGE, AUIPC, JAL)在译码、执行阶段由第三方引入的指令缺陷，可以帮助设计人员对处理器硬件设计缺陷进行检测。后续研究结合处理器微结构信息构造指令生成约束，考虑对处理器微结构进行缺陷检测。

参 考 文 献

[1] WANG Yuze, LIU Peng, WANG Weidong, *et al*. On a consistency testing model and strategy for revealing RISC processor's dark instructions and vulnerabilities[J]. *IEEE Transactions on Computers*, 2022, 71(7): 1586–1597. doi: 10.1109/TC.2021.3097174.

[2] HUANG Yu, AHMAD H, FORREST S, *et al*. Applying automated program repair to dataflow programming languages[C]. 2021 IEEE/ACM International Workshop on Genetic Improvement (GI), Madrid, Spain, 2021: 21–22. doi: 10.1109/GI52543.2021.00013.

[3] PATTERSON D A and HENNESSY J L. Computer Organization and Design RISC-V Edition: The Hardware Software Interface[M]. China Machine Press, 2019: v–vii.

[4] KAMKIN A S and KOTSYNYAK A M. Specification-based test program generation for MIPS64 memory management units[J]. *Proceedings of the Institute for System Programming of the RAS*, 2016, 28(4): 99–114. doi: 10.15514/ISPRAS-2016-28(4)-6.

[5] AHMADI-POUR S, HERDT V, and DRECHSLER R. Constrained random verification for RISC-V: Overview, evaluation and discussion[C/OL]. MBMV 2021; 24th Workshop, 2021: 1–8.

[6] BRUNS N, HERDT V, JENTZSCH E, *et al*. Cross-level processor verification via endless randomized instruction stream generation with coverage-guided aging[C]. 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 2022: 1123–1126. doi: 10.23919/DATE54114.2022.9774771.

[7] HERDT V, GROßE D, and DRECHSLER R. Towards specification and testing of RISC-V ISA compliance[C]. 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2020: 995–998. doi: 10.23919/DATE48585.2020.9116193.

[8] CHUPILKO M, KAMKIN A, and PROTSENKO A. Open-source validation suite for RISC-V[C]. 2019 20th International Workshop on Microprocessor/SoC Test, Security and Verification (MTV), Austin, USA, 2019: 7–12. doi: 10.1109/MTV48867.2019.00010.

[9] 谭坚, 罗巧玲, 王丽一, 等. 基于SMT求解器的微处理器指令验证数据约束生成技术[J]. 计算机研究与发展, 2020, 57(12): 2694–2702. doi: 10.7544/issn1000-1239.2020.20190718.
TAN Jian, LUO Qiaoling, WANG Liyi, *et al*. Data constraint generation technology for microprocessor instruction verification based on SMT solver[J]. *Journal of Computer Research and Development*, 2020, 57(12): 2694–2702. doi: 10.7544/issn1000-1239.2020.20190718.

[10] ADIR A, ALMOG E, FOURNIER L, *et al*. Genesys-Pro: Innovations in test program generation for functional processor verification[J]. *IEEE Design & Test of Computers*, 2004, 21(2): 84–93. doi: 10.1109/MDT.2004.1277900.

[11] BRAUN M, FINE S, and ZIV A. Enhancing the efficiency of Bayesian network based coverage directed test generation[C]. Proceedings Ninth IEEE International High-Level Design Validation and Test Workshop, Sonoma, USA, 2004: 75–80. doi: 10.1109/HLDVT.2004.1431241.

[12] PFEIFER N, ZIMPEL B V, ANDRADE G A G, *et al*. A reinforcement learning approach to directed test generation for shared memory verification[C]. 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2020: 538–543. doi: 10.23919/DATE48585.2020.9116198.

[13] HERDT V, GROßE D, and DRECHSLER R. Closing the RISC-V compliance gap: Looking from the negative testing side[C]. 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, USA, 2020: 1–6. doi: 10.1109/DAC18072.2020.9218629.

[14] WATERMAN A, LEE Y, PATTERSON D A, *et al*. The RISC-V instruction set manual. Volume I: User-level ISA, version 2.0[R]. Technical Report No. UCB/EECS-2014-54, 2014: 9–26.

[15] AYADI Z, BOULILA W, FARAH I R, *et al*. Resolution methods for constraint satisfaction problem in remote sensing field: A survey of static and dynamic algorithms[J]. *Ecological Informatics*, 2022, 69: 101607. doi: 10.1016/j.ecoinf.2022.101607.

[16] UC Berkeley Architecture Research, RISC-V ISA Tests[EB/OL]. https://github.com/riscv/riscv-tests, 2022.

[17] BIN E, EMEK R, SHUREK G, *et al*. Using a constraint satisfaction formulation and solution techniques for random test program generation[J]. *IBM Systems Journal*, 2002, 41(3): 386–402. doi: 10.1147/sj.413.0386.

[18] HERDT V, GROßE D, LE H M, *et al*. Verifying instruction set simulators using coverage-guided fuzzing[C]. 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 2019: 360–365. doi: 10.23919/DATE.2019.8714912.

[19] XI Yuhao, ZHANG Zhendong, WANG Yuze, *et al*. A heterogeneous RISC-V SoC for confidential computing and password recovery[C]. 2022 7th International Conference on Integrated Circuits and Microsystems (ICICM), Xi'an, China, 2022: 500–504. doi: 10.1109/ICICM56102.2022.10011247.

刘　鹏：男，博士，教授，研究方向为计算机体系结构、并行计算机结构、超大规模集成电路设计、硬件安全.

胡文超：男，硕士生，研究方向为集成电路设计、处理器验证.

刘德启：男，硕士，研究方向为超大规模集成电路设计、人工智能、高性能计算.

韩晓霞：女，博士，讲师，研究方向为微电子学、固体电子学、集成电路设计.

刘扬帆：男，博士，研究方向为计算机体系结构、高性能多核处理器设计.

责任编辑：陈　倩