软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(12):3992−4024 [doi: 10.13328/j.cnki.jos.006490]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

RISC-V 指令集架构研究综述*

刘 畅[1,2], 武延军[1,3], 吴敬征[1,3], 赵 琛[1,3]

[1](中国科学院 软件研究所 智能软件研究中心,北京 100190)

[2](中国科学院大学,北京 100190)

[3](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

通讯作者: 武延军, E-mail: yanjun@iscas.ac.cn

摘 要: 指令集作为软硬件之间的接口规范,是信息技术生态的起始原点.RISC-V 是计算机体系结构走向开放的必然产物,其出现为系统研究领域带来了新的思路,即系统软件问题的研究深度可以进一步向下延伸至指令集架构,从而拓展甚至颠覆软件领域的"全栈"概念.对近年来 RISC-V 指令集架构相关的研究成果进行了综述.首先介绍了 RISC-V 指令集的发展现状,指出开展 RISC-V 研究应重点关注的指令集范围.然后分析了 RISC-V 处理器设计要点和适用范围.同时,围绕 RISC-V 系统设计问题,从指令集、功能实现、性能提升、安全策略这 4 个方面,论述了 RISC-V 处理器基本的研究思路,并分析了近年来的研究成果.最后借助具体的研究案例,阐述了 RISC-V 在领域应用的价值,并展望了 RISC-V 架构后续研究的可能切入点和未来发展方向.

关键词: RISC-V;架构设计;处理器;性能优化;系统安全

中图法分类号: TP316

中文引用格式: 刘畅,武延军,吴敬征,赵琛.RISC-V 指令集架构研究综述.软件学报,2021,32(12):3992−4024. http://www.jos.org.cn/1000-9825/6490.htm

英文引用格式: Liu C, Wu YJ, Wu JZ, Zhao C. Survey on RISC-V system architecture research (in Chinese). 2021,32(12): 3992−4024. http://www.jos.org.cn/1000-9825/6490.htm

Survey on RISC-V System Architecture Research

LIU Chang[1,2], WU Yan-Jun[1,3], WU Jing-Zheng[1,3], ZHAO Chen[1,3]

[1](Intelligent Software Research Center, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100190, China)

[3](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

Abstract: ISA (instruction set architecture) is the interface specification between software and hardware, which is also the origin point of an information technology ecosystem. RISC-V is the inevitable product of computer architecture gradually moving towards openness. It brings a new paradigm for system research, i.e. software research issues can be tracked down to ISA, which expands or even subverts the traditional full-stack design theory on the system function, performance, security and other issues, showing a promising development prospect. This paper reviews the research results of RISC-V architecture in recent years. Firstly, the development status of RISC-V instruction set is introduced, and the scope of instruction set that should be paid attention to in RISC-V research is pointed out. Then, the current RISC-V CPU platforms, particularly RISC-V processors are analyzed, and the design points and application scope are summarized. Then, focusing on the design of RISC-V CPU, this paper discusses four fundamental research topics: instruction set, function implementation, performance improvement and security strategy, and reviews some research results in recent years. Finally, with the help of some specific cases, this paper expounds the role of risc-v in specific domains, analyzes the possible future directions of RISC-V research.

* 基金项目: 中国科学院先导专项(XDC05040000)
Foundation item: Strategic Priority Research Program of CAS (XDC05040000)
收稿时间: 2021-06-28; 修改时间: 2021-08-22, 2021-09-28; 采用时间: 2021-11-01

Key words: RISC-V; architecture design; processor; performance improvement; system security

1 前 言

现代信息系统的设计正在出现几个新的趋势:(1) 注重软硬件协同的设计模式.例如,英伟达公司推出CUDA 通用并行计算架构[1],发挥 GPU 的计算能力提升系统性能;Costan 等人[2]提出了硬件扩展 Sanctum,协助软件系统实现强隔离功能.(2) 面向需求的系统精准定制化.例如,近年来出现的AI加速芯片,通过芯片架构的定制,满足神经网络推理或训练的算力需求[3].在这样的趋势下,探究系统软硬件的多种组合方式,设计出最合适应用场景的系统架构,具有较高的理论价值和实践意义.与此同时,为了适应计算环境与应用环境的不断变化,计算机软硬件的发展也存在一种逐渐开放的趋势.以软件为例,从早期代码高度集中、与机器高度耦合,到 1964 年 Multics 操作系统[4]推动系统的通用化进程,UNIX 操作系统[5]开放用户任务、开启内核时代,再到 Mach 微内核的问世[6]、GNU 项目的发起以及 Linux 内核的公开发布,开源软件思想深入人心,体系结构也逐渐走向设计公开化、内容多样化、管理多元化、研发社区化、功能可定制化的局面.在这样的背景下,如何有效整合多种软硬件资源,发挥开放系统的独特优势,精准满足用户任务需求,成为关注重点.指令集架构(instruction set architecture,简称 ISA)作为软件与硬件之间的交互规范,定义了软硬件的组合或整合方式.在上述趋势和背景下,迫切需要一种新的指令集,既满足软硬件的深度协同融合和灵活组合,又具备开放特性.

RISC-V 是一种新兴的开源精简指令集架构,由加州大学伯克利分校在 2010 年首次发布[7].RISC-V 的出现和迅速发展有其必然的原因,它是建立在现有的体系结构(如 x86、ARM、MIPS 等)经长期发展所暴露出的种种问题之上,顺应现代信息系统设计需求和体系结构发展趋势而生的:(1) 现有体系结构往往缺乏开放性,存在许多知识产权、政治干预等非技术性问题.例如,Intel 公司持有 x86 架构的专利(1978 年开始),使用 x86 指令集相关技术需要向其支付高昂的授权费用,对 x86 指令集的模拟也将引发法律上的争议[8].这种封闭的态势与体系结构发展的开放趋势背道而驰,抬高了系统研发与成果转化的成本,阻碍了技术的推广和进步.RISC-V 具有开源、免费、开放、自由的特点,其基金会总部于 2020 年 3 月正式迁往永久中立国瑞士[9],更是释放了坚持服务全世界的信号,使任何组织和个人都可以不受地缘政治影响、自由平等地使用 RISC-V.(2) 现有体系结构经过长期发展,多个版本的迭代,积累了许多历史遗留问题.基于各历史版本的技术产品在市场生态中共存,使得新版本的研发必须考虑向后兼容性,去支持一些过时的定义和其实不需要的技术特性.例如,AMD64 是对 32 位 x86 架构的 64 位扩展,面向 64 位开发与应用环境;但它同时仍要向后兼容 32 位甚至 16 位的 x86 架构,使早期 x86 架构下开发的应用同样可以在 AMD64 系统中正常运行.这种积重难返的状态削弱了现有体系结构的可定制化能力,难以满足现代信息系统对于多样化的工作环境与功能表现的需求.RISC-V 作为一种从零开始设计的新体系结构,吸收了现有各体系结构优点的同时,去除了对历史遗留问题的顾及和旧有技术的依赖;进一步地,RISC-V 采用模块化设计,并提供大量自定义编码空间以支持对指令集的扩展,从而允许开发者根据资源、能耗、权限、实时性等不同需求,基于部分特定的模块和扩展指令集进行精细化的系统设计研发,体现了强大的系统可定制化能力.(3)现有主流架构的文档资源种类繁多、内容冗长,学习与维护的成本较高,使开发者难以在短时间内掌握所需的技术,遇到问题时也不易迅速定位到相关的信息区间.例如,ARMv8-A 架构的官方手册[10]仅一卷就多达 8 538 页;相比之下,RISC-V 官方手册仅有两卷 329 页,包括 238 页的指令集手册[11]和 91 页的特权架构手册[12],文档精简,学习门槛更低,更有助于研发团队的不断壮大和技术实力的不断进步.

因此,对于 RISC-V 的研究已成为近年来学术界和工业界的一大热点,涌现了许多突破性成果,如西部数据公司研发的基于 RISC-V 的通用架构 SweRV[13]、阿里巴巴公司研发的 64 位高性能嵌入式 RISC-V 处理器 Xuantie-910[14]、Koch 等人设计的嵌入式开源 FPGA(field programmable gate array,现场可编程门阵列)框架 FABulous[15]、中国科学院计算技术研究所在 RISC-V 中国峰会发布的开源高性能 RISC-V 处理器核“香山”[16]、上海交通大学开源的基于 RISC-V 的可信执行环境安全系统“蓬莱”[17].其中也包括了对 RISC-V 与不同应用领域结合方式的探索,如 Kadomoto 等人[18]利用 RISC-V 芯片改善了无线总线接口技术,以促进对于小型机器人的研究; Di Tucci 等人[19]将 RISC-V 应用于基因组处理,提出了专用领域架构 SALSA.这些系统、工具都是利用了

RISC-V 在资源依赖性、低功耗性、易用性、可定制性、可扩展性等方面的优势,因而能够高效、迅速而低成本地完成各自领域中的系统级任务.

RISC-V 作为一种指令集架构,一方面,它规定了硬件设备在设计电路、组装元件时应当实现的功能目标;根据指令集的内容,决定运算单元、存储单元等元件的种类、数目、位宽及接线方式.另一方面,它是对硬件能力的一种抽象,提供了机器所能完成的操作种类、地址空间大小、数据格式、访问权限信息;上层软件应用可以将指令集视为硬件运行环境,而无需特别关注具体的硬件实体.图 1 直观地解释了 RISC-V 在系统中的定位.

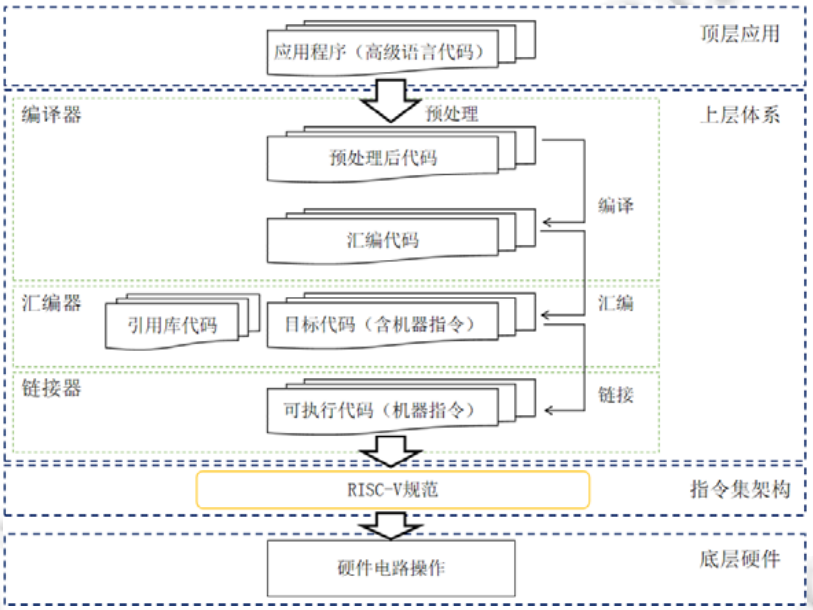


Fig.1 The translation and execution process of application code

图 1 系统对应用程序代码的翻译和执行过程

对于 RISC-V 的研究将至少着眼于 4 个层面中的一个:对于底层 RISC-V 硬件的研究、对于 RISC-V 指令集自身的研究、对于上层 RISC-V 系统(主要是基础软件)的研究,对于顶层应用的研究.而在研究内容上,普遍集中在系统或应用的功能、性能、安全这 3 个方面.本文围绕 RISC-V 体系结构设计过程和主要关注点,从上述 4 个层面对相关研究进行了总结.这些研究工作均来源于对 DBLP 文献数据库的检索,共涉及 126 篇论文文献(截至 2021 年 7 月).从论文发表的期刊和会议看,有 65 篇来自 CCF-A 类期刊或会议中,占比超过 50%;有 104 篇发表在计算机体系结构领域的期刊或会议中,占比超过 82%,其余 22 篇论文分别发表在网络与信息安全、软件工程、系统软件等领域的期刊或会议中.对这些 RISC-V 研究工作的分布统计见表 1.

本文第 1 节介绍 RISC-V 相关概念及其优势,并对本文结构进行概述.第 2 节回顾近年来对于 RISC-V 指令集的研究,包括对于各指令集扩展现状的分析.第 3 节总结处理器、加速器等支持 RISC-V 系统的硬件工作环境.第 4 节讨论各种 RISC-V 系统的设计及其要点.第 5 节介绍一些利用 RISC-V 实现性能、安全提升的应用案例.第 6 节展望 RISC-V 架构未来可能的发展方向和研究点.最后在第 7 节对本文内容进行总结.

2 RISC-V 指令集

RISC-V 的指令集包括基础指令集和扩展指令集两类.RISC-V 指令集架构被定义为一个基础指令集和若干可选扩展指令集的组合,并在一种权限模式下进行工作.本节对 RISC-V 各指令集的当前状态进行了总结和分析,并简要介绍了 RISC-V 的权限规则.

Table 1 Source distribution of major research results

表 1 主要研究成果来源分布

期刊/会议名称	CCF 级别	参考文献数目
Design, Automation & Test in Europe (DATE)	B	23
Design Automation Conf. (DAC)	A	13
Int'l Symp. on Computer Architecture (ISCA)	A	12
Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)	A	7
IEEE Trans. on Computer-aided Design of Integrated Circuits and System (TCAD)	A	6
Int'l Conf. on Cryptographic Hardware and Embedded Systems (CHES)	B	6
IEEE/ACM Int'l Symp. on MICROarchitecture (MICRO)	A	5
ACM/SIGDA Int'l Symp. on Field-programmable Gate Arrays (FPGA)	B	5
USENIX Security Symp.	A	5
IEEE Trans. on Computers (TC)	A	4
IEEE Trans. on Very Large Scale Integration Systems (TVLSI)	B	4
Int'l Conf. on Computer Design (ICCD)	B	4
Int'l Conf. on Parallel Processing (ICPP)	B	4
ACM Conf. on Computer and Communications Security (CCS)	A	3
ACM Symp. on Operating Systems Principles (SOSP)	A	3
ACM Trans. on Architecture and Code Optimization (TACO)	B	2
ACM Trans. on REconfigurable Technology and Systems (TRETS)	B	2
Int'l Test Conf. (ITC)	B	2
ACM SIGPLAN Conf. on Programming Language Design & Implementation (PLDI)	B	2
IEEE Trans. on Parallel and Distributed Systems (TPDS)	A	1
ACM Trans. On Design Automation of Electronic Systems (TODAES)	A	1
ACM Trans. on Embedded Computing Systems (TECS)	B	1
Journal of Systems Architecture: Embedded Software Design (JSA)	B	1
High Performance Computer Architecture (HPCA)	A	1
Int'l Conf. for High Performance Computing, Networking, Storage, and Analysis (SC)	A	1
Code Generation and Optimization (CGO)	B	1
European Conf. on Computer Systems (EuroSys)	B	1
Int'l Conf. on High Performance and Embedded Architectures and Compilers (HiPEAC)	B	1
Int'l Parallel & Distributed Processing Symp. (IPDPS)	B	1
Int'l Symp. on High Performance Distributed Computing (HPDC)	B	1
ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)	A	1
USENIX Symp. on Operating Systems Design and Implementations (OSDI)	A	1
IEEE Symp. on Security and Privacy (S&P)	A	1

2.1 RISC-V基础指令集

RISC-V 的基础指令集包含了能够为编译器、汇编器、链接器、操作系统(结合额外的特权操作)等提供必要功能实现的最小指令集合.它们是构筑 ISA 和软件工具链的骨架,可以围绕它们来构建更多定制的处理器 ISA.根据最新的 RISC-V 规范[11],RISC-V 共有 5 种基础指令集:RVWMO、RV32I、RV64I、RV32E、RV128I,分别代表了弱内存次序指令集、32 位整数指令集、64 位整数指令集、32 位嵌入式整数指令集、128 位整数指令集.其中,RV32I 和 RV64I 是最主要的两种,分别针对 32 位和 64 位工作环境而设计;RV32E 是为嵌入式环境设计的一个 RV32I 的简化版本,RV128I 将用于未来的 128 位环境,而 RVWMO 描述了 RISC-V 所使用的内存一致性模型.任何一种 RISC-V 指令集架构都必须完整地实现一种基础指令集.

2.1.1 RV32I 和 RV64I 指令集

RV32I 指令集和 RV64I 指令集都使用 32 个通用寄存器(x 寄存器)和一个额外的非特权寄存器(pc 寄存器),如图 2 所示[11].区别在于寄存器的位宽(XLEN),RV32I 中是 32 位(XLEN=32),而 RV64I 中是 64 位(XLEN=64).x 寄存器又称为"整数寄存器",可以存放 XLEN 长度的数值,这些数值可以被各指令解释为布尔值的集合、二进制有符号整数,或者二进制无符号整数的二补码.其中,x0 寄存器的所有位都被硬布线为 0 值,不可更改,因此也被称作零寄存器(zero).pc 寄存器又称"程序计数器",用于保存当前指令的地址.

通用寄存器 x1~x31 理论上可以用于各种用途,但是根据标准调用约定[20]的建议,这些寄存器的功能在实践中也是相对固定的:x1~x4 用于保存程序运行相关的指针,共 4 个;x5~x7 及 x28~x31 用作临时寄存器,共 7 个;x8~x9 及 x18~x27 由被调用者使用,共 12 个;x10~x17 用于保存调用的参数,共 8 个.表 2 列出了所有 x 寄存器在标准调用约定下的含义.

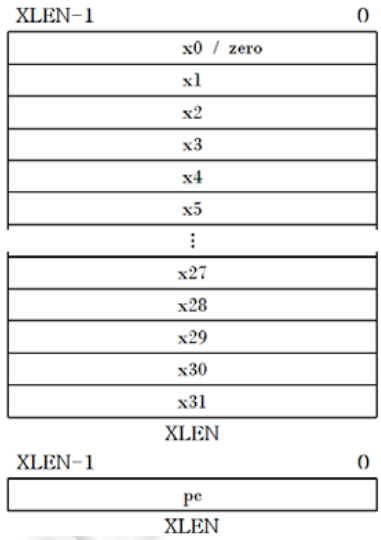


Fig.2 Register status of RV32I/RV64I[11]

图 2 RISC-V 整数指令集(RV32I/RV64I)的寄存器状态[11]

Table 2 The meaning of each x register in standard calling convention

表 2 标准调用约定下的各 x 寄存器含义

寄存器名	ABI 助记符	含义	是否跨调用保留
x0	zero	0	–(不可变)
x1	ra	返回地址	否
x2	sp	栈指针	是
x3	gp	全局指针	–(不允许)
x4	tp	线程指针	–(不允许)
x5~x7	t0~t2	临时寄存器	否
x8~x9	s0~s1	被调用者保存	是
x10~x17	a0~a7	参数寄存器	否
x18~x27	s2~s11	被调用者保存	是
x28~x31	t3~t6	临时寄存器	否

RV32I 有 4 种基础指令格式:R/I/S/U,指令集中的任何指令都可以根据操作数的数量、种类、规模以及自身的功能需求,选用其中一种格式.所有这些指令格式都是 32 位固定长度,并且必须在内存中对齐到 4 字节的边界.图 3 显示了这 4 种基础的指令格式.

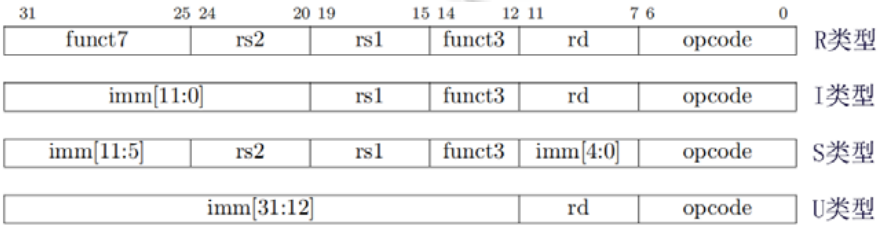


Fig.3 Four basic instruction formats of RV32I[11]

图 3 RV32I 的 4 种基础指令格式[11]

RV64I 采用与 RV32I 相同的指令格式,只是将整数寄存器和所支持的用户地址空间扩展到了 64 位,增加了一些操作低 32 位的“*W”指令.例如,RV32I 中的 ADDI 指令可以将符号扩展的 12 位立即数加到寄存器中,并将

计算结果的低 XLEN 位(即低 32 位)作为指令的执行结果,忽略计算溢出.RV64I 沿用了该指令,但寄存器和执行结果都被扩展到 64 位.同时 RV64I 新增了 ADDIW 指令,用以产生 32 位的计算结果,再将其符号扩展至 64 位并忽略计算溢出.

2.1.2 RV32E 指令集

RV32E 是专为嵌入式环境设计的指令集,是对 RV32I 指令集的一种简化.根据 RISC-V 规范[11]的描述,它与 RV32I 唯一的区别在于,将可用的整数寄存器的数目从 32 减少到 16,即只使用 x0~x15 和 pc 完成所有的指令功能,而去除了 RV32I 中的 x16~x31 寄存器.RV32E 的寄存器状态如图 4 所示.

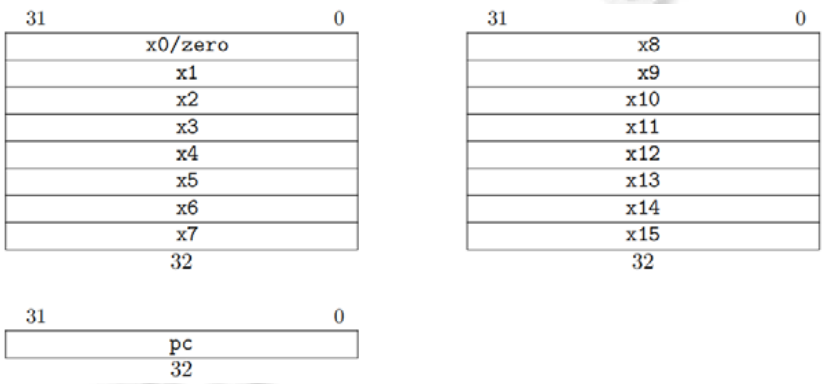


Fig.4 Register status of RV32E[11]

图 4 RV32E 的寄存器状态[11]

由于寄存器数目不同,适用于 RV32I 的标准调用约定与 RV32E 并不兼容.RV32E 需要与之不同的调用约定和 ABI(application binary interface,应用程序二进制接口),但目前还没有这方面较为统一而稳定的标准.作为临时的解决方案,RISC-V 规范提供了一种专用于 RV32E 的调用约定 ILP32E,它要求栈指针对齐只能到 32 位边界;而在寄存器的分配上,采用与 RV32I 一致的分配方案.例如,在高性能开源 RISC-V 微控制器核心 SCR1[21]中,对于 RV32E 的实现便采取了这样的处理方式.通过预处理选项的控制,使寄存器 x0~x15 完全复用,不特别区分 RV32I 或 RV32E,而 x16~x31 只有在 RV32I 指令集中启用.

2.1.3 RV128I 指令集

RV128I 是一个用于支持 128 位地址空间的指令集.根据 RISC-V 规范中的预测,随着计算需求的增长和技术的不断演进,很可能会在 2030 年之前就会需要超过 64 位的地址空间[11].因此,事先提出面向未来的 RV128I 指令集便成为一项积极的应对措施.

RV128I 是对 RV32I 和 RV64I 的直接外扩,仅仅是把整数寄存器宽度扩展到了 128 位(XLEN=128).由于大部分整数运算指令在 XLEN 位上定义,所以无需变化.此外,RV128I 保留了 RV64I 中用于操作低 32 位的“*W”指令,只是把结果从 32 位符号扩展到 128 位;RV128I 还新增了用于操作低 64 位的“*D”指令.

2.1.4 RVWMO 指令集

RVWMO 指令集定义了 RISC-V 的内存一致性模型.内存一致性模型是一组规则的集合,指定了内存的 load 操作所能返回的结果.RVWMO 内存模型属于弱内存次序的,即在一个多线程程序有许多种不同可能的执行,而每种执行有其自己相应的全局内存次序.这里的“全局内存次序”是指所有硬件线程所产生的内存操作的总体次序.

RVWMO 指令集为加载(load)和存储(store)等内存操作提供了一组内存模型原语,用于对相关指令进行标注.表 3 列举了这些标注的内容及其含义.

RVWMO 内存模型的定义基于依存句法(syntactic dependency)的概念,即,指令的源寄存器、目的寄存器,和指令从源寄存器到目的寄存器含有依赖的方式.对于不同的 RISC-V 指令集中的不同指令,会有各自不同

的依存句法.

RVWMO 对程序的执行提出了次序上的要求,即,全局内存次序需要遵循着各个硬件线程的内存次序中的一部分(但不是所有).这部分必须被遵循的次序子集被称为“保留的程序次序”.

内存模型原语、各指令的依存句法和保留的程序次序,以及基于这些内容的 3 条公理[11](加载值公理、原子性公理、进度公理),共同构成了 RVWMO 内存一致性模型.

Table 3 Annotations of memory access instructions in RVWMO

表 3 RVWMO 指令集对内存操作指令的标注

	与处理器一致	顺序一致
加载	acquire-RCpc	acquire-RCsc
释放	release-RCpc	release-RCsc

2.2 RISC-V扩展指令集

RISC-V 的扩展指令集用于为 ISA 提供特定方面的功能操作指令.一个 ISA 可以选择加入多个扩展指令集.而为了使多个指令集能够共存,各指令集的编码空间均按照 RISC-V 国际基金会(RVI)的编码要求进行了划分,以避免冲突.现有的 RISC-V 扩展指令集主要有以下 24 种.

(1) 乘法和除法扩展(M 扩展):用于将两个整数寄存器中的值进行相乘或相除.具体包含了 MUL、DIV、REM 等指令及对应于无符号数操作或 64 位操作相关的各种变体指令.

(2) 原子指令扩展(A 扩展):用于支持相同内存空间中的多个硬件线程间的同步.主要包含了对内存进行原子性读取、修改、写入的指令.

(3) 单精度浮点扩展(F 扩展):用于支持单精度浮点运算.添加了兼容 IEEE 754-2008 算术标准[22]的单精度浮点运算指令,以及 32 个 32 位宽的浮点寄存器 f0~f31,和一个浮点控制与状态寄存器 fcsr.

(4) 双精度浮点扩展(D 扩展):用于支持双精度浮点运算.它依赖于 F 扩展,需要与 F 扩展共同使用.D 扩展添加了兼容 IEEE 754-2008 算术标准的双精度浮点运算指令,并把浮点寄存器 f0~f31 拓宽到 64 位.

(5) 四精度浮点扩展(Q 扩展):用于支持四精度浮点运算.它依赖于 D 扩展,需要与 D 扩展和 F 扩展共同使用.Q 扩展添加了兼容 IEEE 754-2008 算术标准[22]的四精度浮点运算指令,并把浮点寄存器的位宽扩展到 128 位.

(6) 压缩指令扩展(C 扩展):C 扩展增加了一些 16 位的短指令编码,用于替代对应的某些 32 位常见操作指令,以减少静态和动态的代码尺寸.它还移除了原始 32 位指令上的 32 位对齐限制,允许指令可以从任何 16 位边界开始,从而显著提高了代码密度.

(7) 计数器指令扩展(Counters 扩展):Counters 扩展提供了至多 32 个 64 位计数器或计时器,可以通过特定的只读 CSR 寄存器 0xC00-0xC1F 进行访问.其中,前 3 个计数器为专用计数器,分别用于循环计数(CYCLE)、实时时钟(TIME)和过时指令(INSTRET);其余的计数器可用于程序事件的计数.

(8) 十进制浮点扩展(L 扩展):用于支持 IEEE 754-2008 标准[22]中定义的十进制浮点算术.它将浮点寄存器用于存储 64 位或 128 位十进制浮点值.该扩展尚未进入实质性的设计阶段.

(9) 位操作扩展(B 扩展):用于支持位操作.主要包括了插入、提取、测试位域的指令,以及移位、位与字节置换等操作指令.该扩展尚未进入实质性的设计阶段.

(10) 动态翻译语言扩展(J 扩展):用于支持动态翻译语言.该扩展注意到,许多流行的语言都采用动态翻译的方式实现,如 Java 和 JavaScript.因此希望引入一些指令,从 ISA 层面支持这些语言的动态检测、垃圾回收等行为,从而获得收益.该扩展尚未进入实质性的设计阶段.

(11) 事务内存扩展(T 扩展):用于支持事务内存操作.T 扩展将设立一个小型的、容量有限的事务内存缓冲区,以支持涉及多个地址的原子性操作.该扩展尚未进入实质性的设计阶段.

(12) 组合单指令多数据流(packed single instruction multiple data,简称 Packed SIMD)指令扩展(P 扩展):

用于支持小型 RISC-V 实现中的组合 SIMD 定点操作.该扩展曾被认为可以由 V 扩展上的大型浮点 SIMD 操作标准化代替,而几近废弃;但考虑到小型 RISC-V 的实际需要而得以保留,并开始设计和定义新的 P 扩展.

(13) 向量操作扩展(V 扩展):用于支持 32 位指令编码空间中的数据并行执行功能.该扩展增强了处理器的 SIMD 操作,引入了一组向量寄存器和向量操作指令,允许将相同的操作应用于大批量的数据元素之中,使 RISC-V 系统的数据处理能力发生质变,同时保持指令代码的精简性.

(14) 控制和状态寄存器扩展(Zicsr 扩展):RISC-V 定义了一个独立的地址空间,包含与各硬件线程相关的 4 096 个控制与状态寄存器(control and status register,简称 CSR).Zicsr 扩展即提供了在此空间中操作 CSR 指令的完整集合.

(15) 屏障扩展(Zifencei 扩展):仅含有一条 FENCE.I 指令,用于提供相同硬件线程上写指令内存与指令获取之间的显式同步.

(16) 非对齐原子操作扩展(Zam 扩展):用于对非对齐的原子内存操作(atomic memory operation,简称 AMO)提供标准化支持.它是对 A 扩展的扩展,使 AMO 只需要针对具有相同地址和相同尺寸的其他访问(包括非原子性的加载和存储)进行原子化执行即可.

(17) 全存储排序扩展(Ztso 扩展):用于支持“全存储排序”[23]的内存一致性模型.它提供了一种比 RVWMO 更严格的内存一致性模型.

(18) 提示暂停扩展(Zihintpause 扩展):用于指示当前硬件线程应当放缓指令的失效率.目前仅包含一条 PAUSE 指令,并且不会引起架构相关状态的更改.

(19) 半精度浮点扩展(Zfh 扩展):用于支持半精度浮点运算.添加了兼容 IEEE 754-2008 算术标准[22]的半精度浮点运算指令,以操作 16 位浮点数据.它依赖于 F 扩展,需要与 F 扩展共同使用.

(20) 半精度浮点最小集扩展(Zfhmin 扩展):是 Zfh 扩展的一个子集,仅包含了数据转移转化相关的指令,主要用于支持半精度数据的存储、以及执行更高精度的运算.与 Zfh 扩展一样,它也依赖于 F 扩展,需要与 F 扩展共同使用.

(21) 整数寄存器单精度浮点扩展(Zfinx 扩展):用于支持在整数寄存器中进行单精度浮点运算.指令功能与 F 扩展相似,只是操作数来自于整数寄存器而非浮点寄存器,而且不含 FLW、FSW、FMV 等转移指令.

(22) 整数寄存器双精度浮点扩展(Zdinx 扩展):用于支持在整数寄存器中进行双精度浮点运算.与 D 扩展类似,只是操作数来自于整数寄存器而非浮点寄存器.它依赖于 Zfinx 扩展,需要与 Zfinx 扩展共同使用.

(23) 整数寄存器半精度浮点扩展(Zhinx 扩展):用于支持在整数寄存器中进行半精度浮点运算.与 Zfh 扩展类似,只是操作数来自于整数寄存器而非浮点寄存器.它依赖于 Zfinx 扩展,需要与 Zfinx 扩展共同使用.

(24) 整数寄存器半精度浮点最小集扩展(Zhinxmin 扩展):是 Zhinx 扩展的一个子集,与 Zfhmin 扩展的功能及定位类似.与 Zhinx 扩展一样,它也依赖于 Zfinx 扩展,需要与 Zfinx 扩展共同使用.

此外,RISC-V 系统设计者还可以根据实际需要,制定其他自定义扩展指令集,并将其加入指令集架构中.

2.3 RISC-V指令集的状态

RISC-V 指令集需要经过 RISC-V 国际基金会下属的技术工作组审批,成为被批准的稳定版本,才能作为一种统一的规范和标准,供软件开发人员和硬件供应商们使用.尚未被批准的指令集可能处于“草案”或“冻结”状态:草案状态的指令集随时有改动的可能,是指令集研究中最活跃的部分;冻结状态的指令集虽然还未被批准,但在批准之前预计也不应有太显著的改动[11].表 4 总结了当前 RISC-V 各指令集的审批状态.

根据表 4 可知,当前共有 2 个基础指令集(RV32E, RV128I)和 10 个扩展指令集(Counters,L,B,J,T,P,V,Zam,Zfh,Zfhmin)仍处于草案状态,另有 5 个扩展指令集(Zfinx,Zdinx,Zhinx,Zhinxmin,Ztso)处于冻结状态,它们需要

在今后的实践中继续完善.其余 12 个指令集均已被批准,可直接使用.

Table 4 The current approval status of each RISC-V instruction set

表 4 各 RISC-V 指令集的当前审批状态

类别	指令集	内容	当前版本	审批状态
基本指令集	RVWMO	弱内存次序	2.0	已批准
	RV32I	32 位整数	2.1	已批准
	RV64I	64 位整数	2.1	已批准
	RV32E	32 位嵌入式整数	1.9	草案
	RV128I	128 位整数	1.7	草案
扩展指令集	M	乘法和除法	2.0	已批准
	A	原子指令	2.1	已批准
	F	单精度浮点	2.2	已批准
	D	双精度浮点	2.2	已批准
	Q	四精度浮点	2.2	已批准
	C	压缩指令	2.0	已批准
	Counters	计数器和计时器	2.0	草案
	L	十进制浮点	0.0	草案
	B	位操作	0.0	草案
	J	动态翻译语言	0.0	草案
	T	事务内存	0.0	草案
	P	组合 SIMD 指令	0.2	草案
	V	向量操作	1.0-rc	草案
	Zicsr	控制和寄存器	2.0	已批准
	Zifencei	屏障指令	2.0	已批准
	Zihintpause	提示暂停	2.0	已批准
	Zam	非对齐原子操作	0.1	草案
	Zfh	半精度浮点	0.1	草案
	Zfhmin	半精度浮点最小集	0.1	草案
	Zfinx	整数寄存器单精度浮点	1.0.0-rc	冻结
	Zdinx	整数寄存器双精度浮点	1.0.0-rc	冻结
	Zhinx	整数寄存器半精度浮点	1.0.0-rc	冻结
	Zhinxmin	整数寄存器半精度浮点最小集	1.0.0-rc	冻结
	Ztso	全存储排序	0.1	冻结

2.4 RISC-V权限模式

RISC-V 指令集架构必须工作在一种确定的权限模式下.根据 RISC-V 权限规范[12],目前共有以下 4 种权限模式:机器模式(Machine,M 模式)、用户模式(User,U 模式)、管理模式(Supervisor,S 模式)、监视模式(Hypervisor,H 模式).其中,H 模式暂时处于草案状态,因此通常设计时仅考虑前 3 种特权模式.

RISC-V 通过 CSR 来控制当前的权限模式.通过设置 CSR 中特定 2 位的值,可以切换到不同的模式.具体取值见表 5.

Table 5 RISC-V privilege mode settings

表 5 RISC-V 特权级模式设置

等级	CSR[9:8]编码	含义	备注
0	00	U 模式	
1	01	S 模式	
2	10	H 模式	保留用
3	11	M 模式	

2.4.1 M 模式

机器模式是 RISC-V 指令集架构中最高级别的权限模式,具有执行任何机器操作的权限,也是在系统设计中必须被实现的一个工作模式.相比之下,U/S/H 等其他权限模式都是可选的,不同的系统可以根据运行环境和实际需要,决定是否支持实现某一级别的权限模式.

2.4.2 U 模式

用户模式是 RISC-V 特权系统中最低级别的权限模式,又被称作“非特权模式”.它通常用于执行来自用户等外部环境的不可信操作,通过对其操作范围的限制来保护系统内的各种资源不受侵害.

2.4.3 S 模式

管理模式具有比用户级更高的操作权限,可以用于操作一台机器中的敏感资源.RISC-V 管理模式需要与机器模式和用户模式共同实现,因此,不能出现系统中只存在 S 模式而不存在 U 模式的情况.

2.4.4 H 模式

监视模式可用于管理跨机器的资源,或者将机器整体作为组件承担更高级别的任务.如,H 模式可以协助实现一台机器系统的虚拟化操作.

RISC-V 权限规范[12]在图 5 中直观地说明了各权限模式的执行范围.其中,Application 表示用户级应用,ABI 表示应用程序二进制接口,AEE 表示应用程序执行环境;类似地,XBI 表示应用在 X 模式下的二进制接口(binary interface),XEE 表示应用在 X 模式下的执行环境(execution environment).

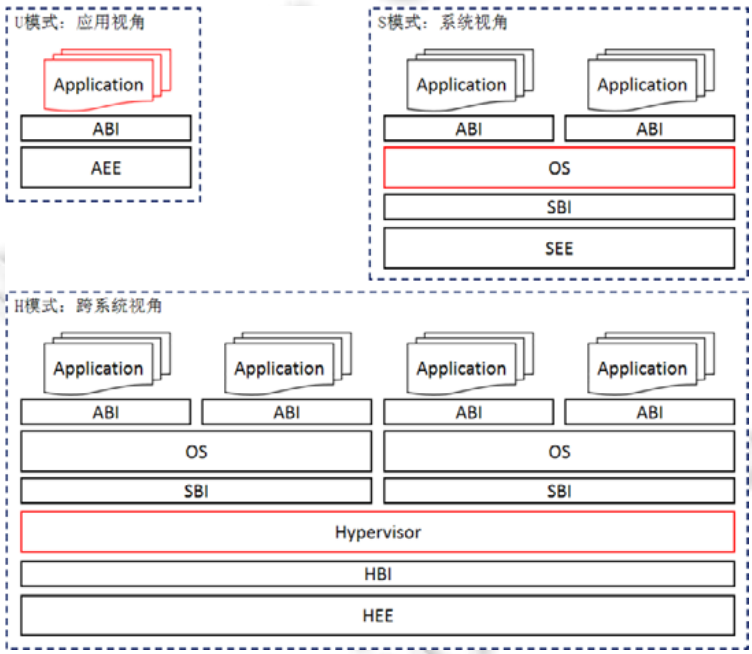


Fig.5 Execution range of different privilege modes in RISC-V[12]

图 5 RISC-V 中各权限模式的执行范围[12]

2.5 小 结

RISC-V 指令集架构正处于发展的活跃期,针对不同的应用环境提出了多种不同的指令集.RISC-V 指令集有基础指令集与扩展指令集之分,一个 RISC-V 系统需要至少完整实现一个基础指令集,并在一种权限模式中进行工作.对于指令集本身的研究是推动 RISC-V 发展的核心问题,所有指令集都需要经过批准才能成为统一而稳定的标准.目前尚有 17 个指令集未被批准,其中 12 个处于草案状态,5 个处于冻结状态.

另外,在 RISC-V 指令集发展过程中,还出现了一些其他的研究和尝试.如,2020 年,Multanen 等人[24]以 RISC-V 指令集架构为例,提出了一种基于细粒度可编程字典的指令压缩方案,通过对编译时区域的控制流分析,可在运行时有选择地更新字典内容,最小化更新开销,同时最大化利用字典空间.这些以指令集为目标的研究对 RISC-V 指令集的完善也起到了一定的促进作用.

3 RISC-V 硬件平台

RISC-V 依托的硬件平台可以含有如下组件.

(1) 带有其他非兼容 RISC-V 核心(如,IP Core)的一个或多个兼容 RISC-V 的处理核心.

(2) 固定功能加速器——这里加速器是指一种不可编程的固定功能单元,或者专用于特定任务的核心.

(3) 各种物理内存结构.

(4) I/O 设备.

(5) 组件通信与交互结构.

从系统的角度看,这些硬件平台组件可以有多种组织形式,从单核心的微控制器到包含数千个节点的集群(每个节点都是一个共享内存的多核心服务器);甚至可以在小型片上系统(system-on-a-chip,简称 SoC)中组织成多层多处理器结构,来协助模块化开发或者提供子系统间的安全隔离.

RISC-V 处理器是 RISC-V 硬件平台的主要研究对象,它可以提供相对全面的组件功能和真实的 RISC-V 执行环境.此外,RISC-V 模拟器能够在软件平台上模拟硬件执行环境,因此虽然自身属于上层 RISC-V 体系,但在研究中也应被视为 RISC-V 硬件平台的范围.本节将对 RISC-V 处理器和模拟器方面的主要研究成果进行介绍,并总结其特征和适用性.

3.1 对RISC-V处理器的研究

处理器的设计需要考虑如下问题.

(1) 所支持的指令集.这代表了该处理器能够完成的功能范围,并包含了对于指令执行通路的需求.

(2) 组成指令执行通路的组件.包括算术逻辑单元(arithmetic and logic unit,简称 ALU)、程序计数器(program counter,简称 PC)、寄存器组、缓存和存储单元等.

(3) 指令的处理方式.包括控制信号、时钟周期、指令流水线等,以及由此衍生出的任务调度、中断/陷入/异常问题的处理、数据同步等问题.

(4) 对于多核心处理器,还应考虑核心结构(同构或异构)、核间通信等方面的设计.

根据应用场景的不同,RISC-V 处理器可以划分为通用处理器、嵌入式处理器、教学研究用处理器.通用处理器不针对特定应用场景,因此将支持最全的 RISC-V 指令集和最多的功能,如 SiFive 公司在 2016 年推出的 64 位单发射五级流水线顺序处理器 Rocket[25].嵌入式处理器特别考虑了资源受限的嵌入式环境,往往需要支持 E 扩展或 C 扩展以优化尺寸,权衡面积和性能需求,如 Zaruba 等人[26]在 2019 年实现的有序单发射 64 位微核心 Ariane,支持 RV64GC 指令集.教学研究用处理器通常采用最精简的设计,只包含最基本的、必要的功能,以清晰地表现处理器的工作流程,如 Lowe-Power 等人[27]在 2019 年设计的单周期五级流水线顺序处理器 DINO CPU 以及计算技术研究所包云岗团队在 2020 年公布的单发射九级流水线顺序处理器 NutShell[28].

经 RISC-V 社区统计[29],目前公开发布并已经提交到社区的 RISC-V 处理器共计 107 种.根据这些处理器支持指令集的情况以及功能特性,本文按上述分类方法依次总结了其中较有代表性的一些处理器,并给出了相应的评价.具体内容见表 6~表 8.

从表 6 中可以得知,Rocket、freedom、BOOM 等处理器支持的指令集更加全面,便于对接各类系统,适用性最为广泛;A70X 自带工具集,开发更方便;SCR5、RiscyOO、XuanTie910 等在性能方面更有优势;DarkRISCV、NEORV32、AX45、NEOL-V 等更专注于特定结构,可以相应地满足各自特殊的设计需求.

嵌入式处理器的设计关键在于对有限资源的利用能力,或者对于资源受限环境的适应性.从表 7 可以看出,仅含 M 模式的 SCR1 和 SweRV EH2 可提供最基本的处理能力,相对适应性最强;Hummingbirdv2 E203、SHAKTI E-CLASS、Ibex 可适用于常规嵌入式环境;PicoRV32、XuanTie E906、CVA6 存在提供更多能力的可能,可以承担有更高能力要求的环境中的数据处理任务.

Table 6 Twelve representative general-purpose RISC-V processors

表 6 12 种典型的通用 RISC-V 处理器

处理器	开发者	位数	指令集	设计语言	许可证	描述
Rocket[25]	SiFive, UCB Bar	RV32	2.3-draft	Chisel	BSD	顺序核心,工具链成熟,应用广泛
freedom[30]	SiFive	RV32, RV64	2.3-draft	Chisel	BSD	用于 Freedom E310 Arty FPGA 平台
BOOM[31]	UCB Bar	RV64	2.3-draft	Chisel	BSD	Rocket 的乱序版本
VexRiscv[32]	SpinalHDL	RV32	RV32I[M][C][A]	SpinalHDL	MIT	2~5 级流水线
SCR5[33]	Syntacore	RV32, RV64	RV[32/64]IMC[FDA]	SystemVerilog	商用	多核心, 7~9 级流水线
RiscyOO[34]	MIT CSAIL CSG	RV64	RV64IMAFD	Bluespec	MIT	乱序超标量缓存一致的多处理器核心
DarkRISCV[35]	Darklife	RV32	most of RV32I	Verilog	BSD	哈佛结构,适用于 Xilinx 平台,流水线无内锁
XuanTie C910[14]	T-Head	RV64	RV64GCV + SV39	Verilog	阿里巴巴商用许可证	ISA 扩展+内存模型扩展+多核心&多集群(最多 16 核心)
NEORV32[36]	Stephan Nolting	RV32	2.2, RV32[I/E][M][A][C][Zfinx][Zicsr][Zifencei]	VHDL	BSD	大端字节序 CPU
A70X[37]	Codasip	RV64	RV64IMAFDC	Verilog	Codasip EULA	7~9 级流水线, Codaship Studio 设计工具集支持, 应用广泛
AX45[38]	Andes	RV64	RV64GCP	Verilog	Andes 商用许可证	顺序双发射 8 级流水线
NOEL-V[39]	Cobham Gaisler	RV32, RV64	RV32GC, RV64GC	VHDL	GPL, 商用	双发射顺序

Table 7 Eight representative embedded RISC-V processors

表 7 8 种典型的嵌入式 RISC-V 处理器

处理器	开发者	位数	指令集	设计语言	许可证	描述
SCR1[21]	Syntacore	RV32	RV32I/E[MC]	SystemVerilog	SHL v. 2.0	仅 M 模式; 2~4 级流水线; 可选 JTAG 调试
Hummingbird v2 E203[40]	Newclai	RV32	RV32IMAC 或 RV32EMAC	Verilog	Apache 2.0	仅 M 模式; 2 级流水线
SHAKTI E-CLASS[41]	IIT Madras	RV32, RV64	RV64[32]IMAC	BSV	BSD 3-Clause	3 阶段顺序核心
Ibex[42]	lowRISC	RV32	RV32I[M]C/RV32E[M]C	SystemVerilog	Apache 2.0	2 阶段参数化
PicoRV32[43]	Clifford Wolf	RV32	RV32I/E[MC]	Verilog	ISC	支持双端口 regfile; 指令平均周期=4
XuanTie E906[44]	T-Head	RV32	RV32IMA[F][D]C	Verilog	阿里巴巴商用许可证	ISA 扩展
CVA6[45]	OpenHW Group	RV32, RV64	RV[32/64]GC	SystemVerilog	Solderpad Hardware License v. 0.51	6 阶段单发射顺序
SweRV EH2[46]	Western Digital Corporation	RV32	2.1, RV32IMAC	SystemVerilog	Apache 2.0	仅 M 模式;9 阶段双线程双发射超标量

Table 8 Seven representative RISC-V processors for teaching and research

表 8 7 种典型的教学研究用 RISC-V 处理器

处理器	开发者	位数	指令集	设计语言	许可证	描述
DINO[27]	Jason Lowe-Power 等	RV32	RV32I	Chisel	BSD-3-Clause	单周期 5 级流水线
Tinyriscv[47]	Blue Liang	RV32	2.1, RV32I	Verilog	Apache 2.0	32 位单核心; 3 级流水线; 支持 JTAG
Taiga[48]	Reconfigurable Computing Lab, Simon Fraser University	RV32	RV32IMA	SystemVerilog	Apache 2.0	提供程序级模拟工具
Maestro[49]	João Chrisóstomo	RV32	无 FENCE/FENCE.I/CSR 相关指令	VHDL	MIT	5 阶段核心; 完全学术用
Kronos[50]	Sonal Pinto	RV32	RV32I	System Verilog	Apache 2.0	3 阶段顺序核心
NutShell (果壳)[28]	UCAS&ICT, CAS	RV32, RV64	RV64IMAC	Chisel	Mulan Permissive Software License V2	基于 NOOP 的教育项目,补全和增强了 RV64 SoC 功能
XiangShan (香山)[51]	UCAS&ICT, CAS	RV64	RV64GCB	Chisel	Mulan Permissive Software License V2	11 级流水线, 6 发射乱序处理器,是 NutShell 的进一步发展

表 8 中的 7 种处理器均在其设计目标中提及了教学背景或学术研究目的,分别针对各自不同的学术需求而设计.但像 Tinyriscv、Kronos 等处理器经过通用化后也可以具备常规的通用处理器的处理能力.这其中,较为值得一提的是中国科学院计算所包云岗团队研发的“香山”处理器核.“香山”立足于中国科学院大学“一生一芯”计划,致力于建立一个既能被工业界广泛应用、又能支持学术界试验创新想法的开源 RISC-V 核主线[52],对于开源 RISC-V 处理器的发展具有里程碑式的意义.截至目前,“香山”已经发展出了两代微架构:第一代“雁栖湖”架构已于 2021 年 7 月流片,是一个 11 级流水、6 发射、4 个访存部件的乱序处理器核,基于 28nm 工艺,频率 1.3GHz,性能达到 ARM Cortex-A76 水平;第二代“南湖”架构自 2021 年 3 月开始设计,基于 14nm 工艺,频率 2GHz,在前一代的基础上加入了对位操作指令集扩展的支持,采用指令预测与取指解耦的前端架构和更优化的后端设计,带有新 L2 缓存和双通道 DDR,预计将在 2021 年底完成.

尽管如此,除嵌入式 RISC-V 处理器对资源管理的要求较为严格外,大部分处理器都可以同时胜任通用目的、教学、研究等多种场景,仅仅是在功能、性能、复杂度等不同方面有所侧重,导致在不同场景中的表现有所差异;许多 RISC-V 处理器还提供了配置机制,可以根据用户的设定切换为嵌入式或其他工作模式(如 Rocket、Ibex、CVA6 等),或者选用不同的指令集(如 PicoRV32 等),通过有针对性的优化提升特定环境下的处理能力.

关于定制处理器的具体过程,将作为 RISC-V 系统实现的一个环节,在第 4.1.1 节进行详细论述.

3.2 对RISC-V模拟器的研究

模拟器是在软件层面对底层硬件平台或其他软件环境进行模拟的工具.它允许开发人员在非目标硬件设备上获得与目标硬件设备相同或相似的体验,简化研发流程,并有助于提前发现开发中的问题.例如,在 RISC-V 平台中使用 QEMU 模拟器运行 Linux 操作系统就是一种很常见的做法.

根据 RISC-V 社区的统计[53],目前公开发布并已经提交到社区的 RISC-V 模拟器共有 24 种,服务于各类 RISC-V 研发项目之中.它们的各自特性见表 9.

这些模拟器中,Spike 和 QEMU 是 RISC-V 国际基金会推荐使用的:它们支持几乎所有的 RISC-V 指令集,功能齐全,适用范围广,可模拟通用和嵌入式等多种处理器环境.基于这两种模拟器的研究成果也很丰富,如 Li 等人[54]在 QEMU 模拟器上进行了安全机制方面的研究,并定量分析了该成果对于 QEMU 模拟器的影响.此外,Ripes 和 WebRISC-V 等模拟器的可视化程度较高,可以直观了解硬件平台的运行情况,入门较容易.Jupiter、

Vulcan、EmulsiV 等以教学为目的的模拟器,内容简化,对初学者相对友好,易于以此为基础开展进一步的研究.

Table 9 Twenty-four RISC-V simulators

表 9 24 种 RISC-V 模拟器

名称	许可证	支持位数	支持扩展	使用场景
riscvOVPsimPlus[55]	Proprietary freeware	64 位	MACSU	裸机、交叉编译
DBT-RISE-RISCV[56]	BSD 3-Clause	32 位或 64 位	RV32IMAC/RV32GC /RC64I/RV64GC	指令集模拟
FireSim[57]	BSD	64 位	RV64	大规模集群加速硬件仿真
gem5[58]	BSD-style	64 位	RV64I/A/C/D/F	微架构模拟
OVPsim[59]	Proprietary&Apache License	32 位或 64 位	IMAFDCNSUE	多处理器共享内存配置和异构平台模拟
jor1k[60]	BSD 2-Clause	32 位或 64 位	RV32/RV64	OpenRISC、网络浏览器,2/4/8/16 核心
Jupiter[61]	GPL-3.0	32 位	RV32IMF	教学
MARSS-RISCV[62]	MIT	32 位或 64 位	RV32GC/RV64GC	全系统模拟
QEMU[63]	GPL	32 位或 64 位	RV32/RV64	标准化高性能,全系统模拟
RARS[64]	MIT	32 位或 64 位	IMFDN	汇编执行模拟
Renode[65]	MIT	32 位或 64 位	RV32/RV64	物联网
Ripes[66]	MIT	32 位	RV32I/M	可视化模拟、汇编编辑
RISC-V Virtual Prototype[67]	MIT	32 位或 64 位	RV32GC/RV64GC	SystemC 验证等
TinyEMU[68]	MIT	32 位/64 位/128 位	IFDC	系统模拟
Spike[69]	BSD 3-clause	32 位或 64 位	IMAFDQCBKV、Zifencei、Zicsr	硬件线程功能模型模拟
Swerv-ISS[70]	GPL-3	32 位或 64 位	ACDFIMSU	指令集模拟
VLAB[71]	Proprietary	32 位	RV32EC	嵌入式为主
WebRISC-V[72]	BSD 3-clause	64 位	RV64I/M	教学
PQSE[73]	Proprietary	32 位	RV32I/E/M/C	密码安全
riscv-rust[74]	MIT	32 或 64 位	RV32/64I/M/Q,部分支持 RV32/64F/D/A/C	Rust/JavaScript
terminus[75]	MIT	32 或 64 位	RV32/64I	Rust
Vulcan[76]	MIT	32 位	RV32I/M/F/A	教学
riscv-vm[77]	MIT	32 位	RV32I/M,部分支持 RV32F/A	RISCV-V 处理器模型
EmulsiV[78]	MPL2.0	32 位	RV32I	Virgule 核心教学

4 RISC-V 系统设计

硬件平台是实现 RISC-V 系统的基础和依托,而如何发挥出 RISC-V 硬件平台的优势,使其能够正确而高效地完成各项任务,则是 RISC-V 系统的设计目标.本节将从系统功能实现、系统性能提升、系统安全策略设计 3 个方面,对 RISC-V 系统的设计过程及其研究进展进行归纳分析.

4.1 RISC-V系统功能实现

根据所含处理器核心的数目和组织形式的不同,RISC-V 系统可以分为单处理器系统、多处理器系统、处理器集群系统 3 类.单处理器系统是最基础的系统形式,只包含一个主处理器,全权管理存储、运算、I/O 等系统资源,遵循冯·诺依曼结构[79]、哈佛结构[80]或者改进型哈佛结构[81]等设计架构之一,主要应用于早期 PC 平台,以及一些专用设备中.多处理器系统最早应用于高性能服务器中:IBM 在 2001 年发布了双核处理器 POWER4[82],并用其实现了最初的多核服务器系统.多处理器系统通过增加系统中处理器核心的数目来提升数据处理能力,目前已普遍应用于 PC、移动设备、分布式网络、服务器等环境.处理器集群系统是多处理器系统的扩大化和进一步发展,它由多个计算处理节点组成,而每个节点都可以是一种单处理器或多处理器系统.本节讨论了这 3 类系统的设计与实现.

4.1.1 单处理器系统

单处理器的设计通常要经过如下环节[83].

(1) 指令集与微架构的设计与实现.

(2) 体系研究与性能建模.

(3) 高层次综合(high level synthesis,简称 HLS)或寄存器传输级(register transfer level,简称 RTL)实现.

(4) RTL 验证.

(5) 关键速度相关部件(缓存、寄存器、算数逻辑单元)电路设计.

(6) 逻辑综合.

(7) 时序分析.

(8) 物理设计,包括布局、布线、版图设计.

(9) 检查 RTL、逻辑门层、晶体管层及物理层表示相符.

(10) 检查信号完整性、芯片可制造性.

在进行操作系统设计时,可以使用系统程序设计语言(system programming language,简称 SPL)对系统功能模块进行描述和开发.SPL 包括专用 SPL 和通用 SPL 两类,前者针对特定平台,如面向 MCP 操作系统的 ESPOL 语言[84];后者提供跨平台的特性,如 C/C++[85,86]、Rust 语言[87].2019 年,Lin 等人[88]描述了一种在 RISC-V 架构中使用 Rust 流和框架的方法,该方法基于 LLVM 编译器基础设施,适用于带有 SIMD 向量扩展的 RISC-V 架构,降低了系统设计的学习成本.

在指令集与处理器架构设计层面,需要根据实际功能需求,合理安排功能组件与相关资源.在这一环节,借助已有的处理器设计框架通常能起到事半功倍的作用.例如,2018 年,Zhang 等人[89]提出了组合模块化设计(composable modular design,简称 CMD)框架,用于无序处理器的设计;2019 年,Liu 等人[90]设计了行为级综合框架 ASSIST,可以根据对指令集的功能描述,协助生成一系列 RISC-V 处理器,满足不同微架构设计选择的需要;Tamimi 等人[91]则提出了一种 FPGA 软核处理器的可重构架构,用于实现基于 SRAM 的 FPGA 软核处理器.此外,当具体到特定硬件组件时:Huang 等人[92]为 RISC-V 处理器的设计提供了一套正式规范和高级抽象 ILA,侧重于加速器功能的设计;Balkind 等人[93]在 2020 年提出了支持 RISC-V ISA 的开源硬件框架 BYOC,提供了一个可扩展的缓存一致性系统.

硬件描述语言(hardware description language,简称 HDL)是用于描述电子元件或数字逻辑电路的结构、行为、数据流向等信息的设计语言.处理器架构的设计方案经 HDL 描述后,形成对数字系统的分层模块表示,然后将进入电子设计自动化(electronic design automation,简称 EDA)工具逐层仿真验证、自动综合工具将模块组合转换为门级电路网表的过程,最后便可使用专用集成电路(application specific integrated circuit,简称 ASIC)或现场可编程门阵列(FPGA)的自动布线工具,将网表转换为具体的电路布线结构,交至硬件实现.根据第 3.1 节对各种处理器的统计,RISC-V 系统多采用 Chisel、Verilog、SystemVerilog、SpinalHDL[94]等作为其硬件描述语言.2018 年,Taylor 还为 SystemVerilog 设计了一个可用于系统合成的标准模板库 BaseJump[95].在 16nm TSMC Celerity SoC、511 核 RISC-V 处理器和 385M 晶体管条件下的实验结果表明,该设计能够将 80%的模块直接从 BaseJump STL 实例化,从而有效减少数字电路设计所需的时间,提升工作效率.SpinalHDL 是一种基于 Scala 的硬件描述语言,它兼容 EDA 工具、学习成本低、设计时不引入额外的面积或性能开销,在实践中被用于基于 RISC-V 的处理器开发.例如,VexRiscv 处理器[32]便是由 SpinalHDL 开发而成.

由于单处理器系统与多处理器系统、处理器集群系统存在结构和功能上的发展联系,关于单处理器系统的设计方案和研究成果同样可以作为原型,迁移到多处理器系统和处理器集群系统之中,作为后者的研究基础和实践参考.

4.1.2 多处理器系统

多处理器系统的设计,在单处理器系统的基础之上,更需要关注处理器核心增加带来的核间管理与通信问题,关注数据的并行或并发处理能力.具体来说,多处理器系统中可能存在如下挑战[96].

(1) 存储一致性与缓存共享问题.

(2) RTL 电路大幅增长带来的复杂性问题.

(3) 组件依赖性问题.

(4) 通信延迟问题.

针对上述挑战,研究人员作出了许多有益尝试,提出了相应的解决方案.

例如,对于存储一致性问题与 RTL 电路问题,2017 年,Manerkar 等人[97]介绍了一种用于缩小微架构/RTL 存储一致性模型(memory consistency model,简称 MCM)验证差距的方法和工具 RTLCheck.RTLCheck 给定了一组关于 MCM 行为的微架构公理、一个用于弥补 RTL 设计与用户之间差距的的映射,并自动生成 SystemVerilog 断言,来验证其是否满足指定测试程序的微架构规范.作者团队在 RISC-V V-scale 处理器的多核版本上评估了 RTLCheck,验证了其有效性.

对于组件依赖性问题,2020 年 Kurth 等人[98]提出了一个模块化、高效、开源的原子单元(atomic unit,简称 ATUN)架构,它可以在不同的内存结构层次中灵活实施,使任何级别的内存层次结构都可以实现可扩展的 AMO.他们在一个具有 32 核 RISC-V 处理器的 FPGA 原型上对 ATUN 进行了演示,证明了该架构的有效性.

Pulte 等人[99]则在 2019 年提供了可用于 RISC-V 的更抽象的操作模型 Promising-ARM/RISC-V 及其交互式探索工具,以彻底检查多种并发行为.

关于以通信延迟为代表的系统性能问题,将在第 4.2 节展开讨论.

4.1.3 处理器集群系统

处理器集群系统由于需要对多个节点统筹管理,因此在设计时往往需要考虑多方面因素,包括节点的高性能、容错能力、可扩展性、稳定性、节点间耦合程度等[100].例如,2021 年,Glaser 等人[101]描述了一个轻量级的、硬件加速的同步和通信单元(synchronization and communication unit,简称 SCU)的设计方案,并将其集成到了一个 8 核心的 RISC-V 处理器集群之中,使其性能平均提升了 23%、能效平均提升了 39%.

其中,集群各节点之间的耦合程度是处理器集群系统设计中的一个重点问题,它将影响到节点的功能划分、节点间通信频率、单个作业规模等多个设计问题的决策.如 Shi 等人[102]在 2019 年提出了一种专用架构 E-LSTM,将嵌入式 RISC-V CPU 与 LSTM 协处理器进行紧耦合,来适应芯片面积和数据访问带宽有限的应用环境,最终对处理吞吐量实现了 2.2 倍的提升.

4.1.4 测试与验证

由于涉及到硬件成本问题,系统的设计方案需要经过测试和验证,确认其功能是否正确完整、性能是否达标、是否具备足够的安全性和稳定性后,才能大规模投入生产.系统的测试是指,在目标实验环境中对系统的各项指标数据进行提取,探索其可达程度范围的过程;系统的验证则是在给定一个预期表现后,判断系统的表现是否与之相符的过程.二者都是对系统进行评估的有效手段.

在系统设计的任何一个环节都需要有与之对应的测试或验证方法来保障其正确性.比如,在内存问题上,2019 年,Armstrong 等人[103]为包括 RISC-V 在内的 3 种主流架构的大部分顺序行为提供了严格的语义模型,并使用 ISA 语义的自定义语言 Sail 实现了这些模型.他们还将 RISC-V 模型集成到了 RMEM 工具中,可使用户模式下的内存并发探索更加容易进行,为软件验证建立了稳固的基础.

在处理器架构层面,2019 年,Deng 等人[104],提出了一个用于安全处理器架构的设计时安全验证框架 SecChisel.该框架以 Chisel 语言和工具为基础,并在设计时使用信息流分析来验证架构的安全性.在用 AES 和 SHA 模块扩展的 RISC-V Rocket 芯片上对该框架进行了评估.Nelson 等人[105]也在 2019 年提出了一种用于系统软件开发自动验证器的框架 Serval,并使用 Serval 为包括 RISC-V 在内的 4 种指令集构建了自动验证器.

在集群网络层面,2018 年 Khamis 等人[106]提出了一种片上网络(networks-on-chips,简称 NoC)硬件仿真和测试台加速的协同建模框架,用于对各种基于网络方案的 NoC 进行验证和评估.其中,通信量通过开源 RISC-V 指令集架构注入或自行生成,来作为处理块(processing tile).

在系统安全策略方面,2019 年 Dessouky 等人[107]构建了一个基于 RISC-V SoC 的、可被真实世界软件利用

的 RTL 缺陷测试平台,并基于所提出的测试平台,分析了一些安全验证方法的有效性,确认了一种现有方法无法检测到的特定类型漏洞,称为 HardFails.随后在 2020 年,Nelson 的团队又提出了将形式化方法应用于 BPF(Berkeley packet filter)虚拟机的即时编译器 JIT(just in time compiler)方法[108],证明可以在一个大型的未验证系统中构建一个经验证的组件,并基于 32 位 RISC-V 开发了一个用于编写 JIT 并证明其正确性的框架 Jitterbug.

4.2 RISC-V系统性能优化

系统优化的主要目标在于提升系统的运行效率,或者减少对资源和能源的消耗.系统运行的各个环节都可能涉及到对性能的评价,而其中最主要的优化需求集中在处理器、内存、通信、能耗 4 个方面.本节即围绕这 4 个方面,对 RISC-V 系统性能优化的思路和有关成果进行详述.

4.2.1 处理器利用率提升

处理器中的性能优化主要体现在其利用率的提升.处理器利用率反映了处理器有效工作时间在总运行时间中的占比,是一项描述处理器繁忙程度的指标.减少处理器空闲等待的时间,提升处理器利用率,将推动系统整体的计算能力或数据处理能力的提升,使系统能够在更短时间内完成既定工作,或者在相同时间内完成更多工作.

“冯·诺依曼瓶颈”是影响 CPU 利用率的一类典型问题.在冯·诺依曼结构[79]的计算机系统中,指令与数据被不加区分地存放在内存,使得取指令和取数据不能同时进行,否则将引起访存混乱.CPU 在执行运算前后,都需要额外的时间等待数据完成存取,而不能一直处于工作状态.因此,冯·诺依曼结构中的 CPU 存在一个相对较低的利用率上限,无论硬件制造工艺如何提升,都无法再获得更高的 CPU 利用率.

相应地,在 2017 年,包云岗等人[109]提出一种标签化冯·诺依曼体系结构 LvNA(labeled von neumann architecture),在经典冯·诺依曼结构上增加了一套基于标签机制的可编程接口,允许向硬件传递更多软件信息,使硬件可以根据软件需求动态调整管理策略.2019 年,该团队主导的“标签化 RISC-V”开发项目[110],基于 RISC-V Rocket Chip 实现了 LvNA,显示了 RISC-V 在促进敏捷开发、提升编码效率方面的优势.在 2021 年,Schuiki 等人[111]提出了一种轻量级的 RISC-V 扩展“流语义寄存器(stream semantic registers,简称 SSR)”,允许在任何指令中编码加载和存储操作,而不再需要依赖显式的加载与存储指令,从而给出了一种解决单发射处理器中存在“冯·诺依曼瓶颈”、影响 CPU 利用率问题的方案.实验结果显示,采用 SSR 扩展的处理器比起未采用的相同处理器,顺序代码单核心运行速度提升 3 倍,即,集群中实现相同的性能可减少 3 倍的核心数目;多核心集群的能效提高 2 倍,指令获取的数量减少 3.5 倍,指令缓存功耗减少 5.6 倍.

4.2.2 内存优化

内存的设计直接制约着系统整体的组织形式和工作方式,在数据存储、通信、同步、处理效率等多方面都对系统有关键性的影响.因此,内存优化问题始终是系统性能研究的一大重点.

例如,多核处理器中,随着芯片上核心数量的增加,会出现“缓存行乒乓”问题,即,当多个 CPU 共享同一缓存行中的变量时,不同 CPU 对该变量的频繁读写会导致其他 CPU 的缓存行频繁失效.共享内存中,硬件缓存一致性范式的线程同步算法,其性能扩展问题会受到缓存行乒乓的阻碍.2019 年,Dogan 等人[112]为了解决多核处理器的缓存行乒乓问题,提出了一种针对数据的硬件内移动计算(moving computation,简称 MC)模型,该模型将共享数据固定在专用核上,以实现数据局部性优化.研究人员还在 RISC-V 上建立了多核仿真环境,评估了 MC 模型在最多 1024 核尺度下的性能扩展优势.评估结果显示,与自旋模型、原子模型等现有模型相比,对每个数据点标准化的完成时间平均分别提升了 60%、27%.

扩展基全局地址空间(extended base global address space,简称 xBGAS)是 2018 年 Leidel 等人[113]为了解决可扩展的高性能计算架构在操作离开单个设备域时常常遭遇不必要的延迟的问题,所提出的一种 RISC-V 指令集微架构扩展.该扩展可以为常见的高性能计算操作提供紧耦合的支持.由于其提供了从基本指令访问全局共享内存地址空间的可能性,从而开创了一条系统优化的新思路,出现了更多以 xBGAS 为基础的研究.

例如,2019 年 Williams 等人在 xBGAS 的基础上,提出了一种构建集体通信库的 RISC-V 微架构扩展方

案[114],给出了它的初始 API 设计和实现,以及底层算法.这项研究的目的是在分布式地址空间模型中,将更直观的界面与更高的性能结合起来,以解决软件开发人员很难适应并行编程方法,尤其是分布式地址空间中的并行编程问题.

2020 年 Wang 等人提出了一种远程原子扩展(remote atomic extension,简称 RAE)的设计[115],为基于 RISC-V ISA 架构的远程原子操作提供了内在的 ISA 级指令和架构支持,以改善高性能计算(high performance computing,简称 HPC)系统的性能.这个设计方案同样基于 xBGAS 扩展,具体内容见第 5.1 节.

4.2.3 通信延迟缓解

通信是连接系统各组件、各成分之间的信息交换过程,通信延迟将推迟目标获得所需信息的时间,从而增大其空闲等待时间,造成整体用时延长、目标利用率降低、或者能量空耗.缓解通信延迟,除了期待相关硬件制作工艺的改进外,提升系统的并行能力、掩盖通信延迟的不利影响也是一种可行的做法.

这方面,Morais 等人[116]在 2019 年提出了一种将任务调度硬件加速器与通用 CPU 紧密集成的体系结构,以减少通信延迟及运行开销,从而提高与任务调度并行的应用程序的性能.他们开发了硬件加速的轻量级任务调度运行时环境 Phentos,允许任务调度软件通过自定义指令与硬件任务调度器和 CPU 直接进行交互,以最大限度减少硬件任务调度器和 CPU 之间的通信开销.Morias 等人认为,任务并行系统的性能会受到自动依赖推理和准备运行任务调度相关开销的限制,而用于提高运行时性能的硬件加速器往往具有较高的 CPU 通信开销,因此他们希望通过这样的设计,来解决 CPU 通信的高开销问题.

4.2.4 能耗优化

能耗优化依赖于对系统负载结构的整体设计,现有研究主要集中在功率控制、电源能效优化等方面.

关于功率控制,2019 年 Zhou 等人[117]介绍了一种基于学习的架构 PRIMAL,它训练了带有设计验证平台的机器学习模型.训练后的模型可以用于生成不同工作负载下的相同块的详细功率剖面.该模型还评估了几种已有机器学习模型的性能,包括:岭回归、梯度树提升、多层感知器、卷积神经网络.实验中,基于卷积神经网络的方法对一个 RISC-V 处理器核实现了 35 倍的加速和 5.2%的逐周期功率估计误差.

2020 年 Bambini 等人基于 RISC-V 核心的并行超低功耗微控制,提出了一种开源功率控制器系统(power controller system,简称 PCS)设计方案[118];同时,在一个开源的实时操作系统中,实现了一种可配置的热功率控制算法.通过对热功率进行控制,在能耗和系统稳定性两方面都取得了较好的收益.

在电源的能源效率提升方面,Wright 等人于 2020 年设计了一种双核 RISC-V 处理器[119],实现了一个完全集成的、片上的、高密度的、由开关电容 DC-DC 转换器供电的细粒度电压域系统.他们希望用这种方式,通过架构和电路的精心设计,抵消 Dennard 定律终结之后日益显著的功耗问题,提高能源效率.他们的实验结果表明,在系统空闲期间,动态开关功率可以降低接近 62%的理想水平,而对总体计算吞吐量几乎没有影响.

4.3 RISC-V系统安全策略设计

系统在工作过程中随时会面临安全威胁,有时是操作者自身的操作失误,有时则是纯粹的恶意攻击和入侵.这些威胁可能导致系统异常和数据破坏,进而造成更大的损失.因此,系统需要采用一定的安全策略来应对这些威胁,维护运行的可靠性和稳定性.本文根据安全威胁发生的主要渠道,将系统可能遭受的安全威胁归纳为如下 3 个方面.

(1) 硬件微架构攻击.

这类攻击通过侵入系统所依赖的硬件微架构并将攻击代码注入底层硬件,造成系统功能的偏差或失效.由于微处理器等各种集成电路的广泛应用,即使仅针对一种型号发起攻击,也可产生相当规模的破坏力.

(2) 内存攻击.

这类攻击以干扰或破坏内存功能为主要手段.内存是存储和提供处理器运算所需信息和数据的关键部件,针对内存进行攻击将严重威胁系统功能的正常运行.不可信代码、软件漏洞、系统弱点都有可能被利用,成为内存攻击的发起点.

在众多的内存攻击中,程序控制流劫持攻击是较为常见和重要的一种攻击类型.这类攻击通常会制造和利

用内存损坏,然后将控制流重定向至攻击者指定的内存位置,使程序脱离正常执行流程,按攻击者的意图执行.

(3) 侧信道攻击(side channel attack,简称 SCA).

这类攻击往往从外部发起,围绕构成系统的硬件设备的物理特征或电气特性实施,意图通过测量、统计、物理观察等手段推测和获取系统敏感信息,造成信息泄露.获得的信息更可作为其他攻击的基础,形成组合攻击,进一步损害系统安全性.

针对上述安全威胁,学术界和工业界提出了许多安全策略.本文将相关方案要点梳理总结如下.

(1) 引入安全硬件,将软件安全威胁转移到硬件层面解决.

(2) 模糊易受攻击的敏感信息,增大攻击者搜索和获取敏感信息的时间或技术成本.

(3) 保护信息流完整性,保障程序执行安全.

(4) 设置配套验证机制,及时发现攻击行为.

(5) 必要时采用冗余设计,增加系统容错性,减少对特定设备的依赖.

以下是对各类有代表性的安全威胁及其应对措施的详细介绍和分析.

4.3.1 硬件微架构攻击的防御

(1) 威胁分析

硬件微架构是指令集体系结构在处理器中具体执行方法的体现,直接关系到指令系统实现系统功能的能力.由于硬件的生存周期相对较长,一般可达数年之久,通常需要经过多次的微代码更新来解决安全或功能性问题,因此微架构与微代码的设计在引入一定的灵活性的同时,也留给了恶意攻击侵入的空间.例如,微码木马可以事先被注入到系统的供应链中,并在硬件更新时完成潜入;当满足特定执行条件时,即可释放和执行所载有的恶意微码,对系统造成侵害.2021 年,Nils Albartus 等人[120]研究了嵌入式硬件环境下的微码木马,揭示了这一侵害方式的具体流程,并在一次攻击实验中成功获取了特定于架构的 AES 秘钥.

(2) 解决方案

应对硬件微架构攻击,可以从供应链安全和微架构安全两方面着手.供应链安全可从源头切断恶意微码的注入渠道,从而避免硬件微架构攻击的发生;微架构安全需要在恶意微码已经侵入时进行有效遏制,增加恶意攻击成功实施的困难程度.通常,微码木马包括触发器和有效负载两部分,前者用于在特定条件下触发木马,后者包含具体的恶意攻击代码,并将在木马被触发时发动攻击.因此,对微码进行加密或对敏感区域进行隔离,以防止相关信息泄露达成触发条件,是一种常见的做法.2020 年,Lee 等人[121]提出了一套可定制的可信执行环境(trusted execution environments,简称 TEE)开放框架 Keystone,通过构建可重用的 TEE 核心原语,实现了对不受信任组件的可编程层的隔离.

4.3.2 程序劫持攻击的防御

(1) 威胁分析

程序劫持攻击是通过对程序的控制流进行干扰、误导、破坏,使程序按攻击者意图执行的一种攻击手段.制造并利用内存损坏,然后将控制流重定向至攻击者指定的内存位置,是程序劫持攻击中的典型做法[122].例如,缓冲区溢出攻击[123]可以通过构造足够长度的字符串,充满并溢出缺乏边界保护机制的缓冲区,将含有恶意代码地址的数据送入保存返回地址的内存区域,覆盖原有的程序返回地址.这样,在当前程序块返回时,便将自动执行恶意代码.Cowan 等人在他们的工作[124]中描述了这个过程,如图 6(a)所示.

(2) 解决方案

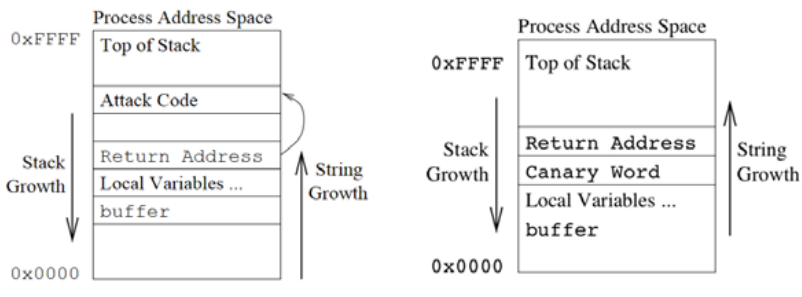
控制流完整性(control-flow integrity,简称 CFI)[125]是应对程序劫持攻击的主要技术措施.它的核心思想是为程序构造对应的控制流图(control flow graph,简称 CFG),并保证程序在执行过程中严格遵守该控制流图,而不会跳转到图以外的其他地址范围之中.关于如何保证程序的执行严格遵守控制流图,研究人员提出了许多具体的方法.例如,在上文提到的缓冲区溢出攻击场景[123]中,基于金丝雀字(canary)的攻击检测技术便是一种可行的设计.金丝雀字是置于返回地址边缘的一段特殊字符,当溢出发生时,金丝雀字将首先被改变.因此,只需核对金丝雀字是否正确,即可判断当前是否已遭到缓冲区溢出攻击.图 6(b)描述了利用金丝雀字技术检测缓冲区溢出攻

击的主要过程.

为了防止攻击者事先猜测到金丝雀字的正确内容,De 等人[126]提出了一种基于安全硬件指纹的随机金丝雀字生成方法 PUFCanary,并在 RISC-V Rocket 芯片中实现了该方法.PUFCanary 随机生成的金丝雀字无需保存在内存或寄存器中,减小了攻击表面.在多缓冲区场景下,PUFCanary 能提供针对单一缓冲区的细粒度保护.此外,PUFCanary 还可以在编译器的支持下,检测源于缓冲区溢出的数据流攻击.在 Wilander 缓冲区溢出套件[127]中的测试结果表明,PUFCanary 能够阻止所有 8 个可移植到 RISC-V 工具链中的栈溢出攻击用例.

基于内存保护的设计也是对抗程序劫持攻击的一种基本方法,具体可以包括边界检查、地址布局随机化、返回地址验证等内容.2020 年,Li 等人[54]为了提高消息认证码(message authentication code,简称 MAC)的安全机制性能,以较低开销实现加密控制流完整性(cryptographic control flow integrity,简称 CCFI)和 ARM 指针认证(pointer authentication,简称 PA),并提高其对于重用攻击的防御能力,提出了一种堆栈布局,将之前的返回地址推送到当前的堆栈帧上,并在函数调用时,将当前返回地址存储到硬件缓冲区中.设计了两种基于错位堆栈的验证方法:延迟验证和批验证.延迟验证直到从堆栈弹出的返回地址被返回时才进行验证;批验证一次验证两个返回地址.他们在 RISC-V 架构上实现了这两种验证方法,并定量分析了它们对于 QEMU 模拟处理器的影响.

基于安全硬件的软硬件协作方案为实现控制流完整性、抵御程序劫持攻击提供了另一种思路.例如,2018 年 Ferraiuolo 等人[128]提出了一种增强安全信息流(包括控制定时信道)的处理器 HyperFlow.它使用一种类型安全的硬件描述语言实现,允许在运行时配置复杂的信息流策略,并在一个提供了完整 RISC-V 指令集的处理器上对 HyperFlow 架构进行了原型实现.2019 年 Gallagher 等人[129]提出了一种安全硬件架构 Morpheus,通过模糊攻击者需要但普通程序不使用的信息,防止利用滥用程序和机器级语义之间的鸿沟进行的攻击,如定位敏感指针、利用 bug 覆盖敏感数据、劫持受害者程序的执行等.他们通过一个基于 RISC-V 指令集的处理器原型演示了 Morpheus,成功阻止了所设计的控制流攻击.2020 年 Bresch 等人[130]提出了一种软硬件系统设计框架 TrustFlow-X,针对基于内存的攻击提供了有效的细粒度控制流完整性保护.他们使用 LLVM 编译器工具链生成 TrustFlow-X 的安全代码,然后在扩展的 RISC-V 处理器上执行这些安全代码,达到了预期保护效果.



(a) (b)

Fig.6 Process of buffer overflow attack (a) and detecting (b)

图 6 缓冲区溢出攻击(a)及利用金丝雀字技术检测攻击(b)的过程

4.3.3 其他内存攻击的防御

(1) 威胁分析

基于内存的攻击不仅可以作为程序劫持攻击的基础,还可能造成许多其他的威胁,比如面向数据编程(data-oriented programming,简称 DOP)攻击.这一类型的攻击往往通过内存行为模拟操作,对数据流中的非控制数据进行攻击.Hu 等人[131]总结了一些关于内存攻击的具体行为类别,包括模拟算术操作、模拟赋值操作、模拟解引用(dereference)操作等.不同的内存攻击行为需要根据其特征,有针对性地缓解和解决.

(2) 解决方案

针对基于内存的攻击,可以有软件和硬件两种不同的应对方式.软件方式如,2019 年 Nyman 等人[132]提出的

运行时作用域增强(run-time scope enforcement,简称 RSE)的方法,通过增强编译时内存安全性约束(如变量可见性规则),有效地减轻所有当前已知的 DOP 攻击.在此基础上,给出了一种 RISC-V 硬件辅助 RSE 的概念实现 Hardscope.硬件方式如,2017 年 Menon 等人[133]提出了一个处理内存攻击的统一硬件框架,并将之与 RISC-V 微架构结合,形成了 Shakti-T 原型系统.这一方案通过为系统提供额外的硬件支持,可以保护敏感应用(如电子商务交易等)的信息免受基于时间或空间的内存攻击.

2020 年,De 等人[126]为他们所提出的硬件生成堆栈金丝雀字引擎 PUFCanary,设计了一个专用的嵌入式 RISC-V 安全协处理器流完整性扩展——FIXER.除缓冲区溢出攻击外,FIXER 还可以帮助 RISC-V 架构对抗代码注入、代码重用等类型的攻击.

其他有代表性的研究成果还包括:Wong 等人[134]在 2018 年将内存保护单元(memory protection unit,简称 MPU)集成到 RISC-V 可信 SoC 中的设计,并以此实现了一个设计框架 SMARTS,以实现对已知攻击向量的轻量级但健壮的对抗.该框架可以为 SoC 提供完整性和保密性,并允许根据需求进行灵活的部分加密.2020 年,Kokologiannakis 等人[135]提出了一种高效的无状态模型检查(Stateless model checking,简称 SMC)算法——HMC 算法,用于在任意硬件内存模型下,以合理、完整和最优的方式验证程序.

4.3.4 侧信道攻击的防御

(1) 威胁分析

侧信道攻击是指攻击者从系统的物理实现出发,通过测量、统计、物理观察等手段推测和获取系统敏感信息的攻击手段.侧信道攻击是密码系统安全的主要威胁之一,因此长期受到业界关注.早在 2011 年,Persial 等人[136]便总结了可能的侧信道攻击种类,他们根据所依托介质的不同,将侧信道攻击划分为基于时间、功率、电磁、错误注入、光学、流量、声学、热成像的八种具体的攻击形式,并重点分析了基于功率的攻击和基于错误注入的攻击.在 2017 年,Guo 等人[137]分析了针对密码设备的侧信道攻击,讨论了在面对不同加密算法(AES、DES、RSA、ECC)时的具体攻击过程,并重点研究了应对差分能量分析(differential power analysis,简称 DPA)的保护策略.Saxena 等人[138]在 2018 年调研了基于缓存的侧信道攻击,总结了一些可以应用于云计算环境中的防御方法.Devi 等人[139]在 2021 年对物联网环境中的侧信道攻击进行了分析,将其划分为简单攻击、差分攻击、功率分析攻击、故障攻击、时间攻击五类.

此外,侧信道攻击还可能与其他攻击手段相结合,作为组合攻击的一个环节发起其他攻击.例如,2019 年,Van Bulck 等人[140]揭示了几个在应用程序二进制接口(ABI)和应用程序编程接口(application programming interface,简称 API)级别上的侧信道漏洞,攻击者可以通过这些漏洞发起对内存安全和编译包的进一步攻击.他们对 TEE 的漏洞空间进行了分析,论证了最先进的缓解技术,如 Intel 的 edger8r 等,都无法完全消除这种攻击面.2020 年,Spruyt 等人[141]也发现了现有的侧信道攻击可以直接应用于纯错误注入(fault injection,简称 FI)攻击的设置,并提出了一种对故障概率轨迹执行相关功耗分析(correlation power analysis,简称 CPA)的技术方案——故障相关性分析(fault correlation analysis,简称 FCA).

Ahmadi 等人[142]在 2020 年讨论了针对 RISC-V 处理器的侧信道攻击问题.他们认为,RISC-V 处理器作为一种开放式硬件,在功率、时间和温度 3 个指标上最易受到利用而泄露信息.其中,功率侧信道攻击和时间侧信道攻击是最主要的两种可由软件发起的攻击方式.但是,对于如何应对这些威胁,文中并没有给出太多的分析,仅列举了一种已有的安全解决方案[143].

(2) 解决方案

经前文分析可知,侧信道攻击可能通过多种渠道发起,以基于功率、时间、电磁等为主.现有的解决方案,也都是适用于其中一种或几种渠道的针对性防御方法.

例如,2019 年,De Mulder 等人[143]将侧信道分析攻击对策集成到 RISC-V 中实现,发现这种做法可以防止 first-order 功率或电磁攻击,同时尽可能地降低实现成本.

同年,Deng 等人[144]针对从转换检测缓冲区(translation lookaside buffer,简称 TLB)到基于时间的侧信道攻击,研究了一种自动生成测试 TLB 漏洞的微安全基准方法,并提出了两种安全 TLB 的设计方案:静态分区(static-

partition,简称 SP)TLB 和随机填充(random-fill,简称 RF)TLB.他们使用 Rocket 开源 IP 核实现的 RISC-V 处理器架构评估了所提出的安全 TLB.

2020 年,Sehatbakhsh 等人[145]提出了一种模拟电磁侧信道信号的方法 EMSim,可以使用器件架构模型对电磁侧信道信号进行逐周期模拟,以定量分析侧信道泄露,从而为设计对侧信道攻击具有高弹性的系统打下基础.

4.3.5 安全策略总结

表 10 总结了本节提及的各类 RISC-V 系统中可用的安全策略.

Table 10 Summary of RISC-V security mechanisms

表 10 RISC-V 安全策略总结

威胁类别	防御原理	优势或能力	劣势或注意事项	相关工作
硬件微架构攻击	供应链安全	从源头防止恶意微代码的注入	要求对供应链各个环节都有足够的防护,波及面广	Keystone[121]
	微架构安全	增强微码信息或代码执行信息的不透明度	力度有限,难以完全防御	Microcode CPU[120]
内存攻击(程序劫持)	控制流完整性	保证控制流转移目标范围;技术相对成熟	粗粒度 CFI 无法彻底保护控制流;细粒度成本较高	PUFCanary[126]
	基于内存保护	规范内存边界,在内存布局层面提供保护,验证方便	要求栈空间具有足够的隔离性	MAC[54]
	基于安全硬件	通过安全的硬件代码设计和制定信息流策略;可利用硬件信息进行防御,减少对软件应用运行的干扰	需要定制安全硬件,成本相对较高	HyperFlow[128], Morpheus[129], TrustFlow-X[130]
内存攻击	内存安全性约束	有效对抗 DOP 攻击	需结合辅助硬件使用	RSE[132]
	辅助硬件	从硬件层面根本上保证强隔离性和强对抗性,简化	需要引入额外的辅助硬件,成本相对较高	Hardscope[132], Shakti-T[133], FIXER[126]
	内存保护单元	通过加密设计提供轻量级但健壮的对抗方式	只针对已知攻击向量,保护范围有限	MPU[134]
	程序验证	理论上支持对任意内存模型的检测	并非直接的对抗手段	HMC[135]
侧信道攻击	电磁对策	有效对抗基于电磁的攻击	需结合具体威胁模型分析	对策集成[143]
	时间对策	有效对抗基于时间的攻击	需结合具体威胁模型分析	安全 TLB[144]
	测试与检验	标准明确,判断准确	并非直接的对抗手段	EMSim[145],FCA[141]

5 RISC-V 应用场景分析

本节将列举一些利用 RISC-V 促进特定领域应用研究的案例,并对 RISC-V 在其中所能发挥的作用以及应用要点进行简要的分析.其中,第 1 个应用[115]描述了一种对 RISC-V 指令集自身的扩展场景;第 2 个应用[19]介绍了利用 RISC-V 实现特定领域功能的系统设计场景;第 3 个应用[146]研究了一种发掘 RISC-V 系统潜力、提升系统性能的方法;第 4 个应用[147]围绕系统安全主题,讨论了一种利用 RISC-V 实现安全批准的实施方案.

5.1 RAE:一种远程原子扩展

5.1.1 场景描述

在图形分析、机器学习、数据驱动的科学计算等新兴的数据密集型应用的推动下,HPC 系统的结构逐渐复杂化,大量数据集被映射到离散的节点中,以实现分布式存储和计算.这种数据分布将导致频繁的跨节点数据事务.尤其是在全局原子操作中,跨节点的读写修改等操作包含了多个操作步骤和特定的原子性管理,导致了大量开销.解决开销问题需要一种有效的通信方法,并结合相应的软件基础设施来实现对离散组件的集成,减少跨设备的冗余软件例程.为此,在 2020 年,Wang 等人提出了一种 RAE 的设计[115],为基于 RISC-V ISA 架构的远程原子操作提供了内在的 ISA 级指令和架构支持,最终改善 HPC 系统的性能.

RAE 首先引入了已有的 xBGAS 扩展指令集以支持全局内存寻址功能.然后,设计了一系列原子指令,通过 xBGAS 寻址来执行远程原子操作.这些操作共有 7 类:fetch-and-add、fetch-and-xor、fetch-and-or、fetch-and-and、fetch-and-max、fetch-and-min、compare-and-swap,每一类都包含一条远程原子操作指令和一条对应的本地原子指令,均采用 R 型指令格式.因此,RAE 扩展共包含了 14 条 R 型原子操作指令.图 7 以 fetch-and-add 指令为例,

描述了一次原子操作的过程.图 8 描述了 RAE 所依托的 xBGAS 原子设计架构.

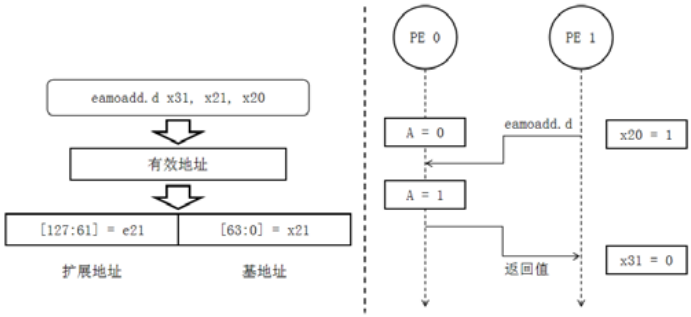


Fig.7 Fetch-and-add atomic operation process[115]

图 7 Fetch-and-add 原子操作过程[115]

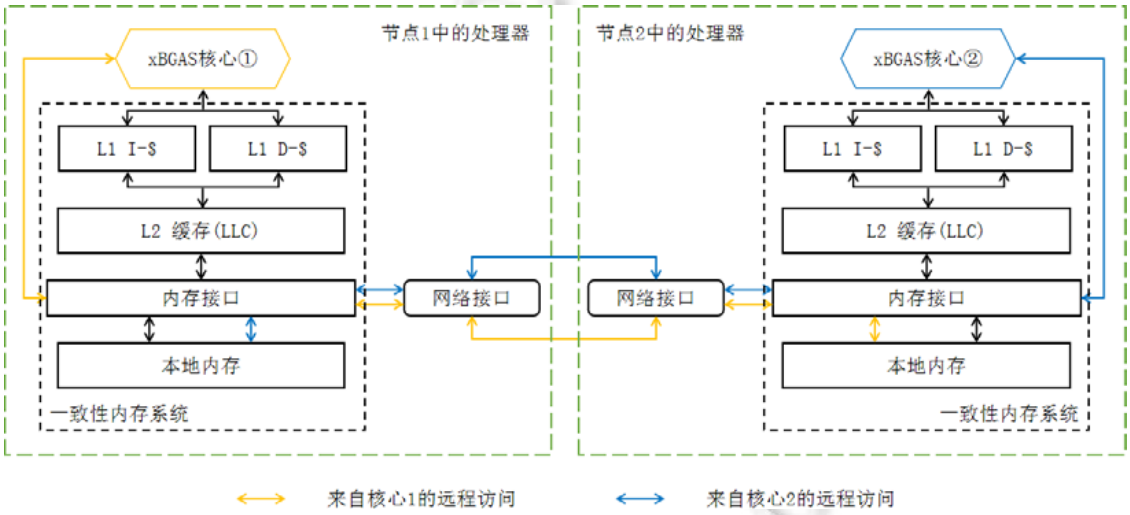


Fig.8 Architecture of Atomic Design in xBGAS[115]

图 8 xBGAS 中的原子设计架构[115]

在此基础上,RAE 还引入了网络接口控制器(network interface controller,简称 NIC)等基础设施来处理远程请求,以及操作映射表(operation mapping table,简称 OMT)将远程请求解包并转换为本地操作,以提供指令执行的效率.

5.1.2 场景分析

对于 RISC-V 指令集自身的扩展研究需要考虑如下问题.

(1) 工作场景或功能目标

(2) 各指令的功能、格式信息

(3) 实现的可行性

在本例中,RAE 将工作场景约束在了高性能系统中的分布式数据跨节点操作环境,并以实现高效的远程原子操作为自身的功能目标,从而确定了各指令应完成的功能.根据指令的功能,判断其所含操作数的情况,选择合适的指令格式进行设计.RAE 的 14 条指令均带有两个操作数,并且需要支持 xBGAS 全局寻址,因此选择了 R 型格式.在指令的实现方面,RAE 设计了各指令的具体工作流程,并引入了 NIC 等相应的基础设施和 OMT 等机制,给出了一套完整且效率理想的实现方案.

5.2 SALSA:一个用于序列比对的领域专用架构

5.2.1 场景描述

基因组数据正在个性化医疗和农业科技等领域发挥重要作用,而基因组序列比对是其数据分析中最重要的任务之一,目标是找到 DNA 序列之间的相似性和样式.完成该任务需要使用计算密集型算法处理大量数据.但现有的硬件设施均存在各自的劣势,不适合直接用于序列比对:通用 CPU 在处理大量数据时效率低下;GPU 虽然提供优秀的性能和可编程性,但功耗较高;FPGA 具有良好的性能和功耗比,但编程难度较大.为了在性能、功耗、可编程性之间取得较好的平衡,Di Tucci 等人[19]提出了一种用于序列比对的领域专用架构(domain specific architecture,简称 DSA)SALSA.

SALSA 在设计上主要由一个流水线组成,共分为 4 个阶段:取码/解码、调度、加载/存储、计算,每个阶段都与解决特定任务的硬件单元相关联.

(1) 取码/解码阶段:从主处理器获取指令,并将其传递给调度器;

(2) 调度阶段:可以决定将指令发送至加载/存储阶段,或直接发送至计算阶段;

(3) 加载/存储阶段:直接连接内存控制器,执行单个或多个加载/存储请求;

(4) 计算阶段:与 I/O 完全分离,完成具体的计算操作.

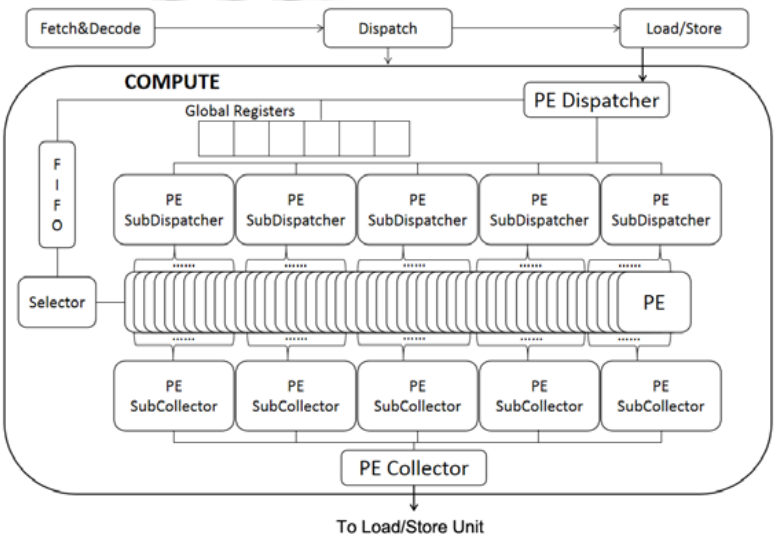


Fig.9 Top level architecture of SALSA[19]

图 9 SALSA 顶层架构设计[19]

SALSA 的顶层架构设计如图 9 所示.其中,计算单元由一个 PE 调度器(PE dispatcher)、一组 PE 子调度器(PE subdispatcher)、一个紧耦合的 PE 线性阵列、一组全局寄存器(global registers,可由所有处理单元以读模式访问)、一个 FIFO 缓冲区,和一组与 PE 采集器通信的 PE 子采集器(PE subcollector)组成.系统可采用自定义的 ALU 来加速计算的执行.具体地,SALSA 设计了 3 个特定于应用的 ALU 来执行序列比对,这些 ALU 用于执行 Smith-Waterman 算法、Needleman Wunsch 算法和 Smith-Waterman 算法.

5.2.2 场景分析

如第 4.1 节所述,RISC-V 系统的功能设计主要考虑的是工作流通路的构造问题,以及围绕该问题所衍生出的指令集选取、内存设计、通信、负载平衡等.在本例中,SALSA 确定了其工作流为对 DNA 序列数据的解码和比对运算过程,并根据其特点选定了以流水线形式的构造方案,分四个阶段依次完成了对指令的获取、调度、内存处理、计算的过程.

为了使 SALSA 具有可编程性和可扩展性,其指令建立在 ROCC 接口[25]之上.ROCC 接口使用了 RISC-V 规范中为非标准扩展预留的编码空间,可以使用 ROCC 指令格式来定义特定 ALU 上的操作.用户也可以利用 SALSA 定义新的算法和操作指令,将其适用范围扩展到更一般的领域中.

5.3 MAC:3D栈内存聚合单元

5.3.1 场景描述

在图形分析、数据挖掘、机器学习、数据驱动的科学计算等新兴的数据密集型应用场景中,对内存的访问呈现出越来越不规则的模式,使得传统的基于缓存的处理器体系结构效率低下.3D 栈存储设备,如混合存储立方体(hybrid memory cube,简称 HMC)和高带宽存储器(high bandwidth memory,简称 HBM)等,可以以较低的内存密度为代价,大幅增加带宽,对于内存受限的应用有很大吸引力.但是传统的内存接口与 3D 栈存储设备匹配性差,无法充分利用后者的带宽优势.为此,Wang 等人[146]提出了一种 3D 栈内存聚合单元——记忆体存取整合器(memory access coalescer,简称 MAC),以构造可扩展的多节点非统一的内存体系结构(non-uniform memory architecture,简称 NUMA).在该结构中,每个节点都是一个独立的处理器,并通过 MAC 直接连接到 3D 栈内存存储设备,实现近内存访问.MAC 聚合单元及 NUMA 体系结构的设计架构如图 10 所示.

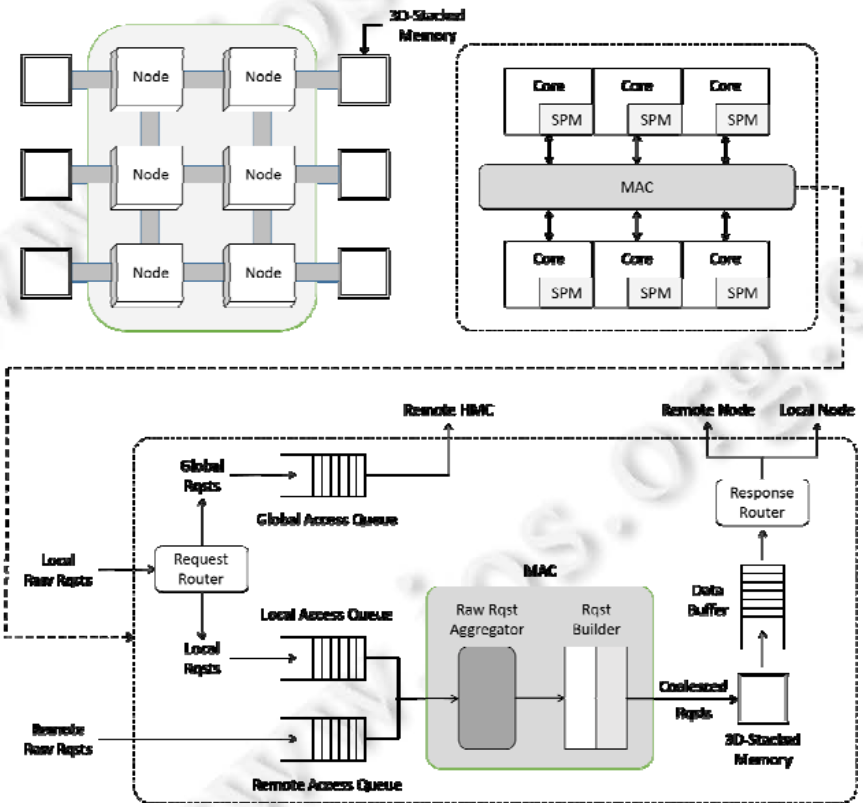


Fig.10 Architecture of MAC design[146]

图 10 MAC 设计架构[146]

在这个设计中,MAC 共包含两个组成部分:原始请求聚合器和请求构建器.原始请求聚合器负责根据请求类型和物理地址对原始请求进行分组,然后将落入同一行的原始请求合并,分派到请求构建器.请求构建器根据体系结构提供的可扩展内存操作量和延迟,确定聚合请求的数据包尺寸,并组装生成具体的 3D 栈内存请求,将其发送到 3D 栈内存.

在本地节点接收到来自本地 3D 栈内存的响应之后,会先将数据存储到缓冲区中;响应路由器对缓冲器中的数据进行分析,根据每个请求的目标(TID、事务 ID 等)将实际数据定向到其目的地.此时,若原始请求是远程的,数据会经互连路由送入相应的远程节点;否则,数据将直接发往本地节点的处理器核.

5.3.2 场景分析

对于 RISC-V 系统性能的优化,可以从处理器、内存、通信、能耗等方面入手,本例即是针对内存结构的优化.而在具体的设计和实施过程中,RISC-V 构成了 MAC 的工作基础,体现了在架构设计、资源管理、工具链等方面的优势.

MAC 基于 RISC-V 核心及 RV64IMAFDC 指令集架构实现,在处理器、内存、存储设备之间构造了新的数据通路,而充分发挥了设备的带宽优势,提升了数据吞吐率和处理效率.

实验验证的环境也采用 RISC-V 构建:首先利用 RISC-V spike 模拟器实现了一个内存跟踪器,可以捕获多处理器系统生成的内存操作;然后在该模拟器中实现了处理器核心的 SPM 部分,并扩展了 RISC-V ISA 及相应的交叉编译器,以实现对 SPM 功能(预取、写回等)的管理.实验系统采用 8 个 RISC-V 核心,每个核心有 1MB 的 SPM 和 8GB 的 HMC 设备.

实验结果表明,系统的合并效率(使用 MAC 的合并请求数占未使用 MAC 的原始请求数的百分比)平均可达到 52.86%,即,MAC 合并了一半以上的原始请求.

5.4 Notary安全批准方案

5.4.1 场景描述

在一些安全敏感的操作场景(如金融、系统管理等)中会需要用户在特定设备(如电脑、智能手机等)上进行批准.Anish Athalye 的团队[147]研究了批准的过程,利用 RISC-V 设计实现了一套软硬件结合的安全批准体系 Notary.Notary 旨在解决如下问题.

(1) 为批准过程提供强力安全保障.

(2) 在单一设备上支持多种批准,而不发生相互干扰和泄密.

Notary 假定远程攻击者可以完全控制用户的计算机,并且可以创建恶意代理,诱骗用户安装和运行,而恶意代理能够破坏其所在代理域的数字门抽象.但是远程攻击者无法与用户进行物理上的交互,用户也能够正确使用 Notary,只批准期望批准的操作.

基于上述威胁模型,Notary 在硬件层面被设计为一个带有小型显示屏和操作按钮的专用安全设备,形状类似于小型 U 盘;在软件层面,Notary 被划分为 3 个域:内核域、代理域、通信域,每个域都独占一块 SoC,并通过 UART 链路相互连接.图 11 显示了 Notary 的整体设计方案.

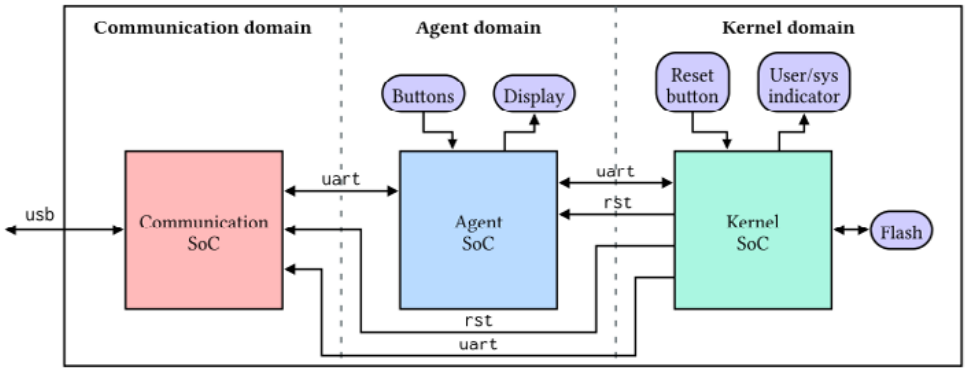


Fig.11 Overall design of Notary[147]

图 11 Notary 整体设计[147]

5.4.2 场景分析

Notary 使用 3 个独立的 SoC 构建构建其原型,并在 FPGA 上实例化.其中,使用基于 PicoRV32 CPU 的 RISC-V SoC 实现了代理域的设计,利用了 RISC-V 的特权机制对域进行管理.Notary 的代理域执行第三方提供的代理代码,是抵御恶意代理的第 1 道关口.它独占显示屏和操作按钮的访问权限,可以防止攻击者伪造虚假的屏幕输出,或者伪造虚假的按钮输入.

Notary 还需要实现多重代理之间的切换过程,为此采用了一种基于重置的交换技术:先重置 SoC,清除所有内部状态,再执行引导程序,令内核域从闪存中加载新的代理,并通过 UART 线路更新其他域的数据.这个过程中,基于 RISC-V 的工具简化了重置的过程,正确清除了所有的 SoC 内部状态,并显示出该技术应用到更复杂的 SoC 的可能性.

6 未来发展方向

RISC-V 是一种很有前景的指令集架构,拥有十分广阔的发展空间.本文认为,至少在以下一些方面,RISC-V 能够取得可以预见的进展.

6.1 硬件的新发展

随着社会需求的不断增长和相关技术的不断进步,会不断涌现出各种新类型硬件.如何适应新的工作环境,适配这些新的硬件,而继续保证 RISC-V 系统的功能完整性、性能优越性、安全稳定性,将成为随之而来的问题.现有的一些 RISC-V 异构硬件解决方案将在解决这一问题中发挥重要的作用;而在硬件方面取得新突破后,RISC-V 架构自身是否也会随之发生新的变化,同样值得关注.

特别地,技术的进步和新硬件的出现也可能导致新的攻击形式产生,使传统的系统安全防御方法失去作用.例如,量子计算机的出现为密码安全带来了新的威胁,需要后量子安全方案才能加以抑制.继续研究完善 RISC-V 指令集及其系统的安全机制和防御手段,将是相当重要的工作方向.

6.2 与新技术结合

RISC-V 自身也可以与新技术结合,在功能、性能、安全等领域实现新的突破.例如,随着神经科学和脑科学的不断发展,通过脑机接口技术,可以实现生命体对机器的直接操控或影响;RISC-V 便可以与此技术结合,在生命医疗、自动化控制等领域发挥更大的作用.目前已有研究团队在此方向上进行了尝试,如 Karageorgos 等人[148]设计的植入式脑机接口通用架构 HALO,已在促进医疗和脑研究方面收获了一定的经验.

7 结束语

本文对近年来 RISC-V 体系结构相关的研究进行了分析和总结.首先介绍了 RISC-V 指令集的有关概念和发展现状,总结了各基础指令集、扩展指令集的审批状态和 RISC-V 的几种权限模式.然后,对 RISC-V 系统所依托的硬件平台(处理器、模拟器等)进行了总结分析.接着,从功能实现、性能优化、安全策略这 3 个方面讨论了 RISC-V 系统的设计问题,并通过一些具体的应用场景,展示了 RISC-V 在促进特定领域应用研究方面所能发挥的作用.最后,我们展望了 RISC-V 架构研究的未来研究和发展方向.根据本文的总结内容,RISC-V 作为一种新兴的开放指令集架构,在计算机系统研究和产业发展的诸多领域都展现出了良好前景.期望通过我们的工作,能够给今后的研究者提供有益的参考,不断扩大 RISC-V 的应用范围和影响力,为相关技术的应用和发展做出更多贡献.

致谢 在此,我们向给予本文工作各项支持和宝贵建议的评审老师和同行们表示衷心的感谢.

References:

[1] Manavski SA. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In: Proc. of the 2007 IEEE Int'l Conf. on Signal Processing and Communications (ICSPC 2007). 2007. [doi: 10.1109/ICSPC.2007.4728256]

[2] Costan V, Lebedev I, Devadas S. Sanctum: Minimal hardware extensions for strong software isolation. In: Proc. of the 5th USENIX Security Symp. 2016. 857−874. Austin, TX. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[3] Chen YJ, Luo T, Liu SL, *et al*. DaDianNao: A machine-learning supercomputer. In: Proc. of the 47th Annual IEEE/ACM Int'l Symp. on Microarchitecture. 2014. 609−622. [doi: 10.1109/MICRO.2014.58]

[4] Corbató FJ, Vyssotsky VA. Introduction and overview of the multics system. In: Proc of the Fall Joint Computer Conf., Part I. New York: Association for Computing Machinery, 1965. 185−196. [doi: 10.1145/1463891.1463912]

[5] UNIX PROGRAMMER'S MANUAL. Vol. 2B.7th ed., Bell Telephone Laboratories, Incorporated. 1979. https://web.archive.org/web/ 20040920172248/http://cm.bell-labs.com/7thEdMan/v7vol2b.pdf

[6] Budhiraja S, Mehrotra R. Mach: A new kernel foundation For UNIX development. Int'l Journal of Innovative Research in Technology (IJIRT). 2014, 7.

[7] Waterman A, Lee Y, Patterson DA. The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA. Vol. 1. 2011.

[8] Rodgers S, Uhlig RA. X86: Approaching 40 and still going strong. 2017. https://newsroom.intel.com/editorials/x86-approaching-40-still-going-strong/

[9] History of RISC-V. 2021. https://riscv.org/about/history

[10] Arm® Architecture Reference Manual. Armv8, for A-profile architecture. ARM Developer. 2021. https://developer.arm.com/documentation/ddi0487/latest

[11] Waterman A, Asanovic K. The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. Vol. 1. SiFive Inc., 2021.

[12] Waterman A, Asanovic K, Hauser J. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture. SiFive Inc., 2021.

[13] Ted M. RISC-V: High performance embedded SweRV™ core microarchitecture, performance and CHIPS Alliance. Western Digital Corporation. 2019. https://riscv.org/wp-content/uploads/2019/04/RISC-V_SweRV_Roadshow-.pdf

[14] Chen C, Xiang XY, Liu C, *et al*. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension: Industrial product. In: Proc. of the 47th ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). 2020. 52−64. [doi: 10.1109/ISCA45697.2020.00016]

[15] Koch D, Dao N, Healy B, *et al*. FABulous: An embedded FPGA framework. In: Proc. of the 2021 ACM/SIGDA Int'l Symp. on Field-programmable Gate Arrays. 2021. 45−56. [doi: 10.1145/3431920.3439302]

[16] XiangShan-doc. UCAS & ICT, PCL. 2021. https://github.com/OpenXiangShan/XiangShan-doc

[17] Feng EH, Lu X, Du D, *et al*. Scalable memory protection in the PENGLAI enclave. In: Proc. of the 15th USENIX Symp. on Operating Systems Design and Implementation (OSDI 2021). 2021. 275−294. https://ipads.se.sjtu.edu.cn/zh/publications/FengOSDI21-preprint.pdf

[18] Kadomoto J, Irie H, Sakai S. Design of shape-changeable chiplet-based computers using an inductively coupled wireless bus interface. In: Proc. of the 38th IEEE Int'l Conf. on Computer Design (ICCD). 2020. 589−596. [doi: 10.1109/ICCD50377.2020. 00103]

[19] Tucci LD, Baghdadi R, Amarasinghe S, *et al*. SALSA: A domain specific architecture for sequence alignment. In: Proc. of the 2020 IEEE Int'l Parallel and Distributed Processing Symp. Workshops (IPDPSW). 2020. 147−150. [doi: 10.1109/IPDPSW50202.2020. 00033]

[20] Kito C, Jessica C, Palmer D, Andrew W, Jim W. RISC-V ELF psABI specification. 2021. https://github.com/riscv-non-isa/riscv-elf-psabi-doc/releases

[21] Syntacore. SCR1 external architecture specification. 2021. https://github.com/syntacore/scr1/blob/master/docs/scr1_eas.pdf

[22] IEEE Standard for Floating-Point Arithmetic. In: IEEE Std 754-2008. 2008. 1−70. [doi: 10.1109/IEEESTD.2008.4610935]

[23] Owens S, Sarkar S, Sewell P. A better x86 memory model: x86-TSO (extended version). Technical Report, University of Cambridge Computer Laboratory, 2009. [doi: 10.48456/tr-745]

[24] Multanen J, Hepda K, Jaaskelainen P. Programmable dictionary code compression for instruction stream energy efficiency. In: Proc. of the 38th IEEE Int'l Conf. on Computer Design (ICCD). 2020. 356−363. [doi: 10.1109/ICCD50377.2020.00066]

[25] Asanovic K, Avizienis R, Bachrach J, *et al*. The rocket chip generator. Technical Report, Berkeley: EECS Department, University of California, 2016. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf

[26] Zaruba F, Benini L. The cost of application-class processing: Energy and performance analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V core in 22-nm FDSOI technology. IEEE Trans. on Very Large Scale Integration (VLSI) Systems. 2019,27(11): 2629−2640. [doi: 10.1109/TVLSI.2019.2926114]

[27] Lowe-Power J, Nitta C. The Davis in-order (DINO) CPU: A teaching-focused RISC-V CPU design. In: Proc. of the Workshop on Computer Architecture Education. New York: Association for Computing Machinery. 2019. 1–8. [doi: 10.1145/3338698.3338892]
[28] OSCPU. NutShell. 2021. https://github.com/OSCPU/NutShell
[29] riscvarchieve. RISC-V Cores and SoC Overview. 2021. https://github.com/riscvarchive/riscv-cores-list
[30] SiFive. Freedom. 2021. https://github.com/sifive/freedom
[31] Celio C, Patterson DA, Asanovic K. The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor. Technical Report, Berkeley: University of California, 2015. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html
[32] SpinalHDL. VexRiscv. 2021. https://github.com/SpinalHDL/VexRiscv
[33] SCR5 efficient application core (RV32 or RV64). Syntacore. 2021. https://syntacore.com/page/products/processor-ip/scr5
[34] MIT CSAIL's Computation Structures Group. RiscyOO: RISC-V Out-of-Order Processors. 2021. https://github.com/csail-csg/riscy-OOO
[35] darklife. DarkRISCV. 2021. https://github.com/darklife/darkriscv
[36] stnolting. The NEORV32 RISC-V Processor. 2021. https://github.com/stnolting/neorv32
[37] Codasip RISC-V Processors. 2021. https://codasip.com/products/codasip-risc-v-processors/
[38] AndesCore™ AX45 Overview. 2021. http://www.andestech.com/en/products-solutions/andescore-processors/riscv-ax45/
[39] NOEL-V Processor. 2021. https://www.gaisler.com/index.php/products/processors/noel-v
[40] Hummingbirdv2 E203 Core and SoC. 2021, Nuclei System Technology. https://github.com/riscv-mcu/e203_hbirdv2
[41] SHAKTI E-CLASS, SHAKTI. 2021. https://gitlab.com/shaktiproject/cores/e-class/blob/master/README.md
[42] lowRISC. Ibex RISC-V Core. 2021. https://github.com/lowRISC/ibex
[43] Wolf C. PicoRV32-A Size-Optimized RISC-V CPU. 2020. https://github.com/cliffordwolf/picorv32
[44] E906. 2021. https://occ.t-head.cn/vendor/cpu/index?id=3806463049662468096
[45] CVA6 RISC-V CPU. OpenHW Group. 2021. https://github.com/openhwgroup/cva6
[46] EH2 SweRV RISC-V CoreTM 1.4 from Western Digital. Western Digital Corporation. 2021. https://github.com/chipsalliance/Cores-SweRV-EH2
[47] liangkangnan. tinyriscv. 2020. https://github.com/liangkangnan/tinyriscv
[48] Taiga. 2020. https://gitlab.com/sfu-rcl/Taiga
[49] Chrisóstomo JVR. maestro. 2020. https://github.com/Artoriuz/maestro
[50] Sonal P. Kronos RISC-V. 2021. https://github.com/SonalPinto/kronos
[51] XiangShan. UCAS & ICT, CAS. 2021. https://github.com/OpenXiangShan/XiangShan
[52] Bao YG. Answer: The 1st RISC-V China Summit will be held in Shanghai from June 21, 2021. Is there anything worthy of attention? 2021 (in Chinese). https://www.zhihu.com/question/466393646/answer/1955410750
[53] riscvarchieve. RISC-V software ecosystem overview. 2021. https://github.com/riscvarchive/riscv-software-list
[54] Li JF, Xu Q, Li YY, *et al*. Efficient return address verification based on dislocated stack. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2020,39(11):3398-3407. [doi: 10.1109/TCAD.2020.3012645]
[55] Imperas RISC-V riscvOVPsim reference simulator and architectural validation tests. 2021. https://www.ovpworld.org/riscvOVPsimPlus/
[56] DBT-RISE-RISCV. 2021. https://github.com/Minres/DBT-RISE-RISCV
[57] Karandikar S, Mao H, Kim D, *et al*. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In: Proc. of the 45th ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). 2018. 29–42. [doi: 10.1109/ISCA.2018.00014]
[58] Roelke A. gem5. University of Virginia. 2021. https://gem5.googlesource.com/public/gem5/
[59] Imperas. OVP RISC-V Solutions. 2021. https://www.ovpworld.org/info_riscv
[60] Sebastian M. jor1k. 2021. https://github.com/s-macke/jor1k/
[61] Andrés C. Jupiter RISC-V Assembler & Runtime Simulator. 2019. https://github.com/andrescv/Jupiter
[62] Binghamton University Computer Architecture and Power-Aware Systems (CAPS) Research Group. MARSS-RISCV: Micro-architectural system simulator for RISC-V. 2020. https://github.com/bucaps/marss-riscv
[63] Official QEMU source repository. 2021. https://git.qemu.org/git/qemu.git/
[64] Benjamin L. RARS—RISC-V Assembler and Runtime Simulator. 2021. https://github.com/thethirdone/rars
[65] Renode. Antmicro. 2021. https://github.com/renode/renode

[66] Morten BP. Ripes. 2021. https://github.com/mortbopet/Ripes
[67] Herdt V. RISC-V based virtual prototype (VP). University of Bremen, AGRA, 2021. https://github.com/agra-uni-bremen/riscv-vp
[68] Bellard F. TinyEMU. 2019. https://bellard.org/tinyemu/
[69] Waterman A, Lee Y. Spike RISC-V ISA simulator. SiFive. 2021. https://github.com/riscv-software-src/riscv-isa-sim
[70] Rahmeh J. SweRV-ISS. Western Digital. 2021. https://github.com/chipsalliance/SweRV-ISS
[71] VLAB. 2020. https://vlabworks.com/vlab/
[72] Giorgi R, Mariotti G. WebRISC-V: A Web-based education-oriented RISC-V pipeline simulation environment. In: Proc. of the Workshop on Computer Architecture Education. New York: Association for Computing Machinery. 2019. [doi: 10.1145/3338698.3338894]
[73] Post-Quantum Crypto IP: Software Support. https://pqsoc.com/software/
[74] Aoyagi T. riscv-rust. 2021. https://github.com/takahirox/riscv-rust
[75] Apollo L. terminus. 2021. https://github.com/shady831213/terminus
[76] Victor MdMC. Vulcan. 2021. https://github.com/vmmc2/Vulcan
[77] Aidan D. RISC-V virtual machine. 2020. https://github.com/bit-hack/riscv-vm
[78] Guillaume S. emulsiV. 2021. https://github.com/Guillaume-Savaton-ESEO/emulsiV
[79] Neumann JV. First draft of a report on the EDVAC. IEEE Annals of the History of Computing. 1993,15(4):27−75. [doi: 10.1109/85.238389]
[80] Wikipedia. Harvard architecture. 2021. https://en.wikipedia.org/wiki/Harvard_architecture
[81] Wikipedia. Modified Harvard architecture. 2021. https://en.wikipedia.org/wiki/Modified_Harvard_architecture
[82] Vajda A. Multi-core and many-core processor architectures. In: Programming Many-core Chips. Boston: Springer US, 2011. 9–43. [doi: 10.1007/978-1-4419-9739-5_2]
[83] Hwang EO. Digital Logic and Microprocessor Design with VHDL. Cengage Learning, 2005.
[84] Burroughs Corporation. B5500 ESPOL Reference Manual. 1967. http://bitsavers.org/pdf/burroughs/B5000_5500_5700/1032638_B5500_ESPOL_RefManOct67.pdf
[85] Kernighan BW, Ritchie DM. The C Programming Language. Englewood Cliffs, NJ: Prentice Hall, 1978.
[86] Programming languages—C++: ISO/IEC 14882:2020. 2020. https://www.iso.org/standard/79358.html
[87] Klabnik S, Nichols C. The Rust Programming Language. 2018. https://doc.rust-lang.org/book/
[88] Lin H, Chen P, Hwang YS, *et al*. Devise rust compiler optimizations on RISC-V architectures with SIMD instructions. In: Proc. of the 48th Int'l Conf. on Parallel Processing: Workshops. New York: Association for Computing Machinery. 2019. 1–7. [doi: 10.1145/3339186.3339193]
[89] Zhang SZ, Wright A, Bourgeat T, *et al*. Composable building blocks to open up processor design. In: Proc. of the 51st Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). 2019,39(3):47−55. [doi: 10.1109/MM.2019.2910012]
[90] Liu G, Primmer J, Zhang ZR. Rapid generation of high-quality RISC-V processors from functional instruction set specifications. In: Proc. of the 56th ACM/IEEE Design Automation Conf. (DAC). 2019. 1−6. https://ieeexplore.ieee.org/abstract/document/8806953
[91] Tamimi S, Ebrahimi Z, Khaleghi B, *et al*. An efficient SRAM-based reconfigurable architecture for embedded processors. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2019,38(3):466–479. [doi: 10.1109/TCAD.2018.2812118]
[92] Huang BY, Zhang HC, Subramanyan P, *et al*. Instruction-level abstraction (ILA): A uniform specification for system-on-chip (SoC) verification. ACM Trans. on Design Automation of Electronic Systems, 2019,24(1):1-24. [doi: 10.1145/ 3282444]
[93] Balkind J, Lim K, Schaffner M, *et al*. BYOC: A "Bring Your Own Core" framework for heterogeneous-ISA research. In: Proc. of the 25th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: Association for Computing Machinery, 2020. 699−714. [doi: 10. 1145/3373376.3378479]
[94] SpinalHDL. 2021. https://github.com/SpinalHDL/SpinalHDL
[95] Tayler MB. Basejump STL: Systemverilog needs a standard template library for hardware design. In: Proc. of the 55th Annual Design Automation Conf. New York: Association for Computing Machinery, 2018. 1−6. [doi: 10.1145/3195970.3199848]
[96] Parkhurst J, Darringer J, Grundmann B. From single core to multi-core: Preparing for a new exponential. In: Proc. of the 2006 IEEE/ACM Int'l Conf. on Computer Aided Design. 2006. 67−72. [doi: 10.1109/ICCAD.2006.320067]
[97] Monerkar YA, Lustig D, Martonosi M, *et al*. RTLCheck: Verifying the memory consistency of RTL designs. In: Proc. of the 50th Annual IEEE/ACM Int'l Symp. on Microarchitecture. New York: Association for Computing Machinery. 2017. 463−476. [doi: 10.1145/3123939.3124536]

[98] Kurth A, Riedel S, Zaruba F, *et al*. ATUNs: Modular and scalable support for atomic operations in a shared memory multiprocessor. In: Proc. of the 57th ACM/IEEE Design Automation Conf. (DAC). 2020. 1−6. [doi: 10.1109/DAC18072. 2020.9218661]

[99] Pulte C, Pichon-Pharabod J, Kang J, *et al*. Promising-ARM/RISC-V: A simpler and faster operational concurrency model. In: Proc. of the 40th ACM SIGPLAN Conf. on Programming Language Design and Implementation. New York: Association for Computing Machinery. 2019. 1−15. [doi: 10.1145/3314221.3314624]

[100] Enokido T, Barolli L, Takizawa M. Network-Based Information Systems. Springer, 2007. [doi: 10.1007/978-3-540-74573-0]

[101] Glaser F, Tagliavini G, Rossi D, *et al*. Energy-efficient hardware-accelerated synchronization for shared-L1-memory multiprocessor clusters. IEEE Trans. on Parallel and Distributed Systems. 2021,32(3):633-648. [doi: 10.1109/TPDS.2020. 3028691]

[102] Shi RB, Liu JJ, So HKH, *et al*. E-LSTM: Efficient inference of sparse LSTM on embedded heterogeneous system. In: Proc. of the 56th Annual Design Automation Conf. New York: Association for Computing Machinery, 2019. 1−6. [doi: 10.1145/3316781. 3317813]

[103] Armstrong A, Bauereiss T, Campbell B, *et al*. ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS. In: Proc. of the 8th Int'l Workshop on Hardware and Architectural Support for Security and Privacy. New York: Association for Computing Machinery, 2019. 1−31. [doi: 10.1145/3290384]

[104] Deng SW, Gümusoglu D, Xiong WJ, *et al*. SecChisel framework for security verification of secure processor architectures. In: Proc. of the 8th Int'l Workshop on Hardware and Architectural Support for Security and Privacy. New York: Association for Computing Machinery, 2019. 1−8. [doi: 10.1145/3337167.3337174]

[105] Nelson L, Bornholt J, Gu RH, *et al*. Scaling symbolic evaluation for automated verification of systems code with Serval. In: Proc. of the 27th ACM Symp. on Operating Systems Principles. New York: Association for Computing Machinery, 2019. 225−242. [doi: 10.1145/3341301.3359641]

[106] Khamis M, El-Ashry S, Shalaby A, *et al*. A configurable RISC-V for NoC-based MPSoCs: A framework for hardware emulation. In: Proc. of the 11th Int'l Workshop on Network on Chip Architectures (NoCArc). 2018. 1−6. [doi: 10. 1109/NOCARC.2018.8541158]

[107] Dessouky G, Gens D, Haney P, *et al*. HardFails: Insights into software-exploitable hardware bugs. In: Proc. of the 28th USENIX Conf. on Security Symposium. USENIX Association, 2019. 213−230. https://dl.acm.org/doi/abs/10.5555/3361338.3361354

[108] Nelson L, Geffen JV, Torlak E, *et al*. Specification and verification in the field: Applying formal methods to BPF just-in-time compilers in the Linux kernel. In: Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation. USENIX Association, 2020. 41−61. https://www.usenix.org/conference/osdi20/presentation/nelson

[109] Yu ZH, Huang B, Ma JY, *et al*. Labeled RISC-V: A new perspective on software-defined architecture. In: Proc. of the 1st Workshop on Computer Architecture Research with RISC-V (CARRV 2017) Co-located with MICRO. 2017.

[110] Yu ZH, Liu ZG, Li YW, *et al*. Practice of chip agile development: Labeled RISC-V. Journal of Computer Research and Development, 2019,56(1):35−48 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2019. 20180771]

[111] Schuiki F, Zaruba F, Hoefler T, *et al*. Stream semantic registers: A lightweight RISC-V ISA extension achieving full compute utilization in single-issue cores. IEEE Trans. on Computers, 2021,70(2):212−227. [doi: 10.1109/TC.2020.2987314]

[112] Dogan H, Ahmad M, Kahne B, *et al*. Accelerating synchronization using moving compute to data model at 1,000-core multicore scale. ACM Trans. on Architecture and Code Optimization, 2019,16(1):1−27. [doi: 10.1145/3300208]

[113] Leidel JD, Wang X, Conlon F, *et al*. xBGAS: Toward a RISC-V ISA extension for global, scalable shared memory. In: Proc. of the Workshop on Memory Centric High Performance Computing. New York: Association for Computing Machinery, 2018. 22−26. [doi: 10.1145/3286475.3286478]

[114] Williams B, Wang X, Leidel JD, *et al*. Collective communication for the RISC-V xBGAS ISA extension. In: Proc. of the 48th Int'l Conf. on Parallel Processing: Workshops. 2019. 1−10. [doi: 10.1145/3339186.3339196]

[115] Wang X, Williams B, Leidel JD, *et al*. Remote atomic extension (RAE) for scalable high performance computing. In: Proc. of the 57th ACM/IEEE Design Automation Conf. (DAC). 2020. 1−6. [doi: 10.1109/DAC18072.2020.9218589]

[116] Morais L, Alvarez C, Bosch J, *et al*. Adding tightly-integrated task scheduling Acceleration to a RISC-V multi-core processor. In: Proc. of the 52nd Annual IEEE/ACM Int'l Symp. on Microarchitecture. 2019. 861−872. [doi: 10. 1145/3352460.3358271]

[117] Zhou Y, Ren HX, Zhang YQ, *et al*. PRIMAL: Power inference using machine learning. In: Proc. of the 56th ACM/IEEE Design Automation Conf. (DAC). 2019. 1−6. https://ieeexplore.ieee.org/document/8806775

[118] Bambini G, Balas R, Conficoni C, *et al*. An open-source scalable thermal and power controller for HPC processors. Proc. of the 38th IEEE Int'l Conf. on Computer Design (ICCD). 2020. 364−367.

[119] Wright JC, Schmidt C, Keller B, *et al*. A dual-core RISC-V vector processor with on-chip fine-grain power management in 28-nm FD-SOI. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2020(28):2721−2725. [doi: 10.1109/TVLSI.2020.3030243]

[120] Albartus N, Nasenberg C, Stolz F, *et al*. On the design and misuse of microcoded (embedded) processors — A cautionary note. Proc. of the 30th USENIX Security Symp. USENIX Association, 2021. 267−284. https://www.usenix.org/conference/usenixsecurity21/presentation/albartus

[121] Lee D, Kohlbrenner D, Shinde S, *et al*. Keystone: An open framework for architecting trusted execution environments. In: Proc. of the 15th European Conf. on Computer Systems. New York: Association for Computing Machinery, 2020. 1−16. [doi: 10.1145/3342195.3387532]

[122] Burow N, Carr SA, Nash J, *et al*. Control-flow integrity: Precision, security, and performance. ACM Computing Surveys, 2017(50): 1−33. [doi: 10.1145/3054924]

[123] One A. Smashing the stack for fun and Profit. 1996. https://web.archive.org/web/20100514141355/http://www.phrack.com/issues.html?issue=49&id=14&mode=txt

[124] Cowan C, Pu C, Maier D, *et al*. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In: Proc. of the 7th Conf. on USENIX Security Symp. USENIX Association, 1998(7):5. https://dl.acm.org/doi/10.5555/1267549.1267554

[125] Abadi M, Budiu M, Erlingsson U, *et al*. Control-flow integrity: Principles, implementations, and applications. Proc. of the 2005 ACM Conf. on Computer and Communications Security (CCS 2005). New York: Association for Computing Machinery, 2005, 13(1):1−40. [doi: 10.1145/1609956.1609960]

[126] De A, Basu A, Ghosh S, *et al*. Hardware assisted buffer protection mechanisms for embedded RISC-V. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2020. 4453−4465. [doi: 10.1109/TCAD.2020.2984407]

[127] Wilander J, Kamkar M. A comparison of publicly available tools for dynamic buffer overflow prevention. In: Proc. of the Network and Distributed System Security Symp., NDSS 2003. 2003(3):149−162.

[128] Ferraiuolo A, Zhao M, Myers AC, *et al*. HyperFlow: A processor architecture for nonmalleable, timing-safe information flow security. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. 2018. 1583−1600. [doi: 10.1145/3243734.3243743]

[129] Gallagher M, Biernacki L, Chen SB, *et al*. Morpheus: A vulnerability-tolerant secure architecture based on ensembles of moving target defenses with churn. In: Proc. of the 24th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: Association for Computing Machinery, 2019. 469−484. [doi: 10.1145/3297858.3304037]

[130] Bresch C, Hely D, Lysecky R, *et al*. TrustFlow-X: A practical framework for fine-grained control-flow integrity in critical systems. ACM Trans. on Embedded Computing Systems, 2020,19(5):1−26. [doi: 10.1145/3398327]

[131] Hu H, Shinde S, Adrian S, *et al*. Data-oriented programming: on the expressiveness of non-control data attacks. In: Proc. of the 2016 IEEE Symp. on Security and Privacy (SP). 2016. 969−986. [doi: 10.1109/SP.2016.62]

[132] Nyman T, Dessouky G, Zeitouni S, *et al*. HardScope: Hardening embedded systems against data-oriented attacks. In: Proc. of th 56th ACM/IEEE Design Automation Conf. (DAC). 2019. 1−6.

[133] Menon A, Murugan S, Rebeiro C, *et al*. Shakti-T : A RISC-V processor with light weight security extensions. In: Proc. of the Hardware and Architectural Support for Security and Privacy. New York: Association for Computing Machinery, 2017. 1−8. [doi: 10.1145/3092627.3092629]

[134] Wong MM, Haj-Yahya J, Chattopadhyay A. SMARTS: Secure memory assurance of RISC-V trusted SoC. In: Proc. of the 7th Int'l Workshop on Hardware and Architectural Support for Security and Privacy. New York: Association for Computing Machinery, 2018. 1−8. [doi: 10.1145/3214292.3214298]

[135] Kokologiannakis M, Vafeiadis V. HMC: Model checking for hardware memory models. In: Proc. of the 25th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: Association for Computing Machinery, 2020. 1157−1171. [doi: 10.1145/3373376.3378480]

[136] Joy PG, Prabhu M, Shanmugalakshmi R. Side channel attack-survey. Int'l Journal of Advanced Scientific Research and Review, 2011,1(4):54−57.

[137] Guo DX, Chen KY, Zhang Y, *et al*. A survey of side-channel attack and security assessment for cryptographic equipment. Proc. of the 7th Int'l Conf. on Computer Engineering and Networks (CENet2017). 2017(299). [doi: 10.22323/1.299.0042]

[138] Saxena S, Sanyal G, Manu. Cache based side channel attack: A Survey. In: Proc. of the Int'l Conf. on Advances in Computing, Communication Control and Networking (ICACCCN). 2018. 278−284. [doi: 10.1109/ICACCCN.2018.8748811]

[139] Devi M, Majumder A. Side-channel attack in Internet of Things: A survey. In: Applications of Internet of Things. Singapore : Springer Singapore, 2021. 213−222. [doi: 10.1007/978-981-15-6198-6_20]

[140] Bulck JV, Oswald D, Marin E, *et al*. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. New York: Association for Computing Machinery, 2019. 1741−1758. [doi: 10.1145/3319535.3363206]

[141] Spruyt A, Milburn A, Chmielewski Ł. Fault injection as an oscilloscope: Fault correlation Analysis. IACR Trans. on Cryptographic Hardware and Embedded Systems, 2021(1):192−216. [doi: 10.46586/tches.v2021.i1.192-216]

[142] Ahmadi MM, Khalid F, Shafique M. Side-channel attacks on RISC-V processors: Current progress, challenges, and opportunities. In: Proc. of the CYBER 2020: The 5th Int'l Conf. on Cyber-Technologies and Cyber-systems. 2020. 18−23.

[143] Mulder ED, Gummalla S, Hutter M. Protecting RISC-V against side-channel attacks. In: Proc. of the 56th Annual Design Automation Conf. 2019 (DAC'19). 2019. 1−4. [doi: 10.1145/3316781.3323485]

[144] Deng SW, Xiong WJ, Szefer J. Secure TLBs. In: Proc. of the 46th ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). 2019. 346−359.

[145] Sehatbakhsh N, Yilmaz BB, Zajic A, *et al*. EMSim: A microarchitecture-level simulation tool for modeling electromagnetic side-channel signals. In: Proc. of the 2020 IEEE Int'l Symp. on High Performance Computer Architecture (HPCA). 2020. 71−85. [doi: 10.1109/HPCA47549.2020.00016]

[146] Wang X, Tumeo A, Leidel JD, *et al*. MAC: Memory access coalescer for 3D-stacked memory. In: Proc. of the 48th Int'l Conf. on Parallel Processing (ICPP). 2019. 1−10. [doi: 10.1145/3337821.3337867]

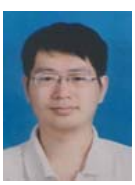
[147] Athalye A, Belay A, Kaashoek MF, *et al*. Notary: A device for secure transaction approval. In: Proc. of the 27th ACM Symp. on Operating Systems Principles. New York: Association for Computing Machinery, 2019. 97−113. [doi: 10.1145/3341301. 3359661]

[148] Karageorgos I, Sriram K, Veselý J, *et al*. Hardware-software co-design for brain-computer interfaces. In: Proc. of the 47th ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). 2020. 391−404. [doi: 10.1109/ ISCA45697.2020.00041]

附中文参考文献:

[52] 包云岗.回答:首届 RISC-V 中国峰会 2021 年 6 月 21 日起在上海举行,有什么值得关注的地方?2021. https://www.zhihu.com/question/466393646/answer/1955410750

[110] 余子濠,刘志刚,李一苇,等.芯片敏捷开发实践:标签化 RISC-V.计算机研究与发展,2019,56(1):35−48. https://crad.ict.ac.cn/CN/10.7544/issn1000-1239.2019.20180771 [doi: 10.7544/issn1000-1239.2019.20180771]



刘畅(1991－),男,博士,主要研究领域为操作系统,系统安全.

吴敬征(1982－),男,博士,博士生导师,CCF专业会员,主要研究领域为系统安全,漏洞挖掘,操作系统.

武延军(1979－),男,博士,博士生导师,CCF高级会员,主要研究领域为操作系统,系统安全.

赵琛(1967－),男,博士,博士生导师,CCF高级会员,主要研究领域为编译技术,操作系统,网络软件.