Embedded Technology and Intelligent Systems 嵌入式技术与智能系统,2024, 1(1), 23-30
Published Online August 2024 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2024.11003

Hans汉斯

移植Cortex-M程序到RV32中的问题

林金龙

北京大学软件与微电子学院，北京

收稿日期：2024年7月4日；录用日期：2024年7月25日；发布日期：2024年8月5日

摘要

具有开源、简单和灵活等特点，RISC-V架构受到业界广泛关注。近年来，市场上相继出现了多款RISC-V架构微处理器，32位RISC-V架构MCU正逐步进入Cortex-M MCU应用领域。本文针对将应用程序从RV32移植到Cortex-M的需求，分析RV32与Cortex-M结构、编程模型和过程调用规范等方面的不同之处，提出程序移植过程中遇到的问题，提出方法和建议，并进行相关性能分析和比较。

关键词

RISC-V，RV32，Cortex-M，程序移植

Problems in ProgramPorting from Cortex-M to RV32 Programs

JinlongLin

SchoolofSoftwareandMicroelectronics, PekingUniversity, Beijing

Received: Jul. 4th, 2024; accepted: Jul.25th, 2024; published: Aug.5th, 2024

Abstract

As a simplicityand flexibilityopen source microprocessor architecture, RISC-V has received widespread attention in the industry. In recent years, a lot of RISC-V microprocessors have been emerging in the market. The MCUs with the core of RV32 core are gradually entering into the application fields of Cortex-M MCUs. This article introduces the differences between RV32 and Cortex-M in terms of structure, programmer's model, and procedure call convention, discusses the problems during the program porting processfromCortex-MtoRV32, proposes some suggestions, and conducts relevant performance comparison.

Keywords

RISC-V,RV32,Cortex-M,Program Porting

文章引用:林金龙.移植 Cortex-M 程序到 RV32 中的问题[J].嵌入式技术与智能系统,2024,1(1):23-30.
DOI: 10.12677/etis.2024.11003

1. 引言

开源架构处理器 RISC-V[1]，在嵌入式系统中得到了越来越多的应用，近年多家处理器厂商发布了RV32 架构 MCU。2019 年 4 月，SiFive[2]发布了 Freedom E310；2020 年 2 月，兆易创新[3]发布 RV32 MCU GD32VF103；沁恒微电子[4]发布 CH32V、CH32X、CH32L 系列 RV32 MCU；先辑半导体[5]发布 HPM5XXX 和 HPM6XXX 系列 RV32 单核和多核 MCU；瑞萨电子[6]发布 RV32 汽车 MCU RH850/U2B 和通用 MCU R9A02G021 等。

随着 RISC-V MCU 的快速发展，其软件生态正逐步完善和丰富。Embedded Studio[7]、IAR[8]等主流商业 IDE 以及开源 IDE eclipse [9]等嵌入式软件开发环境支持 RISC-V MCU。一些主流操作系统已经支持 RISC-V 架构[10]。QEMU RISC-V 虚拟化平台[11]支持多种 RISC-V 处理器仿真；FreeRTOS 发布支持 RISC-V MCU 版本[12]；郃阳等[13]基于 QEMU RISC-V 实现 OpenHarmony 移植和优化；Nicholas Gordon 等[14]将 Kitten Lightweight Kernel 操作系统移植到 RISC-V；Luming Zhang[15]移植并优化 RISC-V UFEI Boot；Robert Balas 等[16]分析 RV32IMC 结构特点并提出程序方法。

第二届滴水湖中国 RISC-V 产业论坛现场调查[17]结果表明，RISC-V 架构处理器已经成功应用于无线连接芯片、工业控制芯片，网络通信芯片和边缘计算芯片等场景。目前，在移动通信、物联网和工业控制等嵌入式应用领域，ARM 架构处理器占市场主导地位[18]。在一些领域和应用场景，RISC-V 将对 ARM 的市场地位构成挑战。RV32 MCU 正争夺 ARM Cortex-M MCU 的市场份额。将 Cortex-M MCU 应用程序移植到 RV32 MCU，充分利用 Cortex-M MCU 的成熟生态，将有利于 RISC-V 的发展和推广。由于 RV32 和 Cortex-M 的结构和编程模式存在差异，尽管使用程序开发工具能够将源程序的编译、汇编和链接生成 RV32 执行程序，但在程序移植过程中仍然会遇到一些问题。

本文根据 RV32 与 Cortex-M 在结构、编程模型和过程调用规范等方面的不同点，分析应用程序从 RV32 移植到 Cortex-M 过程中遇到的问题，提出解决方法和建议，并进行相关性能分析和比较。

本文第一节简介 RISC-V 发展状况，讨论 RV32 的应用前景；第二节比较 RV32 与 Cortex-M 异常处理器机制，说明移植中断处理程序面临的问题；第三节对比 RV32 与 Cortex-M 指令架构，分析 RV32 指令模块组合对程序性能的影响；第四节讨论 RISC-V 与 ARM 处理器程序过程调用规范的差别，分析其对程序移植影响；第五节对论文工作进行总结。

2. 中断处理

在 ARM 架构处理器手册[19]和 RISC-V 架构处理器使用手册[20]中，用编程模式(Programmer’s model)表示处理器架构中涉及程序开发的内容。编程模式通常包括处理器支持的数据类型，通用和特殊功能寄存器、异常和中断响应机制和指令集等，异常和中断处理处理器基本功能。

通常将由外部信号引起的异常称为中断。中断处理是 MCU 应用系统的关键功能之一。中断处理功能包括两个部分，中断响应机制和中断服务程序(Interrupt Service Routine)。中断响应机制由处理器硬件结构决定，中断服务程序则与中断响应机制相关。

2.1. 中断响应机制

表 1 列出了 Cortex-M 和 RV32 中断响应机制的差异。将应用程序从 Cortex-M 移植到 RV32 时，需要修改中断处理功能相关程序。

Table 1. The exception handling of RV32 and Cortex-M
表 1.RV32 和 Cortex-M 异常管理

	Cortex-M	RV32
中断响应	向量	向量和非向量
上下文保存	自动	手动
C	可重定位(VTOR)	设置 mtvec
控制器	NVIC	定制

Cortex-M 仅支持向量中断响应，中断向量指向对应的中断服务程序入口，处理器启动后通过寄存器 VTOR 冲重定位中断向量表。RV32 支持向量和非向量两种中断响应方式，处理器器复位时缺省为非向量响应形式，中断响应时指向程序空间的起始地址，通常为 0x00。处理器启动后，通过设置机器模式异常向量基址寄存器 mtvec 设置中断响应模式和异常向量入口地址。

为了保证移植前后功能的一致性，将应用程序移植到 RV32 后仍保证向量中断响应方式。RV32 MCU 复位后首先执行初始化序，将中断向量表地址值的高 30 位写入 RV32 寄存器(CSR)mtvec 的 mtvec[31:2]，将 01 写入 mtvec [1:0]，选择向量中断响应模式。

目前市场上 RV32 MCU 外设类型、接口和控制方法与 Cortex-MMCU[3][19][20]相似，中断向量表结构的结构相近。表 2 对比了 Cortex-M MCU STM32F429 与 RV32 MCU GD32VF103 中断向量表结构。

Table 2. The interrupt vector table ofSTM32F429 and GD32VF103
表 2. STM32F429 与 GD32VF103 中断向量表结构

	STM32F429	GD32VF103
中断向量表	_vectors: .word __stack_end__ .wordReset_Handler …… .word ExternalISR0 .word ExternalISR1 ……	_vectors: j Reset_Handler .word 0 …… .word ExternalISR0e .word ExternalISR1 ……

如表 2 所示，RV32 MCU 中断向量中，除向量 0 外，每个 32 位向量值是对应中断服务程序的入口地址。由于复位后 RV32 缺省为非向量中断响应模式，需要首先设置中断响应模式和向量表基地址，向量 0 是跳转指令，跳转到复位启动程序入口。

2.2. 上下文处理

Cortex-MMCU 响应中断请求时，硬件自动依次将 xPSR，PC，LR，R12 以及 R3 - R0 压入栈中，保存上下文；中断服务程序返回时，硬件自动从栈中将数据弹回对应寄存器，恢复上下文。

RV32 MCU 响应中断请求时硬件自动保存 PC 到控制和状态寄存器 mepc，并保存特权模式至 mstatus.MPP，但不保存上下文相关的通用寄存器和其他特殊功能寄存器。从中断服务程序返回时，硬件仅自动恢复寄存器 mstatus 和 PC。将 Cortex-MMCU 中断服务程序移植到 RV32 MCU 时，需要在中断服务程序中添加保存和恢复上下文语句或函数。

Cortex-MMCU 应用程序和中断服务程序遵守 ARM 过程调用规范 AAPCS[17]，硬件在中断响应过程中自动保存 4 个特殊功能寄存器和 4 个参数寄存器 r0-r3 或称为 a0-a3。RISC-V 应用程序过程调用规范约定 8 个参数寄存器 x10-x17 或称为 a0-a7。与 Cortex-MMCU 自动保存的上下文内容对照，RV32 MCU 中断服务程序中需要保存和恢复上下文内容最小包括参数寄存器 a0-a7 和特殊功能寄存器。表 3 列出了 RV32 MCU 中断服务程序中保存和恢复上下文最小集的程序语句。

Table 3. The programs of save and restore context in RV32 MCU interrupt service routine
表 3. RV32 MCU 中断服务程序保存和恢复上下文程序语句

	保存上下文	恢复上下文
修改 sp	addisp, sp, -12*4	addisp, sp, -12*4

控制和状态寄存器	csrr x5, mepc
	sw x5, (sp)
	csrr x5, mstatus
	sw x5, 4(sp)
		lw x5, (sp)
		csrwmepc, x5
		lw x5, 4(sp)
		csrwmstatus, x5

通用寄存器	swra, 2*4(sp)
	sw a0, 3*4(sp)
	sw a1, 4*4(sp)
	sw a2, 5*4(sp)
	sw a3, 6*4(sp)
	sw a4, 7*4(sp)
	sw a5, 8*4(sp)
	sw a6, 9*4(sp)
	sw a7, 10*4(sp)
		lwra, 2*4(sp)
		lw a0, 3*4(sp)
		lw a1, 4*4(sp)
		lw a2, 5*4(sp)
		lw a3, 6*4(sp)
		lw a4, 7*4(sp)
		lw a5, 8*4(sp)
		lw a6, 9*4(sp)
		lw a7, 10*4(sp)

在中断服务程序的入口添加表 3 中保存现场语句或函数，在返回语句前添加恢复现场语句或函数。对于定制中断响应机制 Gd32VF103[21]等 MCU，则需要依据其使用手册编写中断处理相关的程序。

3. 指令集模块

Cortex-M MCU 采用 Thumb 指令集。RV32 MCU 采用模块化指令，生成应用程序时可以选择指令集模块及其组合。将 Cortex-M MCU 应用程序移植到 RV32 MCU 时，选择与 Thumb 指令集相应功能指令集模块组合，以保持移植前后程序功能和性能的一致性。

为了便于分析和评估性能，本文选择 STM32F429 和 FE310 分别作为 Cortex-M MCU 和 RV32 MCU 样本，使用 CoresMark[22]做性能分析；程序生成工具选择 Segger 公司 Embeddedstudio[7]；汇编和编译器选择 gcc，版本为 gnu4.2.1。

STM32F429 内核为 Cortex-M4[23]，指令集为 Thumb2。FE310 支持 RV32imac 指令集模块，处理器指令集功能是所有子模块功能的并集。表 4 对 STM32F29 与 FE310 指令集部分功能进行了比较。

Table 4. The partial functions of STM32F29 and FE310 instruction sets
表 4. STM32F29 与 FE310 指令集部分功能

功能	Thumb2	RV32I	M	A	C
算数/逻辑	是	是	-	-	-
乘法/除法	是	-	是	-	-
内存访问	是	是	-	-	-
原子	否	-	-	是	-

16 位指令	支持	-	-	-	是

生成 RV32 MCU 应用程序时，选择不同的指令集或指令集组合，将会影响程序的性能。

3.1. RV32i vs RV32im

Embedded studio 创建 FE310 应用程序工程时缺省使用 RV32i 指令集模块，该模块没有乘法和除法指令。程序中的乘除法运算能够正常编译，编译器通过调用由 RV32i 指令实现的运算函数库实现。如果在编译时选择 RV32im 指令集模块组合，则编译器将直接使用乘法和除法指令实现运算，提高程序执行速度。表 5 列出了 c 程序采用 RV32i 和 RV32im 指令集编译后生成汇编指令对照。

Table 5. The assembly instructions for RV32i and RV32im
表 5.RV32i 与 RV32im 汇编指令

C 函数	RV32i	RV32im

int add4(int p1, int p2, int p3, int p4)
{
return (p1 * p2 + p3 /p4);
}
	add4:
	mv s1, a2
	mv s2, a3
	call __mulsi3
	mv s0, a0
	mv a1, s2
	mv a0, s1
	call __divsi3
	add a0, s0, a0
	jrra
		add4:
		div a2, a2, a3
		mul a0, a0, a1
		add a0, a0, a2
		ret

表 6 列出了选择 RV32i 与 RV32im 指令集在 FE310 模拟器上运行 CoreMark 得分。结果表明，如果应用程序中含有乘法和除法运算，将 RV32i 改为 RV32im 指令集将提高程序运行速度。

Table 6. The CoreMark scores of RV32i and RV32im
表 6. RV32i 与 RV32imCoreMark 得分

	RV32i	RV32im
得分(/MHZ)	1.33	2.43

3.2. RV32im vs RV32imc

由于 MCU 处理器中存储资源受限，对 ROM 和 RAM 的需求是开发 MCU 应用程序关注的重点之一。Cortex-M 采用支持 16 位指令 Thumb 指令模式，以减少应用程序的体积。RV32i 和 RV32im 指令长度是 32 位。对于同一源程序，使用 RV32im 指令集生成的二进制目标程序的长度将会大于 Cortex-M 目标程序的长度，从而增加对存储资源的需求。选择 RV32imc 指令集组合，将指令长度从 32 位变为 16 位，减少所生成二进制目标程序的长度。表 7 列出了 Coremark 4 个主要文件不同指令集生成的二进制代码长度。

Table 7. The length of binary code generated from CoreMark’s main files (in bytes)
表 7. CoreMark 主要文件生成的二进制代码长度(字节)

目标文件	Cortex-M4	RV32i	RV32im	RV32imc
core_list_join.o	988	1864	1884	1224
core_matrix.o	760	1956	1356	868
core_state.o	622	1112	1104	732

core_util.o	174	308	284	178

从表 7 可见，RV32i 程序的平均长度是 Cortex-M4 的 2.05 倍，RV32imc 程序的平均长度是 Cortex-M4 的 1.2 倍。可见，在将程序从 Cortex-M 移植到 rv32 时，选择 RV32imc 指令集组合，将基本满足原系统对存储资源限制要求。

添加指令集模块“A”，选择 RV32imac 指令集组合，编译后主要文件二进制代码长度与 RV32imac 完全相同，CoreMark 得分 2.34/MHz，与选择 RV32imc 指令集组合时相近。

4. 过程调用规范

过程调用规范(Procedure Call Standard)定义了应用程序二进制接口(Application Binary Interface)，通常包括函数或过程调用中参数传递和结果返回方式，处理器寄存器使用，以及数据类型处理等内容。将 Cortex-MMCU 应用程序移植到 RV32 MCU 时，需要考虑 ARM 处理器和 RISC-V 处理器过程调用规范之间的差异。本节将讨论函数调用过程参数传递方式的差别对移植程序带来的影响。

ARM 架构过程调用规范(AAPCS)[24]约定：在函数和过程调用过程中，调用者(主程序)通过 4 个寄存器 r0-r3 或称为 a0-a3，向函数(被调用者)传递参数。如果参数超过 4 个寄存器数值范围，超出部分利用栈传递；函数通过 a0-a1 返回结果。RISV-V 过程调用规范(RISC-V Procedure Calling Convetion)[25]约定：调用者通过 8 个寄存器 x10-x17 或称为 a0-a7，向被调用者传递参数。如果参数超过 8 个寄存器数值范围，超出部分利用栈传递；被调用者利用 a0-a1 返回结果。表 8 列出了 6 个整数型参数 C 语言函数编译后所生成的 Cortex-M 和 RV32imac 汇编函数。

Table 8. The generated assembly functions from C function
表 8.由 C 函数生成汇编函数

C 函数	Cortex-M4 汇编	RV32imac 汇编

int add6(int p1, int p2, int p3, int p4,
int p5, int p6)
{
return(p1 + p2 + p3 + p4 + p5 + p6);
}
	.global add6
	.thumb
	.type add6, %function
	add6:
	adda0, a0, a1
	add a0, a0, a2
	add a0, a0, a3
	ldr a3, [sp]
	add a0, a0, a3
	ldr a3, [sp, #4]
	add a0, a0, 3
	bxlr
		.global add6
		.type add6, %function
		add6:
		adda0, a0, a1
		add a0, a0, a2
		add a0, a0, a3
		add a0, a0, a4
		add a0, a0, a5
		ret

如表 8 汇编函数所示，Cortex-M4 函数，参数 1 到参数 4 通过 a0-a3 传递，参数 5 和 6 通过栈传递。在 RV32imac 函数中，参数 1-6 通过 a0-a5 传递。Cortex-M4 和 RV32imac 利用 a0 返回结果。

由于访问栈的延时高于访问寄存器，在设计 Cortex-MMCU 应用函数时通常使参数不超过 4*32 位。将 Cortex-MMCU 应用程序移植到 RV32 MCU 时，利用多达 8*32 位寄存器参数传递特性，将减少函数和过程调用带来的延时。

5. 总结与展望

本文从 MCU 中断处理机制，指令集模块组合，以及程序过程调用规范三方面比较 Cortex-M 和 RV32 MCU 的差别，分析了这些差别对将应用程序从 Cortex-M MCU 移植到 RV32 MCU 的影响。为了兼容中

断处理程序，将 RV32 MCU 设置为向量中断响应方式，并在中断服务程序中添加保存和恢复上文语句。在生成应用程序时选择 RV32imc 指令集组合，实现高性能和小体积的应用程序。利用 MCU 寄存器在函数和过程调用过程传递更多参数，降低调用过程中的延时。

RV32 MCU 与 Cortex-M MCU 指令差别很大，指令之间的差异对程序移植性能优化带来挑战。未来将进一步探讨 RV32 程序移植中的性能优化问题。

参考文献

[1] Waterman, A. (2016) Design of the RISC-V Instruction Set Architecture. Ph.D. Dissertation, University of California. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-1.html

[2] SiFive FE310-G002 Manual v1p5.https://www.sifive.com/

[3] RISC-V. https://www.gigadevice.com/product/mcu/risc-v

[4] 青稞 RISC-V 通用系列[EB/OL]. https://www.wch.cn/products/productsCenter/mcuInterfacecategoryId=70, 2024-06-10.

[5] 微控制器[EB/OL]. https://www.hpmicro.com/product/product.html?id=07e8d638-1c6c-44d1-9e53-a48d8560ad78, 2024-06-10.

[6] Reness Introduces Industry's First General-Purpose 32-bit RISC-V MCUs with Internally Developed CPU Core. https://www.renesas.com/us/en/about/press-room/renesas-introduces-industry-s-first-general-purpose-32-bit-risc-v-mcus-internally-developed-cpu-core

[7] Embedded-Studio.https://www.segger.com/downloads/embedded-studio/

[8] The Leading Commercial Tools for RISC-V. https://www.iar.com/products/architectures/risc-v/

[9] Eclipse Embedded CDT (C/C++ Development Tools). https://projects.eclipse.org/projects/iot.embed-cdt

[10] Singhal, S.P., Sridevi, M., Narayanan, N.S., *et al*. (2021) Porting of eChronos RTOS on RISC-V Architecture. In: Hura, G.S., Singh, A.K. and Siong Hoe, L.,Eds.,*Advances in Communication and Computational Technology*, Springer, 1269-1279.https://doi.org/10.1007/978-981-15-5341-7_96

[11] Pieper, P., Herdt, H. and Drechsler, R. (2022) Advanced Embedded System Modeling and Simulation in an Open Source RISC-V Virtual Prototype. *Journal of Low Power Electronics and Applications*, **12**,Article 52. https://doi.org/10.3390/jlpea12040052

[12] 在 RISC-V 微控制器上使用 FreeRTOS[EB/OL]. https://www.freertos.org/zh-cn-cmn-s/Using-FreeRTOS-on-RISC-V.html, 2024-06-13.

[13] 邰阳, 韩昌刚, 全雨, 于佳耕, 武延军. 基于 QEMU RISC-V 架构的 OpenHarmony 标准系统移植[J]. 计算机系统应用, 2023, 32(11): 21-28.

[14] Gordon, N., Pedretti, K. and Lange, J.R. (2022) Porting the Kitten Lightweight Kernel. Operating System to RISC-V.*IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers* (*ROSS*), Dallas,13-18 November 2022, 1-7.https://doi.org/10.1109/ROSS56639.2022.00008

[15] Zhang, L.(2022) The Porting and Optimization of RISC-V UEFI Boot. 2022 7*th International Conference on Intelligent Computing and Signal Processing* (*ICSP*), Xi'an, 15-17 April 2022, 840-843. https://doi.org/10.1109/ICSP54964.2022.9778600

[16] Balas, R. and Benini, L. (2021) RISC-V for Real-Time MCUs—Software Optimization and Microarchitectural Gap Analysis. 2021 *Design, Automation & Test in Europe Conference & Exhibition* (*DATE*), Grenoble,1-5 February 2021, 874-877. https://doi.org/10.23919/DATE51398.2021.9474114

[17] Arm Holdings Market Share across Key Technology Markets Worldwide 2020-2022. https://www.statista.com/statistics/1132112/arm-market-share-targets/

[18] 21 世纪经济报道. RISC-V 快速成长，哪些应用领域落地更快更完整? [EB/OL]. https://new.qq.com/rain/a/20221130A0AATG00.html, 2022-11-30.

[19] ARMv7-M Architecture Application Level Reference Manual. https://www.arm.com/

[20] RISC-V Foundation, the RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. https://riscv.org/

[21] GigaDevice, GD32VF103 User Manual V1.2. fhttps://www.gd32mcu.com/data/documents/userManual/GD32VF103_User_Manual_Rev1.5.pdf

[22] Coremark 1.0. https://github.com/eembc/coremark

[23] STM32 Cortex®-M4 MCUs and MPUs Programming Manual. https://www.st.com.cn/resource/en/programming_manual/pm0214-stm32-cortexm4-mcus-and-mpus-programming-manual-stmicroelectronics.pdf

[24] Procedure Call Standard for the ARM Architecture. https://eecs.umich.edu/courses/eecs373/readings/ARM-AAPCS-EABI-v2.08.pdf

[25] Understanding RISC-V Calling Convention. https://cs.sfu.ca/~ashriram/Courses/CS7ARCH/assets/notebooks/RISCV/RISCV_CALL.pdf