1.	Record Nr.	UNINA9910835062103321
	Autore	Smith Stephen
	Titolo	RISC-V Assembly Language Programming [[electronic resource]] : Unlock the Power of the RISC-V Instruction Set / / by Stephen Smith
	Pubbl/distr/stampa	Berkeley, CA : , : Apress : , : Imprint : Apress, , 2024
	ISBN	979-88-6880-137-2
	Edizione	[1st ed. 2024.]
	Descrizione fisica	1 online resource (369 pages)
	Collana	Maker Innovations Series, , 2948-2550
	Disciplina	004
	Soggetti	Computer programming
	Lingua di pubblicazione	Inglese
	Formato	Materiale a stampa
	Livello bibliografico	Monografia
	Note generali	Includes index.
	Nota di contenuto	Intro -- Table of Contents -- About the Author -- About the Technical Reviewer -- Acknowledgments -- Introduction -- Chapter 1: Getting Started -- History and Evolution of the RISC-V CPU -- What You Will Learn -- Ten Reasons to Learn Assembly Language Programming -- Running Programs on RISC-V Systems -- Coding a Simple "Hello World" Program -- Hello World on the Starfive Visionfive 2 -- Programming Hello World in the QEMU Emulator -- Install QEMU on Windows -- Install QEMU on Linux -- Compiling in Emulated Linux -- About Hello World on the ESP32-C3 Microcontroller -- Summary -- Exercises -- Chapter 2: Loading and Adding -- Computers and Numbers -- Negative Numbers -- About Two's Complement -- RISC-V Assembly Instructions -- CPU Registers -- RISC-V Instruction Format -- About the GCC Assembler -- Adding Registers -- 32-bits in a 64-bit World -- Moving Registers -- About Pseudoinstructions -- About Immediate Values -- Loading the Top -- Shifting the Bits -- Loading Larger Numbers into Registers -- More Shift Instructions -- About Subtraction -- Summary -- Exercises -- Chapter 3: Tooling Up -- GNU Make -- Rebuild a Project -- Rule for Building .S files -- Define Variables -- Build with CMake -- Debugging with GDB -- Preparation to Debug -- Setup for Linux -- Start GDB -- Set Up gdb for the ESP32-C3 -- Debugging with GDB -- Summary -- Exercises -- Chapter 4: Controlling Program Flow -- Creating Unconditional Jumps -- Understanding Conditional Branches -- Using Branch Pseudoinstructions -- Constructing Loops -- Create FOR Loops --

Code While Loops -- Coding If/Then/Else -- Manipulating Logical
Operators -- Using AND -- Using XOR -- Using OR -- Adopting Design
Patterns -- Converting Integers to ASCII -- Using Expressions
in Immediate Constants -- Storing a Register to Memory -- Why Not
Print in Decimal? -- Performance of Branch Instructions.
Using Comparison Instructions -- Summary -- Exercises -- Chapter 5:
Thanks for the Memories -- Defining Memory Contents -- Aligning
Data -- About Program Sections -- Big vs. Little Endian -- Pros of Little
Endian -- About Memory Addresses -- Loading a Register
with an Address -- PC Relative Addressing -- Loading Data
from Memory -- Combining Loading Addresses and Memory -- Storing
a Register -- Optimizing Through Relaxing -- Converting to Uppercase
-- Summary -- Exercises -- Chapter 6: Functions and the Stack --
About Stacks -- Jump and Link -- Nesting Function Calls -- Function
Parameters and Return Values -- Managing the Registers -- Summary
of the Function Call Algorithm -- Uppercase Revisited -- Stack Frames
-- Stack Frame Example -- Defining Symbols -- Macros -- Include
Directive -- Macro Definition -- Labels -- Why Macros? -- Using
Macros to Improve Code -- Summary -- Exercises -- Chapter 7: Linux
Operating System Services -- So Many Services -- Calling Convention
-- Finding Linux System Call Numbers -- Return Codes -- Structures
-- About Wrappers -- Converting a File to Uppercase -- Building .S
Files -- Opening a File -- Error Checking -- Looping -- Summary --
Exercises -- Chapter 8: Programming GPIO Pins -- GPIO Overview -- In
Linux, Everything is a File -- Flashing LEDs -- Moving Closer
to the Metal -- Virtual Memory -- In Devices, Everything is Memory --
Registers in Bits -- GPIO Enable Registers -- GPIO Output Set Registers
-- More Flashing LEDs -- GPIOTurnOn in Detail -- Root Access --
Summary -- Exercises -- Chapter 9: Interacting with C and Python --
Calling C Routines -- Printing Debug Information -- Preserving State --
Calling Printf -- Passing a String -- Register Masking Revisited --
Calling Assembly Routines from C -- Packaging the Code -- Static
Library -- Shared Library -- Embedding Assembly Language Code
inside C Code.
Calling Assembly from Python -- Summary -- Exercises -- Chapter 10:
Multiply and Divide -- Multiplication -- Examples -- Division --
Division by Zero and Overflow -- Example -- Example: Matrix
Multiplication -- Vectors and Matrices -- Multiplying 3x3 Integer
Matrices -- Accessing Matrix Elements -- Register Usage -- Summary
-- Exercises -- Chapter 11: Floating-Point Operations -- About
Floating Point Numbers -- About Normalization and NaNs --
Recognizing Rounding Errors -- Defining Floating Point Numbers --
About Floating Point Registers -- The Status and Control Register --
Defining the Function Call Protocol -- Loading and Saving FPU
Registers -- Performing Basic Arithmetic -- Calculating Distance
Between Points -- Performing Floating-Point Conversions -- Floating-
Point Sign Injection -- Comparing Floating-Point Numbers -- Example
-- Summary -- Exercises -- Chapter 12: Optimizing Code --
Optimizing the Uppercase Routine -- Simplifying the Range
Comparison -- Restricting the Problem Domain -- Tips for Optimizing
Code -- Avoiding Branch Instructions -- Moving Code Out of Loops --
Avoiding Expensive Instructions -- Use Macros -- Loop Unrolling --
Delay Preserving Registers in Functions -- Keeping Data Small --
Beware of Overheating -- Summary -- Exercises -- Chapter 13:
Reading and Understanding Code -- Browsing Linux &amp -- GCC
Code -- Comparing Strings -- About the Algorithm -- Macros
and Kernel Options -- Code Created by GCC -- Reverse Engineering
and Ghidra -- Summary -- Exercises -- Chapter 14: Hacking Code --

Buffer Overrun Hack -- Causes of Buffer Overrun -- Stealing Credit Card Numbers -- Stepping Through the Stack -- Mitigating Buffer Overrun Vulnerabilities -- Do Not Use strcpy -- PIE Is Good -- Poor Stack Canaries Are the First to Go -- Preventing Code Running on the Stack -- Tradeoffs of Buffer Overflow Mitigation Techniques -- Summary.
Exercises -- Appendix A: The RISC-V Instruction Set -- RV32I Base Integer Instruction Set -- RV64I Base Integer Instruction Set-in Addition to RV32I -- RV32M Standard Extension -- RV64M Standard Extension-in Addition to RV32M -- RV32F Standard Extension -- RV64F Standard Extension-in Addition to RV32F -- RV32D Standard Extension -- RV64D Standard Extension-in Addition to RV32D -- Appendix B: Binary Formats -- Integers -- Floating Point -- Addresses -- Appendix C: Assembler Directives -- Appendix D: ASCII Character Set -- Appendix E: Answers to Exercises -- Chapter 2 -- Chapter 3 -- Chapter 5 -- Chapter 6 -- Chapter 8 -- Chapter 10 -- Chapter 12 -- Index.

Sommario/riassunto

Gain the skills required to dive into the fundamentals of the RISC-V instruction set architecture. This book explains the basics of code optimization, as well as how to interoperate with C and Python code, thus providing the starting points for your own projects as you develop a working knowledge of assembly language for various RISC-V processors. The RISC-V processor is the new open-source CPU that is quickly gaining popularity and this book serves as an introduction to assembly language programming for the processor in either 32- or 64-bit mode. You’ll see how to write assembly language programs for several single board computers, including the Starfive Visionfive 2 and the Espressif ESP32=C3 32-bit RISC-V microcontroller. The book also covers running RISC-V Linux with the QEMU emulator on and Intel/AMD based PC or laptop and all the tools required to do so. Moving on, you’ ll examine the basics of the RISC-V hardware architecture, all the groups of RISC-V assembly language instructions and understand how data is stored in the computer’s memory. In addition, you’ll learn how to interface to hardware such as GPIO ports. With RISC-V Assembly Language Programming you’ll develop enough background to use the official RISC-V reference documentation for your own projects. What You'll Learn See how data is represented and stored in a RISC-V based computer Make operating system calls from assembly language and include other software libraries in projects Interface to various hardware devices Use the official RISC-V reference documentation.