Scot W. Stevenson

Einführung in die moderne

Assembler-Programmierung

RISC-V spielerisch und fundiert lernen

dpunkt.verlag

Scot W. Stevenson programmiert seit den Tagen von Acht-Bit-Prozessoren, wie dem 6502 in Assembler. Vom Bytegeschiebe konnten ihn weder sein Medizinstudium, ein Graduiertenkolleg Journalismus, mehr als zwei Jahrzehnte als Nachrichtenredakteur noch ein Blog über die USA abbringen. Er behauptet trotzdem, jederzeit damit aufhören zu können.

Scot W. Stevenson

Einführung in die moderne Assembler-Programmierung

RISC-V spielerisch und fundiert lernen

dpunkt.verlag

Scot W. Stevenson

Lektorat: Gabriel Neumann, Julia Griebel
Lektoratsassistenz: Friederike Demmig
Copy-Editing: Annette Schwarz, Ditzingen
Satz: lll-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner, Frank Heidt
Umschlaggestaltung: Eva Hepper, Silke Braun

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-98889-007-8
PDF 978-3-98890-155-2
ePub 978-3-98890-156-9

1. Auflage 2024

Wieblinger Weg 17
69123 Heidelberg

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *hallo@dpunkt.de*.

For my parents, for curiosity endlessly encouraged

Inhaltsübersicht

Inhaltsverzeichnis

Vorwort

»But I don't want to go among mad people,« Alice remarked. »Oh, you ca'n't help that,« said the Cat: »we're all mad here. I'm mad. You're mad.« »How do you know I'm mad?« said Alice. »You must be,« said the Cat, »or you wouldn't have come here.«

–Lewis Carroll, Alice's Adventures in Wonderland (1865) in Originalschreibweise

Ein modernes Buch über Assembler-Programmierung, was soll das denn? Macht das heute nicht alles eine Maschine, sei es ein Compiler oder eine KI? Was für Wahnsinnige befassen sich denn noch mit so was?

Nun, einige Leute werden im Studium von gemeinen Menschen dazu gezwungen, sich mit Assembler zu beschäftigen. Aus tiefstem Mitgefühl heraus versprechen wir ihnen: Wir bringen das schnell, schmerzlos und so unterhaltsam wie möglich über die Bühne.

Entsprechend gibt es am Anfang dieses Buches etwas zu den Grundlagen, einen Überblick über Prozessoren, die benötigten Werkzeuge und natürlich Assembler. Allgemeines Wissen über die Programmierung reicht aus, Vorkenntnisse zu Assembler oder spezifischen Hochsprachen wie C sind nicht nötig.

Wir nutzen dabei den offenen Prozessorstandard RISC-V, der auch gezielt für Forschung und Lehre entwickelt wurde. Das macht die Sache für alle einfacher, denn der Kern-Befehlssatz, den wir hier vorstellen, umfasst weniger als 50 Instruktionen. Noch besser: Wer RISC-V lernt, lernt fürs Leben, denn der Befehlssatz ist »eingefroren« und ändert sich nicht mehr.

Das bringt uns zu den Leuten, die speziell RISC-V-Assembler-Programmierung lernen wollen (oder müssen). Für sie gehen wir im Mittelteil den Aufbau des Prozessors durch, wobei der Schwerpunkt auf der Software liegt. Wir stellen die einzelnen Befehle vor, warnen vor Fallstricken und verraten Tricks. Die Schwachstellen des Standards werden gnadenlos beleuchtet.

Dabei verzichten wir zwar keinesfalls auf Compiler oder KI. Aber als offener, freier Standard wird RISC-V zunehmend für Hobby- und Studenten-Projekte ein-

gesetzt, wo der Compiler nur schlecht oder gar nicht an die Hardware angepasst ist, falls es überhaupt einen gibt. Dann muss der Mensch ran, ob mit oder ohne eine künstliche Helferin.

Aber um ehrlich zu sein: Dieses Buch entstand auch aus schierer Begeisterung für Assembler heraus. Wer es liebt, die schnellste Schleife herauszuarbeiten und sich diebisch über jedes eingesparte Byte freut, wird die hinteren Abschnitte kaum abwarten können. Damit ist es am Ende tatsächlich auch ein Buch für die Leute, die vielleicht ein klein wenig wahnsinnig sind. Hier sind sie unter Freunden.

Viel Spaß.

Thema

Ein Buch wie dieses steht vor der Herausforderung, unterschiedlichen Lesergruppen gerecht zu werden: von Neulingen an der Universität, die vielleicht zum ersten Mal in die Eingeweide eines Prozessors blicken, bis hin zu Vollprofis, die vermutlich bereits mehr über Assembler vergessen haben, als der Autor jemals wissen wird. Wir werden für die erste Gruppe am Anfang Begriffe erklären und Hintergründe erläutern. Je tiefer wir in die Materie eindringen, desto mehr Wissen setzen wir als bekannt voraus.

Was ist Assembler (ganz kurz)?

Eigentlich sind Computer fürchterliche Geräte. Sie verstehen nur Zahlen, was Menschen langfristig nicht guttut. Zudem gehen sie in winzigen Arbeitsschritten vor, was uns nur deswegen nicht auffällt, weil sie jeden einzelnen dieser Schritte *sehr* schnell ausführen. Auf dieser untersten, menschenfeindlichen Ebene der Programmierung sprechen wir von der »Maschinensprache«, auf Englisch *machine language*.

Um die Arbeit mit Computern für Nicht-Freaks erträglich zu machen, wird die Maschinensprache unter einem Berg von Abstraktionen versteckt. Dazu gehören Hochsprachen wie Python oder Rust, die von speziellen Programmen wie Compilern oder Interpretern über Zwischenschritte in Maschinensprache übersetzt werden. Bei einer Low-Code-Entwicklungsumgebung werden grafische Elemente zusammengeklebt, sodass – wenn überhaupt – nur »niedrige« Programmierkenntnisse erforderlich sind. Die gegenwärtig höchste Abstraktionsstufe sind die Anweisungen an eine Künstliche Intelligenz (KI), die »Prompts«.

»Assembler« ist in diesem Modell die erste Abstraktionsebene nach der Maschinensprache. Das Wort kommt von dem englischen Verb *to assemble,* hier im Sinne von »aus Einzelteilen zusammensetzen« und nicht von »Menschen sammeln sich« wie der Schlachtruf *Avengers assemble!* bei Marvel. Vereinfacht gesagt wird dabei jeder Maschinenbefehl 1:1 durch einen Assembler-Befehl abgebildet, der aus mehr oder weniger leicht zu merkenden Abkürzungen besteht. Das

Problem der winzigen Arbeitsschritte bleibt. Der Katalog aller Befehle eines Prozessors ist sein »Befehlssatz«, auf Englisch *instruction set*.

Da jede Prozessorfamilie eine andere Maschinensprache mitbringt, die sich zum Teil auch noch von Prozessormodell zu Prozessormodell unterscheidet, gibt es viele Varianten von Assembler. Wir sprechen von dem jeweiligen Befehlssatz eines Modells oder einer Chip-Familie, der *instruction set architecture* (ISA). Neben dem RISC-V-Befehlssatz gibt es etwa den x86 bei Intel und AMD bei klassischen PCs sowie die Arm-Befehlssätze insbesondere bei mobilen Geräten.

Der unvermeidliche geschichtliche Überblick

Dieses Programm hätte die traumhafte Laufzeit von 32 Millisekunden, aber den alptraumhaften Speicherbedarf von 24 KByte.

–Peter-Mattias Oden, Grafik-Tuning. Schneller Bildaufbau mit 6502-Prozessoren am Beispiel des Apple II (1984)

Als Erfinderin von Assembler gilt Kathleen Booth vom Birkbeck College in London – das war 1947. [BK] Zwar wurde sogar noch die erste Version des IBM-Mainframe-Betriebssystems OS/360 1966 in Assembler geschrieben. Allerdings ging es dann wegen des Aufstiegs von höheren Sprachen wie Fortran, Lisp und Cobol bergab. Assembler galt als tot. [SD]

Der Aufstieg der Acht-Bit-Computer auf der Grundlage von Prozessoren wie dem 6502 oder Z80 Mitte der 70er Jahre brachte eine Auferstehung. [SD] Zunächst gab es keine Compiler für diese Mikroprozessoren (MPU). Wer die maximale Leistung aus der Kiste holen wollte, kam an Assembler nicht vorbei. Computerzeitschriften wie *c't* waren in den 80er Jahren entsprechend noch ein Stück mehr *hardcore* als heute und muteten ihren Lesern seitenweise Listings von Assembler-Mathematik und -Grafikcode zu. Assembler auf diesen Prozessoren machte einfach *Spaß*, nicht zuletzt weil die Befehlssätze für Menschen geschrieben wurden.

Mit einer größeren Leistungsfähigkeit der Rechner wurden allerdings auch die Compiler immer besser. Der Aufwand, einen schnelleren Code per Hand zu schreiben, lohnte sich immer weniger. Zudem traten die x86-Prozessoren ihre jahrzehntelange Dominanz an mit riesigen, barocken Befehlssätzen, die für Normalsterbliche kaum zu durchdringen waren und dazu noch mit Nutzungseinschränkungen bewehrt sind. Assembler wurde bestenfalls für Bootloader verwendet, und wer das machen musste, tat es meistens fluchend. Die Befehlssätze werden seitdem und bis heute für Compiler entworfen. Wenn es um die Praxis ging, schien Assembler tot, schon wieder.

Also warum dann jetzt noch Assembler?

> *I could skip the middleman and talk right to the machine. (…) I could talk to God, just like IBM.* [TK]
>
> *–Tracy Kidder, The Soul of a New Machine (1981)*

Vermutlich sollte an dieser Stelle eine flammende Rede für den Nutzen von Grundwissen über Assembler in der Computerwissenschaft stehen – dass erst damit klar wird, wie die Maschine auf der untersten Ebene funktioniert, dass dieses Wissen für den Compilerbau unabdingbar ist, dass es bei der Optimierung von Code helfen kann. Das ist auch alles wahr.

Allerdings müssen wir der Realität ins Auge sehen: Eine Menge Leute lernen im 21. Jahrhundert Assembler nur, weil es Teil eines Pflichtkurses an der Uni ist. Deswegen ist dieses Buch erstens kurz und zweitens befasst es sich mit RISC-V. Wie wir gleich ausführlicher sehen werden, geht dieser Befehlssatz auf frustrierte Informatik-Dozenten zurück, die unter anderem ein besseres Werkzeug für die Lehre haben wollten. Der minimalistische Kern-Befehlssatz kann selbst den desinteressierten Massen im Grundkurs zugemutet werden, die danach nie wieder irgendwas mit Assembler zu tun haben (wollen).

Profis werden dagegen vermutlich die ersten Teile des Buches überspringen und sich auf RISC-V konzentrieren wollen. Es gibt immer noch Situationen, wo es Assembler sein muss, zum Beispiel die Portierung von Betriebssystemen auf neue Prozessoren. Der langsame, aber stetige Aufstieg von RISC-V in der Industrie zwingt mehr Leute dazu, sich mit diesem Neuling zu beschäftigen. Diese Teile des Buches sind daher auch zum Nachschlagen gedacht und absichtlich etwas nüchterner geschrieben (wenn auch nicht viel). Profis haben ja keine Zeit.

Langfristig sollten diese beiden Lesergruppen deckungsgleich werden: Ein Ziel von RISC-V ist, dass im Studium derselbe Befehlssatz gelehrt wird, der auch in der Praxis verwendet wird. Die Neueinstellungen würden dann nützliches, sofort einsetzbares Wissen von der Uni mitbringen, was einem kleinen Wunder gleichkäme.

Hobby-Coder als dritte Gruppe sind dagegen nicht nur die Spiel-, sondern auch die Glückskinder der Programmierwelt. Sie können tun, was sie wollen, in welcher Sprache sie es wollen, so lange, wie sie es wollen. Ihre Projekte können den praktischen Nutzen eines überfahrenen Stinktiers haben. Assembler coden sie entsprechend zum Spaß, auch wenn ihnen besorgte Bekannte T-Shirts schenken mit Schriftzügen wie »Hauptsache, es tut weh«.

Der große Feind des Hobby-Coders sind Updates und neue Features. Neben Familie und Job und was sonst noch im Leben wirklich wichtig ist bleibt ein Projekt manchmal so lange liegen, bis sich Staub auf der Tastatur sammelt. Wer sich dann erst in neue Frameworks, Funktionen oder Updates reinfuchsen muss,

kommt weniger zum Programmieren. Lizenzbedingungen können ein weiteres Problem sein. Es ist daher kein Wunder, dass sich viele Freizeit-Assembler-Fans in die Retro-Programmierung etwa des 6502 aus dem Acht-Bit-Zeitalter geflüchtet haben. RISC-V macht jetzt damit Schluss: Der Befehlssatz ist nicht nur kurz, sondern auch »eingefroren« und ändert sich nicht wieder.

Für diese Gruppe ist insbesondere der dritte und letzte Teil des Buches gedacht. Die dort vorgestellten Routinen, Verfahren und Programme würden von vernünftigen Menschen in einer Hochsprache geschrieben (oder gar nicht), der Stoff bietet jedoch tiefere Einblicke in die Assembler-Welt. Für die ganz Harten diskutieren wir schließlich am Ende noch weitergehende Projekte, die zu umfangreich für dieses Buch wären. An ihnen dürften nur noch zwei Gruppen Freude haben: sadistische Dozenten und masochistische Hobby-Coder. T-Shirts für alle!

Was ist RISC-V?

> *The most pervasive change in this edition is switching from MIPS to the RISC-V instruction set. We suspect this modern, modular, open instruction set may become a significant force in the information technology industry. It may become as important in computer architecture as Linux is for operating systems.*
>
> *–Hennessy und Patterson, Computer Architecture: A Quantitative Approach (2019)*

RISC-V wird auf Englisch *»risk five«* ausgesprochen und als »Risk fünf« eingedeutscht, auch wenn einige KIs gegenwärtig noch auf *»risk vee«* bestehen. Der erste Teil des Namens kommt von *Reduced Instruction Set Computer* und bezeichnet Prozessoren, die über vergleichsweise wenige Befehle verfügen, aber diese sehr schnell ausführen. Dabei wird etwas mehr Code benötigt als bei einem *Complex Instruction Set Computer* (CISC). Die römische Ziffer V verweist darauf, dass es der fünfte Anlauf der Erfinder ist. Das RISC-V-Projekt ist vergleichsweise jung, in der heutigen Form nahm es 2010 seinen Anfang. Über die Einhaltung der Standards wacht seit 2020 die Stiftung RISC-V International mit Sitz in der Schweiz.

Als Befehlssatzdefinition existiert RISC-V eigentlich nur auf dem Papier. Sie besteht aus einer Spezifikation, die nichts darüber aussagt, wie der Prozessor die Befehle umsetzt. Ob konventionelle Hardware wie Logikgatter, elektromagnetische Relaistechnik wie zu Zeiten von Konrad Zuse oder dressierte Hamster in speziellen Laufrädern, alles ist möglich.

Unter uns gesagt: Der Befehlssatz an sich ist nicht fürchterlich aufregend und schon gar nicht revolutionär. Wer sich bereits mit der ISA von anderen RISC-Pro-

zessoren beschäftigt hat, wird vieles wiedererkennen. Vielmehr zeichnen RISC-V zwei Dinge aus:

Erstens, der Standard ist »offen« oder »frei«, denn die Spezifikation unterliegt einer Creative-Commons-Lizenz. Damit kann jeder selbst RISC-V-Prozessoren bauen, ob als Bastelfreak im Hobbykeller, multinationaler Konzern mit eigener Chip-Fertigung oder Verein für ambitionierte Hamster-Trainer. Forschung und Lehre sind keine Grenzen gesetzt, Unternehmen müssen keine Lizenzgebühren bezahlen und Freizeit-Coder bekommen keine Auflagen aufgedrückt.

Zweitens, es handelt sich um einen »modularen« Standard. Während der x86-Befehlssatz durch sein unablässiges Wachstum inzwischen bei einer vierstelligen Zahl von Instruktionen angekommen ist, werden die RISC-V-Befehle in »Module« verpackt. [HS] Diese werden nach eingehender Prüfung »eingefroren« *(frozen)* und nie wieder verändert. Es gibt ein Basismodul I, das alle RISC-V-Prozessoren haben müssen. Darüber hinaus entscheidet jede Chip-Schmiede und jeder Hamster-Trainer selbst, welche Module sie benötigen.

Formalitäten

So weit ein erster Überblick. Leider kommt kein Buch ohne Bürokratie aus. Bringen wir sie schnell hinter uns.

Englisch

> *Deutsch im Code sagt dem Leser auf den ersten Blick: Hier hat jemand nur für sich selbst programmiert, ohne damit zu rechnen, dass sich jemals jemand anders für den Code interessieren könnte. Das tun überwiegend Anfänger, also ist der Code wahrscheinlich nicht besonders gut.*
>
> *–Passig und Jander, Weniger schlecht programmieren (2013)*

Jedes deutschsprachige Buch über Computer muss damit klarkommen, dass Englisch die Weltsprache der Informatik ist. Besonders bei RISC-V liegt bislang viel Literatur nur auf Englisch vor. Früher oder später kommt niemand daran vorbei. Der Einsatz von Künstlicher Intelligenz verstärkt diesen Effekt nur, weil die Modelle gegenwärtig deutlich besser mit Englisch zurechtkommen als mit Deutsch. *Sorry.*

Die gute Nachricht ist, dass die erforderlichen Englischkenntnisse eher auf der Sprachebene von *Friends* liegen als von William Shakespeare. Auch Englischmuffel kommen mit etwas Übung klar. Wir führen am Anfang die englischen Fachbegriffe ein und setzen sie dann nach und nach als bekannt voraus. Auch die Kommentare in den Quelltexten *(source code)* sind irgendwann durchgängig auf Englisch, weil das der Situation in der wirklichen Welt entspricht.

Code

Ein Ziel dieses Buches ist es, möglichst viele gut lesbare Code-Beispiele zur Verfügung zu stellen. Dabei steht insbesondere am Anfang die Klarheit des Designs im Vordergrund. Tricks, um ein Maximum an Leistung oder den kürzesten Code herauszukitzeln, führen wir erst ein, wenn das Grundprinzip klar ist. Die Programme sind ausführlich kommentiert. Wer schon mal versucht hat, fremden Assembler-Code zu lesen – oder nach einigen Wochen den eigenen –, weiß, warum. Ein Kommentar pro Zeile wird keine Seltenheit sein.

Eine historisch gewachsene Unsitte bei Assembler ist die Verwendung von sehr kurzen Namen oder gar einzelnen Buchstaben für Variablen und Sprungmarken *(label)*. Dafür gibt es im 21. Jahrhundert keine Entschuldigung, wir verwenden lange Namen. Konstanten werden in VERSALIEN geschrieben, auch wenn es der Maschine egal ist.

Die Grobstruktur von Routinen wird meist so aussehen, dass wir ganz oben einsteigen und ganz unten wieder rausgehen. Anders formuliert soll jede Routine möglichst immer nur einen Eingang und einen Ausgang haben. Im Rahmen des *defensive programming* bauen wir hin und wieder Code ein, der nur dazu dient, das Programm robuster zu machen. Entsprechende Stellen markieren wir in den Kommentaren als *paranoid*. Wir sagten ja bereits, hier sind Wahnsinnige am Werk.

Werkzeuge

Die Beispielprogramme wurden entweder auf dem RARS-Simulator (*https://github.com/TheThirdOne/rars*) oder mit QEMU unter Ubuntu Linux mit dem GCC Compiler (*https://gcc.gnu.org/*) getestet. RARS ist der einfachere Weg. Das Programm wird als ausführbare `jar`-Datei bereitgestellt und müsste auf ziemlich jedem Betriebssystem mit

```
java -jar <DATEI>
```

von der Kommendozeile aus ausführbar sein.

GCC ist dafür deutlich mächtiger. Ubuntu bietet für QEMU ein vorgefertigtes Image unter *https://wiki.ubuntu.com/RISC-V* an, das ein komplettes RISC-V-System emuliert. Das heißt, wir können innerhalb dieser Umgebung mit normalen Werkzeugen arbeiten. Für dieses Buch wurde benutzt:

```
ubuntu-22.04.1-preinstalled-server-riscv64+unmatched.img
```

Wir rufen die QEMU-Instanz auf mit:

```
qemu-system-riscv64 \
-machine virt \
-nographic \
-m 2048 \
-smp 4 \
-bios /usr/lib/riscv64-linux-gnu/opensbi/generic/fw_jump.elf \
-kernel /usr/lib/u-boot/qemu-riscv64_smode/uboot.elf \
-device virtio-net-device,netdev=eth0 \
-netdev user,id=eth0,hostfwd=tcp::10022-:22 \
-drive file=<UBUNTU_IMAGE>,format=raw,if=virtio
```

Wir können uns dann von einem anderen Rechner aus mit `ssh -p 10022 ubuntu@<RECHNER>` einloggen.

That said, above all this book tries not to take itself (or anything) too seriously. There is humour here, the difference is that you need to look for it.

–Doug Hoyte, Let Over Lambda (2008)

Teil I

Grundlagen

In diesem Teil geht es um die allgemeinen Grundlagen von Assembler. Außerdem wird hier das Mindestwissen vermittelt, um verstehen zu können, was ganz unten in einem Computer passiert. Wer den Stoff parat hat, kann diesen Teil überspringen.