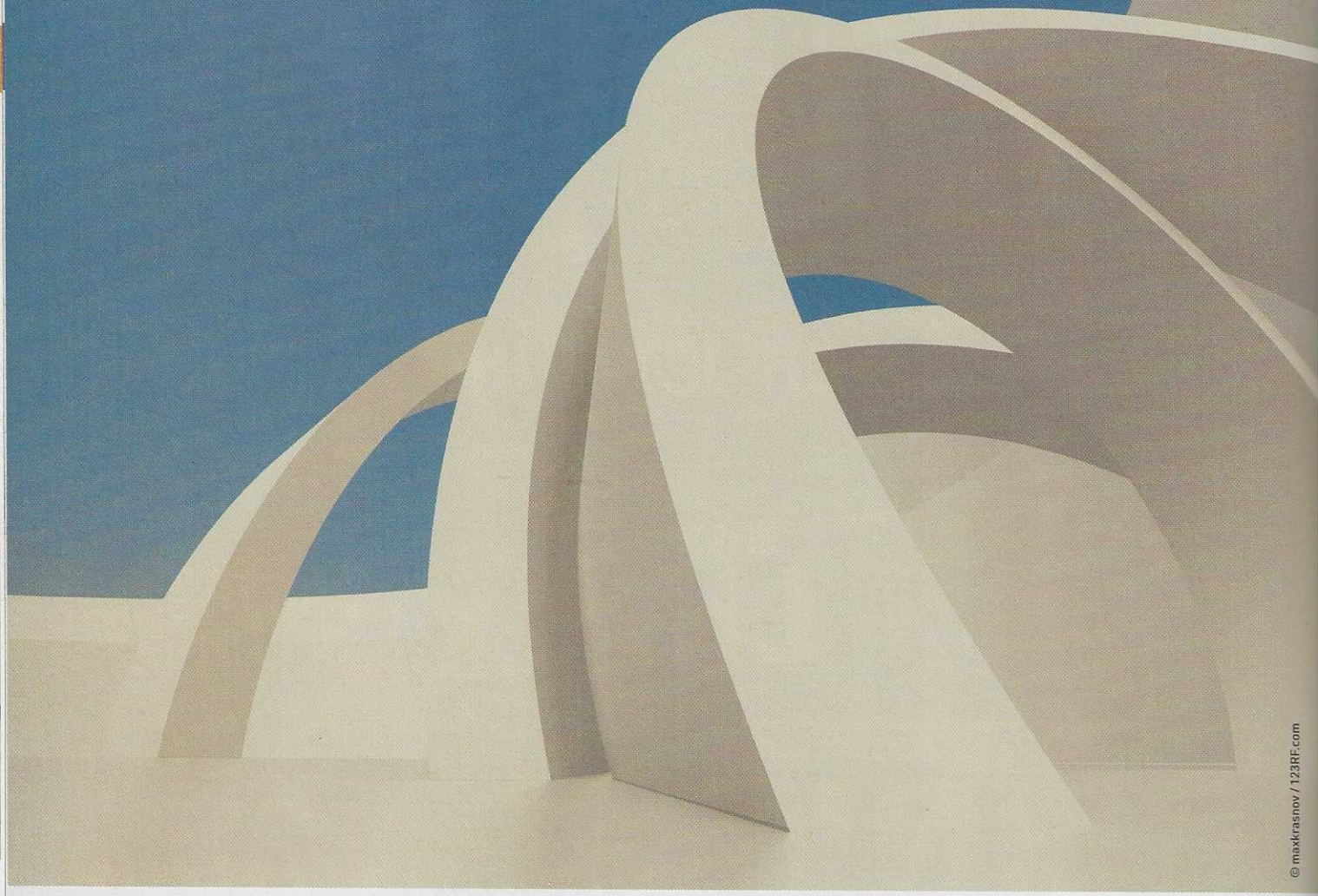


Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 119

Moderne Architektur

Ausgestattet mit Open-Source-Genen, mischt die neue CPU-Architektur RISC-V momentan den Prozessormarkt auf. Dank des Emulators Qemu machen Sie auch ohne echte Hardware erste Bekanntschaft mit dem Prozessortyp.

Eva-Katharina Kunst, Jürgen Quade

Der Autor

Eva-Katharina Kunst ist seit den Anfängen von Linux Fan von Open Source. **Jürgen Quade**, Professor an der Hochschule Niederrhein, führt auch für Unternehmen Schulungen zu den Themen Treiberprogrammierung und Embedded Linux durch.

RISC-V – ausgesprochen Risk Five – gilt als neuer Stern am Prozessor-Himmel. Server, PCs und Notebooks vertrauten bislang das Verarbeiten von Daten einer CPU von Intel oder AMD an, die auf der sogenannten x86-Architektur beruht. Der Raspberry Pi, Smartphones und Tablets dagegen basieren überwiegend auf Prozessoren einer ARM-Architektur.

Noch vor zwei Jahrzehnten kämpften mit MIPS, PowerPC, Alpha und SPARC weitaus mehr CPU-Hersteller und Architekturen um den Markt, insbesondere um den der eingebetteten Systeme. Diesen Bereich konnte x86 trotz massiver Investitionen aufgrund seiner Komplexität und dem damit verbundenen Stromhunger nicht einmal ansatzweise für sich erschließen.

Advanced RISC Machine – dafür steht das Kürzel ARM – punktete hier mit einer simplen RISC-Architektur, ausreichender Rechenleistung, einem einfachen Lizenzschema und vor allem mit Effizienz. Ein Glück übrigens, denn sonst würden uns unsere Smartphones heute womöglich mit einem ständig surrenden Lüfter nerven. Dementsprechend konsolidierte sich der Markt in den letzten Jahren auf die beiden CPU-Architekturen x86 und ARM.

Vor zehn Jahren entwickelte die University of California, Berkeley, einen neuen CPU-Befehlssatz (Instruction Set Architec-

ture, ISA) namens RISC-V, der modern und sowohl für die Lehre und Forschung, aber auch für die tatsächliche Anwendung geeignet sein sollte. Der besondere Clou war das Lizenzmodell, das auf BSD basierend dem Lizenznehmer die kostenfreie Nutzung der Architektur ermöglicht (siehe auch Tabelle RISC-V: Merkmale).

Die Cal, wie man die University of California auch liebevoll nennt, überträgt damit das Open-Source-Prinzip auf die Hardware und scheint damit ähnlichen Erfolg zu haben wie Linux. Große wie kleine Firmen springen auf den RISC-V-Zug auf, entwickeln, produzieren und vertreiben Hardware, die einen RISC-V-Kern enthält, und stellen ihre Designs kostenfrei zur Verfügung. Einen ersten Eindruck diesbezüglich vermittelt der Wikipedia-Eintrag zur Architektur.

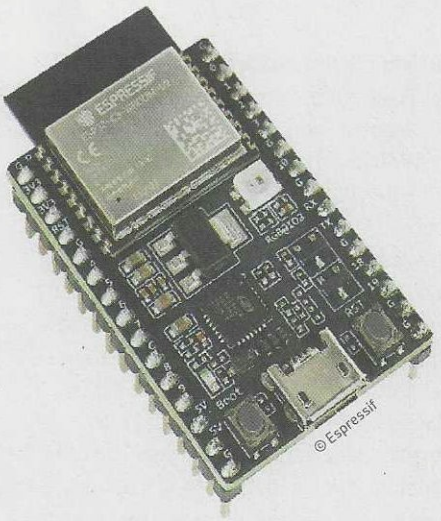
Der große chinesische Online-Händler Alibaba beispielsweise gibt den von ihm entwickelten RISC-V-Kern komplett frei, ebenso die Platten-Schmiede Western Digital. Selbst Apple, das mit seinem M1-Chip eine sehr leistungsstarke und effiziente ARM-CPU in der Hinterhand hat, sucht seit neuestem RISC-V-Entwickler.

Stabiler Kern

Ähnlich wie ARM, aber ganz anders als x86 lässt sich die RISC-V-Architektur in weiten Bereichen skalieren. Neben Varianten, die eine ausgewachsene Memory Management Unit (MMU) mitbringen und sich demzufolge für Linux eignen, lassen sich auch simpelste Mikrocontroller herstellen, die man bare-metal – also ohne Zuhilfenahme irgendeiner Systemsoftware – programmieren oder mit dem Echtzeitbetriebsystem FreeRTOS bestücken kann.

Die einfachsten RISC-V-Varianten implementieren lediglich die Core-Befehle, auch I-Befehle genannt. Sie enthalten neben logischen Operationen gerade noch Addieren und Subtrahieren; Multiplizieren und Dividieren bleibt der sogenannten M-Erweiterung vorbehalten. Die Tabelle RISC-V: Befehlssatzerweiterungen listet übliche Extensions auf wie etwa die F-Erweiterung für Fließkomma oder die C-Erweiterung für komprimierte Instruktionen. Standardmäßig schlägt die Cal vor, zumindest die Erweiterungen I, M, F, D und A zu implementieren und kürzt eine solche Architektur mit dem Buchstaben G (für „general") als RV32G ab.

RISC-V ist als 32-, 64- und sogar 128-Bit-Architektur definiert. Anders formuliert: Alle internen Register sind abhängig von der Architektur 32, 64 oder sogar 128 Bit breit. Von diesen Registern gibt es nicht weniger als 32 beziehungsweise 64 Stück, die man als *X0* bis *X31* respektive *X63* bezeichnet. Sieht man vom Register *X0* ab, das man nur lesen kann und bei dem alle Bits auf 0 stehen, lassen sich die Register universell einsetzen. Einen speziellen Stack Pointer gibt es beispielsweise nicht. Allerdings sieht die Architektur einen separaten Program Counter (PC) vor, also einen Befehlszähler, der die Adresse des Befehls enthält, den die CPU als Nächstes abarbeiten soll.



1 Espressif bietet das ESP32-C3-DevKitC-02 schon für knapp 8 Euro an.

Während es sich bei der x86-Architektur um eine sogenannte Zwei-Register-Maschine handelt, sind bei RISC-V typischerweise drei Register an einer Operation beteiligt. Addiert eine x86-Maschine die Inhalte zweier Register, muss sie das Ergebnis in eines der beiden beteiligten Register ablegen. Bei einer Drei-Register-Maschine hingegen landet das Ergebnis in einem dritten Register (*X1 = X2 + X3*).

Jungspund

Mit rund 10 Jahren darf die RISC-V-Architektur als vergleichsweise jung gelten. Das ist auch der Grund dafür, dass es noch keine wirklich preiswerte Bastelhardware gibt, auf der ein Linux liefe. Für einen Einstieg müssen Enthusiasten derzeit noch rund 600 Euro hinblättern. Das dürfte sich allerdings schon in naher Zukunft ändern.

RISC-V: Merkmale	
Gegenstand	Befehlssatzdefinition (ISA)
Modularität	Basisbefehlssatz plus Erweiterungen
Registerbreite	32, 64, 128
Anzahl Register	16 (für Embedded), 32, 64
Architekturtyp	Load/Store
Dateiablageformat	Little Endian
Anzahl Befehle	40 (RV32I, Computational, Control Flow, Memory Access)
Privilege Level	3
Maschinentyp	Drei-Register-Maschine
Befehlstypen	6 (R, I, U, S, B, J)
Software	umfangreiches Software-Ökosystem mit populären Software-Stacks (Linux, FreeRTOS)
Lizenz	BSD-Lizenz
Sonstiges	stabile Spezifikation, keine Änderungen mehr geplant

RISC-V: Befehlssatzerweiterungen	
RV32I	Base Instruction Set
RV32A	A-Extension (Atomic read-modify-write)
RV32B	B-Extension (Bit Manipulation)
RV32C	C-Extension (Compressed Instructions)
RV32D	D-Extension (Double Precision Floating Point)
RV32F	F-Extension (Single Precision Floating Point)
RV32M	M-Extension (Multiplication, Division)
RV32S	S-Extension (Supervisor Level Instructions)
RV32V	V-Extension (Vector Instructions)

Sobald die Prozessoren in relevanten Stückzahlen produziert werden, fallen die Preise erfahrungsgemäß signifikant.

Wer übrigens auf Linux verzichtet und mit FreeRTOS vorliebnimmt, kann heute schon preiswert einsteigen und Erfahrungen sammeln. Das mit einem RISC-V-Kern ausgestattete Board ESP32-C3-DevKitC-02 kostet weniger als 10 Euro 1. Hier hat der chinesische Hersteller Espressif seinem bekannten Erfolgsmodell ESP32 einen RISC-V-basierten ESP32-C3 zur Seite gestellt, der zwar etwas weniger Leistung bietet als das klassische Modell, dafür aber kleiner und vor allem preiswerter ist.

Wollen Sie RISC-V in Kombination mit Linux kennenlernen, kommen Sie sogar komplett ohne eigenständige Hardware aus. Nicht nur, dass sich unter Ubuntu ein RISC-V-Compiler installiert lässt: Auch der Hardwareemulator Qemu bildet schon seit einiger Zeit ein vollständiges RISC-V-System nach, sowohl in der 32- als auch in der 64-Bit-Variante. Die Distributionhersteller wiederum haben RISC-V-Varianten ihrer Softwaresammlungen im Programm, sodass Sie flugs zur Tat schreiten können.

RISC-V-Linux installieren

Mit einem unter Ubuntu aufgerufenen `apt install` ist es allerdings nicht getan. Tatsächlich benötigen wir für unseren emulierten RISC-V-Rechner ein BIOS, einen Bootloader und schließlich mit Linux das Betriebssystem selbst.

Damit steht am Anfang der RISC-V-Compiler, den Sie über das Kommando `apt install gcc-riscv64-linux-gnu` installieren. Das quelloffene BIOS OpenSBI laden Sie in der aktuellsten Version herunter und generieren es. Als Bootloader dient das im Embedded-Bereich bekannte U-Boot, das Sie ebenfalls in der RISC-V-Version kompilieren.

Das bei Canonical gehostete RISC-V-Server-Image schließlich zielt mit seinen 466 MByte Umfang auf einen kleinen Speicher-Footprint ab. Sie müssen es nach dem Herunterladen und Entpacken mithilfe der Qemu-Werkzeuge noch auf eine brauchbare Größe bringen. Ob das 20 GByte oder nur 5 GByte sein sollen, muss jeder für sich selbst entscheiden.

Qemu schließlich will mit einer Fülle an Parametern gestartet werden, insbesondere auch für die Netzwerkverbindung, falls Sie denn einen Internet-Zugang nutzen wollen. Der User *Aaronfranke* stellt ein Skript bereit, das die wesentlichen Installationsschritte abbildet. Listing 1 zeigt eine von uns modifizierte und erweiterte Variante, die Optimierungen und Anpassungen vornimmt und quasi automatisch ein bootbares System zaubert.

Dazu legt das Skript in Ihrem Home-Verzeichnis einen neuen Unterordner `risc-v/` an, installiert die notwendige Software per `apt install` und lädt ansonsten benötigte Komponenten inklusive des vorgefertigten Ubuntu-Images aus dem Internet nach. Es generiert die

Listing 1: RISC-V-Ubuntu für Qemu

```
#!/bin/bash
BASISDIR=~/risc-v
UBUNTU_IMG=ubuntu-20.04.3-preinstalled-server-riscv64+unmatched.img

if test ! -f `type -p riscv64-linux-gnu-gcc`; then
  echo "install compiler..."
  sudo apt install gcc-riscv64-linux-gnu
fi
if test ! -f `type -p qemu-system-riscv64`; then
  echo "install qemu..."
  sudo apt install qemu-system-misc qemu-utils
fi
if test ! -d ${BASISDIR}; then
  echo "mkdir ${BASISDIR}"
  mkdir ${BASISDIR}
fi
cd ${BASISDIR}

if test ! -f opensbi/build/platform/generic/firmware/fw_payload.bin; then
  wget https://cdimage.ubuntu.com/releases/20.04/release/${UBUNTU_IMG}.xz

  xz -dk ${UBUNTU_IMG}.xz
  qemu-img resize -f raw ${UBUNTU_IMG} +6G
  export CROSS_COMPILE=riscv64-linux-gnu-
  git clone https://source.denx.de/u-boot/u-boot.git
  cd u-boot/
  git reset --hard v2021.10-rc3
  make qemu-riscv64_smode_defconfig
  make -j$(nproc)
  cd ..
  git clone https://github.com/riscv/opensbi.git
  cd opensbi/
  make PLATFORM=generic FW_PAYLOAD_PATH=../u-boot/u-boot.bin
  cd ..
fi

qemu-system-riscv64 -machine virt -m 4G -nographic \
  -bios opensbi/build/platform/generic/firmware/fw_payload.bin \
  -smp cores=2 -gdb tcp::1234 \
  -device virtio-net-device,netdev=eth0 \
  -netdev user,id=eth0,hostfwd=tcp::2222-:22 \
  -drive if=none,file=${UBUNTU_IMG},format=raw,id=mydisk \
  -device ich9-ahci,id=ahci -device ide-hd,drive=mydisk,bus=ahci.0 \
  -device virtio-rng-pci
```

Firmware und den Bootloader und startet schließlich das RISC-V-Linux. Der Download und die Installation erfolgen übrigens nur beim ersten Aufruf.

Speichern Sie das Skript beispielsweise in eine Datei `risc-v-linux.sh` ab, ändern Sie die Zugriffsrechte per `chmod 755 risc-v-linux.sh` auf Ausführen, und starten Sie schließlich die Installation beziehungsweise das Linux in der Konsole über den Aufruf `./risc-v-linux.sh`. An dieser Stelle brauchen Sie allerdings etwas Geduld, denn im Gegensatz zur Virtualisierung wird die RISC-V-CPU hier in Software emuliert. Einige Minuten später sollte jedoch die Aufforderung zum Login erscheinen, der Sie mit dem Benutzernamen *ubuntu* und dem gleichlautenden Passwort nachkommen 2. Das System zwingt Sie dann als Erstes, aus Sicherheitsgründen das Passwort zu wechseln.

```
[  OK  ] Reached target Network.
[  OK  ] Reached target Network is Online.
[  OK  ] Reached target Host and Network Name Lookups.
[  OK  ] Reached target Remote File Systems (Pre).
[  OK  ] Reached target Remote File Systems.
         Starting LSB: automatic crash report generation...
         Starting Deferred execution scheduler...
         Starting Availability of block devices...
[  OK  ] Started Regular background program processing daemon.
         Starting OpenBSD Secure Shell server...
         Starting Permit User Sessions...
[  OK  ] Finished Remove Stale Onli…ext4 Metadata Check Snapshots.
[  OK  ] Started Deferred execution scheduler.
[  OK  ] Finished Availability of block devices.
[  OK  ] Finished Permit User Sessions.
[  OK  ] Started System Logging Service.
         Starting Hold until boot process finishes up...
         Starting Terminate Plymouth Boot Screen...
[  OK  ] Finished Hold until boot process finishes up.
[  OK  ] Finished Terminate Plymouth Boot Screen.

Ubuntu 20.04.3 LTS ubuntu ttyS0

ubuntu login:
```

2 **Der Login-Screen des RISC-V-Ubuntu.**

Kein Konsolenspaß

Da die serielle Konsole einige Schwächen aufweist, loggen Sie sich am besten über ein anderes Terminalfenster per SSH auf dem neuen System ein. Qemu hat unser Skript dazu beim Aufrufen so konfiguriert, dass das per `ssh -p 2222 ubuntu@localhost` klappt. Geben Sie jetzt das Kommando `uname -a` ein, dann sehen Sie, dass Sie auf einer RISC-V-Maschine arbeiten. Ein `cat /proc/cpuinfo` zeigt ebenfalls die Architektur inklusive der implementierten Erweiterungen an, gibt

```
The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Fri Oct 22 07:46:05 2021
ubuntu@ubuntu:~$ uname -a
Linux ubuntu 5.11.0-1017-generic #18~20.04.1-Ubuntu SMP Thu Aug 12 00:38:00 UTC
2021 riscv64 riscv64 riscv64 GNU/Linux
ubuntu@ubuntu:~$ cat /proc/cpuinfo
processor       : 0
hart            : 0
isa             : rv64imafdcsu
mmu             : sv48

processor       : 1
hart            : 1
isa             : rv64imafdcsu
mmu             : sv48

ubuntu@ubuntu:~$ file /usr/bin/bash
/usr/bin/bash: ELF 64-bit LSB shared object, UCB RISC-V, version 1 (SYSV), dynam
ically linked, interpreter /lib/ld-linux-riscv64-lp64d.so.1, BuildID[sha1]=28443
143ab66709ba096a1a175d915571464dd7f, for GNU/Linux 4.15.0, stripped
ubuntu@ubuntu:~$
```

3 Spuren der RISC-V_Architektur lassen sich leicht finden.

sich aber sonst überraschend schmallippig. Bei einem `cat /proc/interrupts`, einem `dmesg` oder einem `file /usr/bin/bash` zeigen sich weitere Spuren der RISC-V-Architektur 3. Insgesamt aber macht Linux seinem Ruf eines plattformunabhängigen Betriebssystems alle Ehre, indem es die neuartige CPU-Architektur unter der Haube gut versteckt.

Per `apt update` aktualisieren Sie wie gewohnt die Paketliste. Vermeiden Sie aber ein `apt upgrade` – das könnte Kernel und Bootloader austauschen und zu Problemen führen. Generell gilt: Die Software läuft noch nicht in allen Teilen hundertprozentig stabil. Haben Sie erst einmal per `apt install build-essential gdb vim` einen Compiler und Entwicklungswerkzeuge installiert, können Sie ein kleines Hello-RISC-V-Programm eingeben und kompilieren. Sehen Sie sich dann CPU-Register und Assembler-Code einmal im GNU-Debugger an 4.

Die CPU-Register tragen übrigens sowohl im Debugger als auch bei der Ausgabe des Kommandos `dmesg --kernel` nicht die oben erwähnte Bezeichnung *X0* bis *X31*. Vielmehr tauchen dort die Namen `ra`, `sp`, `t0`, `s6` und so weiter auf. Die RISC-V Foundation hat eine Konvention zur Verwendung der Register vorgeschlagen respektive festgelegt, an die sich die Programmiererinnen und Programmierer beziehungsweise die Hersteller der Entwicklungswerkzeuge auch halten. Demnach fungiert beispielsweise das Register *X2* als Stack Pointer (`sp`); *X1* enthält bei einem Unterprogrammaufruf die Rücksprungadresse (`ra`). Die Register mit einem T im Namen dienen dem temporären Zwischenspeichern von Daten, die mit einem A als ersten Buchstaben enthalten bei einem Funktionsaufruf die Aufrufparameter.

Aber mit diesen Details müssen sich Entwicklerinnen und Entwickler in den meisten Fällen nicht herumschlagen, sie sind vor allem für Kernel-Programmierer bei der Fehlersuche relevant.

```
t5             0x3      3
t6             0x40     64
pc             0x2aaaaaa640     0x2aaaaaa640 <main+22>
(gdb) disassemble
Dump of assembler code for function main:
   0x0000002aaaaaa62a <+0>:     addi    sp,sp,-48
   0x0000002aaaaaa62c <+2>:     sd      ra,40(sp)
   0x0000002aaaaaa62e <+4>:     sd      s0,32(sp)
   0x0000002aaaaaa630 <+6>:     addi    s0,sp,48
   0x0000002aaaaaa632 <+8>:     mv      a5,a0
   0x0000002aaaaaa634 <+10>:    sd      a1,-32(s0)
   0x0000002aaaaaa638 <+14>:    sd      a2,-40(s0)
   0x0000002aaaaaa63c <+18>:    sw      a5,-20(s0)
=> 0x0000002aaaaaa640 <+22>:    auipc   a0,0x0
   0x0000002aaaaaa644 <+26>:    addi    a0,a0,128 # 0x2aaaaaa6c0
   0x0000002aaaaaa648 <+30>:    jal     ra,0x2aaaaaa550 <puts@plt>
   0x0000002aaaaaa64c <+34>:    li      a5,0
   0x0000002aaaaaa64e <+36>:    mv      a0,a5
   0x0000002aaaaaa650 <+38>:    ld      ra,40(sp)
   0x0000002aaaaaa652 <+40>:    ld      s0,32(sp)
   0x0000002aaaaaa654 <+42>:    addi    sp,sp,48
   0x0000002aaaaaa656 <+44>:    ret
End of assembler dump.
(gdb)
```

4 Ein kleines „Hello, RISC-V", in GDB disassembliert.

Qemu verlassen

Am Ende des Tages fahren Sie das RISC-V Ubuntu beispielsweise im Terminal mit `sudo poweroff` sauber herunter. Um danach Qemu zu verlassen, benutzen Sie die Tastenkombination [Strg]+[A],[X].

Topf und Deckel

Neben dem hier vorgestellten Ubuntu-RISC-V Image stellen auch Fedora und Debian vorgefertigte Images zur Verfügung. Die von uns getestete Debian-Variante zickte allerdings etwas herum und ließ die Installation des Lieblingseditors Vim nicht zu. Es ist ohnehin spannender, auf einen Selbstbau statt auf ein vorgefertigtes System-Image zu setzen. Das klappt mit dem System-Builder Buildroot und einer kurzen Anleitung.

Es gibt also keinen Grund, sich über mangelnde Linux-Unterstützung für die neue Befehlssatzarchitektur zu beschweren. Das ist auch kein Wunder: Ein pfiffiges, modulares, quelloffenes Design, dazu eine freie Lizenz – Linux und RISC-V passen einfach perfekt zusammen. (*jlu*)

Dateien zum Artikel herunterladen unter
www.lm-online.de/dl/44748

Weitere Infos und interessante Links
www.lm-online.de/qr/44748