Universität Bamberg

A RISC-V Experience Report

Michael Engel (michael.engel@uni-bamberg.de)

Practical Computer Science, esp. Systems Programming (SYSNAP) University of Bamberg, Germany
https://www.uni-bamberg.de/sysnap

Chair of the RISC-V academia & training SIG

The RISC-V ecosystem

New processor architecture developed at UC Berkeley since 2010

- Krste Asanović and grad students Y. Lee and A. Waterman at the Parallel Computing Lab, headed by David Patterson (one of the RISC inventors)
- **Open standards** (e.g. TCP/IP) and **open source software** (e.g. Linux) are successful in the industry
  - *Why is the **Instruction Set Architecture** (ISA) proprietary?*
  - Patents prevent competing implementations of binary compatible processors for the x86, ARM and MIPS ISAs
- ***Plan: aid both academic and industrial users with a new ISA design***
  - Innovation via competition from designers of open and proprietary implementations of RISC-V compliant processors
  - Shared open core designs: shorter time to market, lower cost from reuse, fewer errors given many more eyeballs, and transparency that would make it hard, e.g., for government agencies to add secret trap doors
  - Processors becoming affordable for more devices, which helps expand the Internet of Things (IoTs), which could cost as little as $1

Why a new ISA?

- Open ISA standards already existed [1,3]
  - SPARC V8 – IEEE standard since 1994
  - OpenRISC – Open source effort since 2000, 64 bit in 2011

ISA	Base+Ext	Compact Code	Quad FP	Address 32-bit	Address 64-bit	Address 128-bit	Software GCC	Software LLVM	Software Linux	Software QEMU
SPARC V8			✓	✓			✓	✓	✓	✓
OpenRISC				✓	✓		✓	✓	✓	✓
RISC-V	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Requirements for a new ISA

- Future-proof due to
  - Extensible (base+extension) ISA
  - Quad-precision (QP) as well as SP and DP floating point
  - 128-bit addressing as well as 32-bit and 64-bit
- Scalable from tiny embedded to large applications
  - Optional compact instruction set encoding for memory-constrained devices, e.g. in IoT applications

Open source vs. open ISA

RISC-V is an *open specification*

- Contrast to *open source* hardware or software
  - RISC-V implementations can be closed or open source
  - Open source (e.g. Linux) software always provides source code
    - could also built on closed specifications

Open specifications → lots of variants

- More than 100 RISC-V implementations (closed and open source) available
- Compliance tests ensure conformity [13]

Open source can also lead to variants

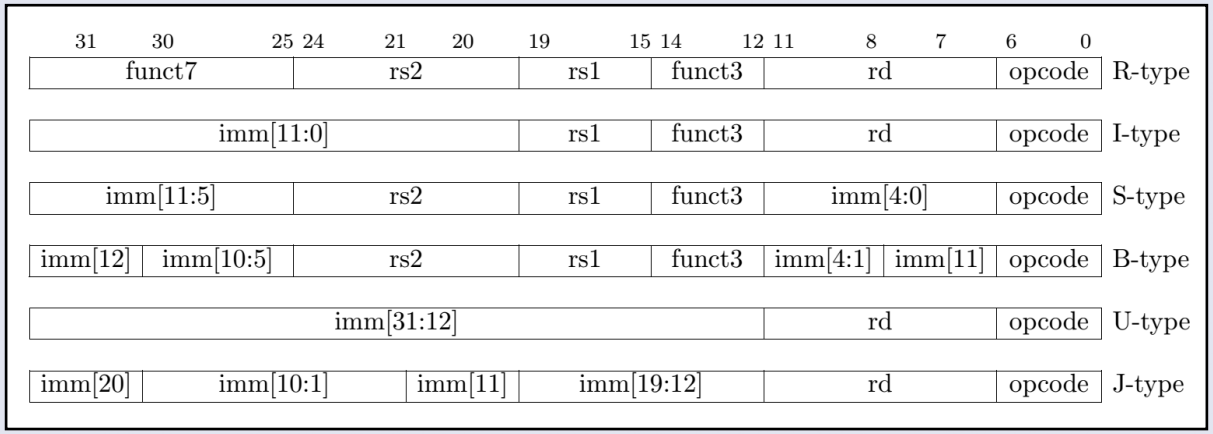
- One (official) Linux kernel source tree, hundreds of different distributions

Microsoft ad in Germany
by Jung v. Matt, 2000

Basic RISC-V ISA

32-bit (RV32) and 64-bit (RV64) versions

- Six instruction types (similar to MIPS)
- Immediate values scattered all over the instruction word (simpler decoding)



Opcode space reserved for extensibility

- Custom ISA extensions defined by RISC-V international as well as third parties

inst[4:2] inst[6:5]	000	001	010	011	100	101	110	111 (>32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	≥80b

RISC-V instruction set extensions

Two base ISAs in 32 and 64 bit:

- RV32I – 32 bit integer instructions
- RV32E – small register set (only 16)
- RV64I – 64 bit integer instructions

Typical extensions

- M: multiplication and division
- F/D: single/double precision FP

Support for system software

- Zicsr: Control/status register access
- A: atomics

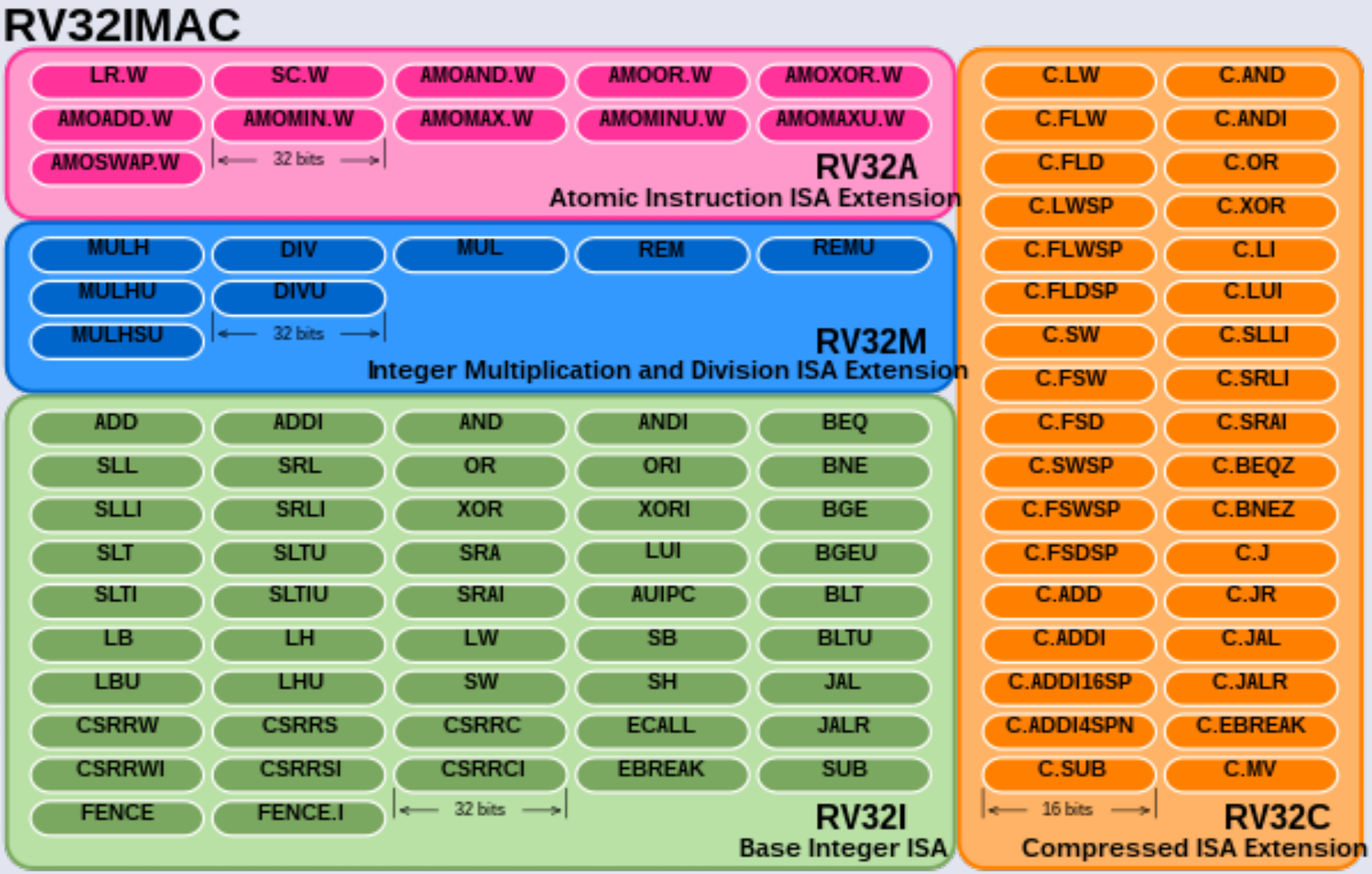
RISC-V defines an exact order that must be used to define the RISC-V ISA subset:

`RV [32, 64, 128]` `I, M, A, F, D, G, Q, L, C, B, J, T, P, V, N`

For example, `RV32IMAFDQC` is legal, whereas `RV32IMAFDCQ` is not.

Subset	Name	Implies
Base ISA		
Integer	I	
Reduced Integer	E	
Standard Unprivileged Extensions		
Integer Multiplication and Division	M	
Atomics	A	
Single-Precision Floating-Point	F	Zicsr
Double-Precision Floating-Point	D	F
General	G	IMADZifencei
Quad-Precision Floating-Point	Q	D
Decimal Floating-Point	L	
16-bit Compressed Instructions	C	
Bit Manipulation	B	
Dynamic Languages	J	
Transactional Memory	T	
Packed-SIMD Extensions	P	
Vector Extensions	V	
User-Level Interrupts	N	
Control and Status Register Access	Zicsr	
Instruction-Fetch Fence	Zifencei	
Misaligned Atomics	Zam	A
Total Store Ordering	Ztso	
Standard Supervisor-Level Extensions		
Supervisor-level extension "def"	Sdef	
Standard Hypervisor-Level Extensions		
Hypervisor-level extension "ghi"	Hghi	
Standard Machine-Level Extensions		
Machine-level extension "jkl"	Zxmjkl	
Non-Standard Extensions		
Non-standard extension "mno"	Xmno	

Base RISC-V ISA overview



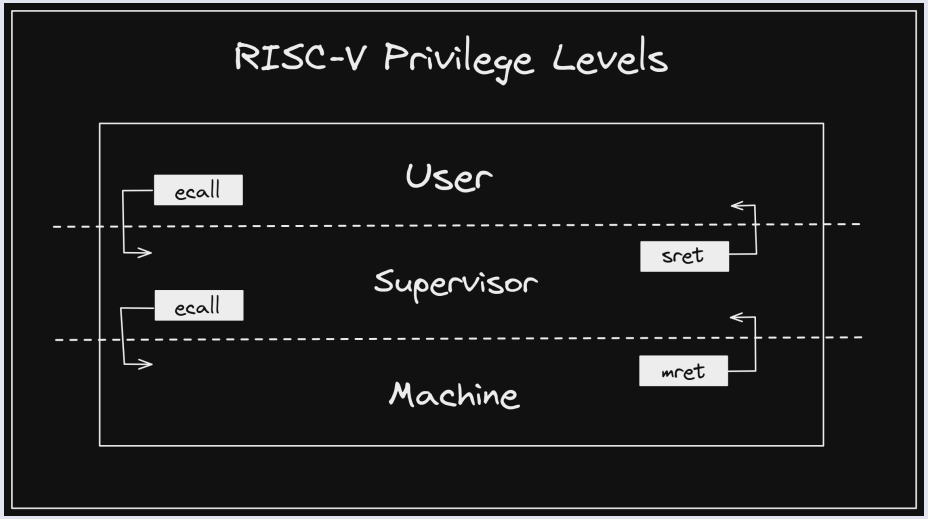
RISC-V privilege architecture

RISC-V defines different privilege levels [5]

- Machine (**M**) – highest privilege full hardware access (firmware), e.g. interrupt control and forwarding and system timer
- Supervisor (**S**) – restricted in terms of interactions with physical hardware, e.g. physical memory and device interrupts, to support clean virtualization (OS kernel)
- User (**U**) – applications

Only machine mode is mandatory, M+S+U (common) **and M+U are possible**

- Optional: Hypervisor mode (**H**)
  - Extension of S mode with additional virtualization capabilities
  - Enables two-level virtual address translation for user processes



from [4]

The RISC-V open HW/SW ecosystem

- **Processor cores** (examples, list at [6])
  - **Academia:** BOOM, PULP, PicoRV, FemtoRV, SERV, …
  - **Industry:** WD SweRV, XuanTie C/E90x, …
- **Firmware**
  - OpenSBI, coreboot, oreboot, u-boot
- **Operating systems and hypervisors**
  - Linux, seL4, FreeRTOS, Zephyr, Haiku, Plan 9/Inferno, Oberon
  - Xen, KVM (work in progress)
- **Compilers and development tools**
  - C/C++: gcc and clang/LLVM
  - Go, Rust, Java
  - Debugging: gdb, lldb, OpenOCD
- **Simulation and emulation**
  - qemu, tinyemu, SystemC models, …

RISC-V challenge: hardware availability

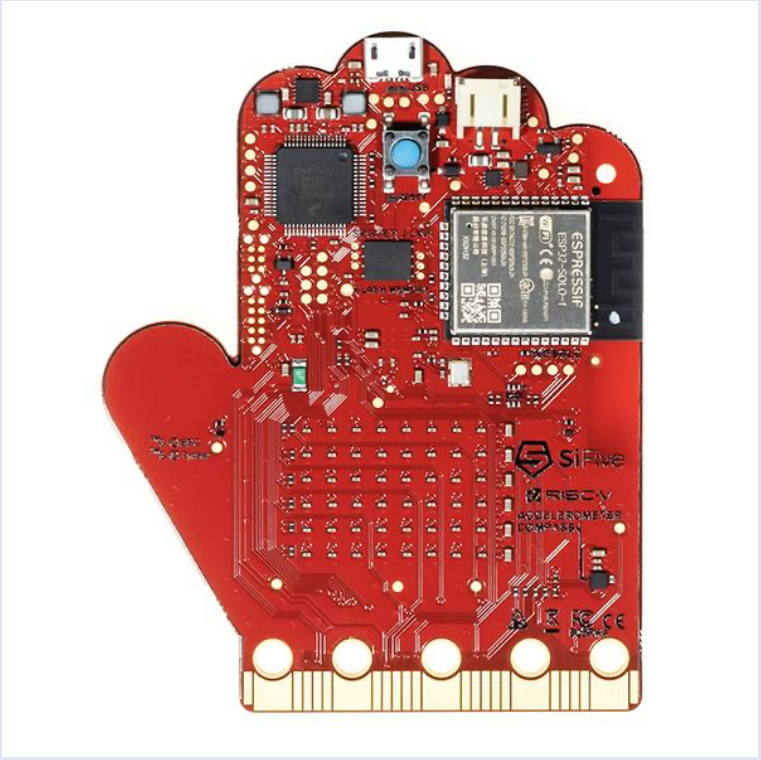
Which RISC-V hardware is available?

- **ASICs**
  - Microcontrollers – RV32I(MA), M mode only, little memory
    - SiFive FE310, GigaDevice GD32VF103, WCH CH32V307
  - High performance SoCs – special applications (wireless radio, audio)
    - Espressif ESP32-C, Bouffalo Labs BL602/4, Kendryte K210/510
  - Desktop/server – Linux-capable, M+S+U mode, MMU
    - Allwinner D1, SiFive U740
  - Accelerators – custom highly parallel systems, e.g. for machine learning
    - Esperanto ET-SoC-1 1088-core RV64-based AI Inference Accelerator
- **FPGA soft cores**
  - Closed source commercial cores – SiFive, Andes, codasip, SemiDynamics, IQonIC
  - Open source commercial cores – T-Head E/C9xx (Allwinner D1), WD SweRV
  - Numerous open source projects – some of very high quality
    - very special implementations such as the space-efficient bit-serial SERV
- **FPGA hard core**
  - Microchip PolarFire FPGA (4+1 core RV64)

Available RISC-V hardware

- SiFive FE310
  - RV32IMAC, 150 MHz, 16 KB Instruction Cache, 16 KB Data Scratchpad, external flash
  - GPIO, UART, SPI, PWM

- Available on the Dr Who inventor kit
  - WiFi & BT via external espressif ESP32 module
  - Light, acceleration sensor, compass, pushputtons
  - Color LED matrix (6x8)



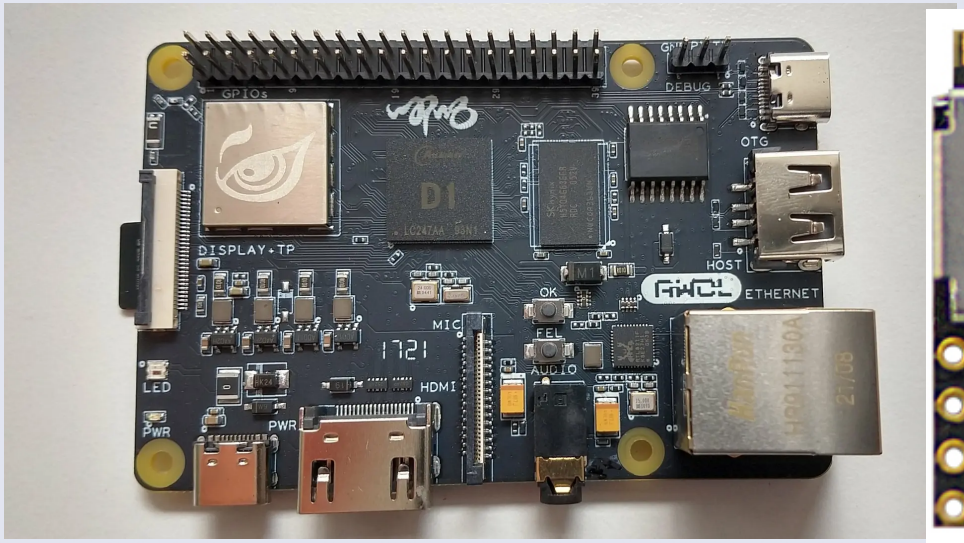
Available RISC-V hardware

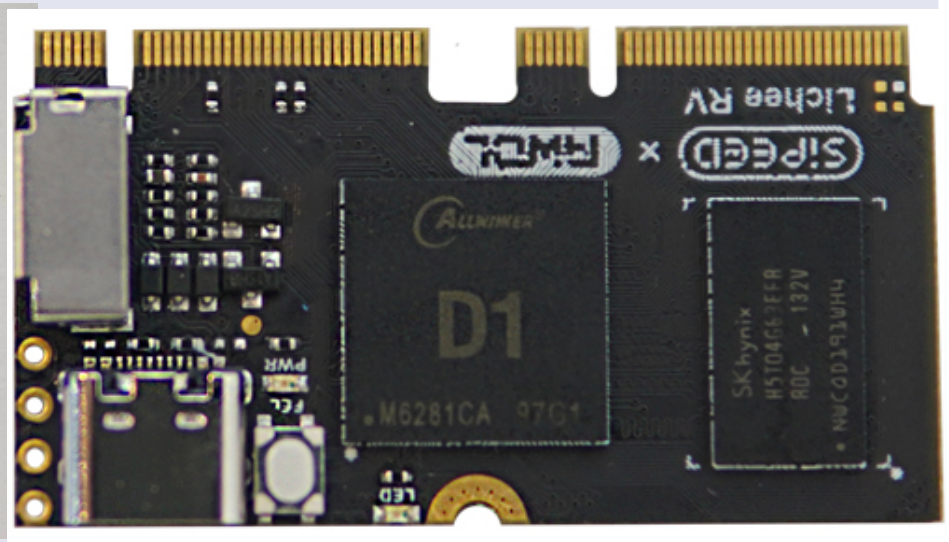
	GigaDevice GD32VF103	WCH CH32V307	Bouffalo Labs BL602/604	Espressif ESP32-C3	Kendryte K210
Core speed	RV32IMAC 108 MHz	RV32IMAC 144 MHz	RV32IMAC 192 MHz	RV32IMAC 160 MHz	dual RV64GC 400 MHz
Flash RAM	128 kB 32 kB	256 kB 64 kB	0-4 MB 277 kB	– 400 kB	– (external) 8 MB
Peripherals	USB, GPIO, UART, IIC, SPI, I2S, PWM	USB, GPIO, UART, IIC, SPI, I2S, PWM	USB, GPIO, UART, IIC, SPI, I2S, PWM	GPIO, UART, IIC, SPI, I2S, PWM	USB, GPIO, UART, IIC, SPI, I2S, PWM
Radio / network	–	1 Gbps MAC 10 Mbps PHY 2 x CAN 2.0B	WiFi 802.11b/g/n BT5 (LE)	WiFi 802.11b/g/n BT5 (LE)	–
Special I/O	–	–	–	–	Camera, mic. array
Other	–	–	–	–	MMU Neural accel.

Linux-capable RISC-V hardware

Nezha and LicheeRV boards based on Allwinner D1 SoC https://linux-sunxi.org/D1 – $99 vs. $25

- Single-core 1 GHz 64-bit RV64GC, 512 MB–2 GB RAM
- Peripherals: USB, Wifi/BT, Ethernet, audio, multiple GPIO
- Our xv6 OS port running "bare metal" on D1 platforms: https://github.com/michaelengel/xv6-d1





Desktop-class RISC-V hardware

SiFive HiFive Unmatched

- SiFive Freedom U740 SoC (quad core RV64GC, 1.5 GHz)
- 16 GB DDR4 RAM
- Gigabit Ethernet
- 4x USB 3.2 Gen 1 Type A
- PCIe Gen 3 x16 Expansion Slot (8 lanes useable)
- M.2 M-Key Slot (PCIe Gen 3 x4) for NVME 2280 SSD Module
- M.2 E-Key Slot (PCIe Gen 3 x1) for Wi-Fi / Bluetooth Module

Research-oriented teaching @ SYSNAP

Universität Bamberg

Operating system projects (at NTNU and now Uni Bamberg)

- Enable students to **learn about the HW/SW interface of RISC-V** by **porting** small real-world operating systems
  - Plan 9 and Inferno (WiP) on RV64GC [11,12]
  - Oberon (https://github.com/solbjorg/oberon-riscv) on RV32IM
  - f9 microkernel on ESP32-C3 (RV32IM) [10]
- Enable students to **explore RISC-V ISA extensions**
  - Rust-based hypervisor using the H extension
- **Operating systems engineering** course (at Uni Bamberg)
  - Design and implement ideas from OS research papers in xv6
    - e.g. virtual memory and virtualization, efficient syscalls…
  - RISC-V platform: Allwinner D1 RV64GC
  - Uses emulation (qemu) and real hardware (D1 Nezha boards)

Current research @ SYSNAP

New virtual memory approaches for persistent main memories

- Provide flexible page sizes and object protection to enable object management in persistent main memory
  - Revisit ideas from Liedtke's guarded page tables (GPT [8]) in a RISC-V core with software-based TLB management
- Ensure data consistency via SW transactional memory (STM [7]) or HW transactional memory extensions to RISC-V

Distributed operating system environments for IoT applications

- Fill the gap between small microcontrollers and Linux-capable processors
- Provide a secure OS built for a distributed heterogeneous environment
  - Microkernel-based OS + Plan 9/Inferno user space as IoT OS basis [9]
- Current projects:
  - Run Inferno OS on top of the f9 microkernel (MPU only system)
  - Code size reduction of Inferno

RISC-V challenge: tooling

- Many different RISC-V implementations
  → several different languages and tools used to develop cores

- ***Chisel*** used to design the first RISC-V cores at Berkeley is a Scala-based hardware design language

  "The Chisel hardware construction language used to design many RISC-V processors was also developed in the [UCB] Par Lab"

- Today, RISC-V implementations exist [4] written in
  - Chisel, Clash, PyMTL, nMigen, Bluespec, SpinalHDL, CDL
  - …as well as in traditional HDLs VHDL and (System)Verilog

RISC-V challenge: binary compatibility

- Opcodes with different meanings as RV32 and RV64 instructions
  - RISC-V started as 64-bit ISA, 32-bit was an afterthought
- Programs compiled for RV32I can work on a RV64I machine
  - It will run without generating an illegal instruction exception
  - but ***the result will most probably be wrong***
  - opcodes are almost all completely legal, but ***semantics differ***
- Example:

```
addi t0, x0, 1
slli t0, t0, 16
slli t0, t0, 16
```

  results in `t0 == 0` for RV32 and `t0 == 2^32` in RV64/128
- So far, no cores support both RV32 and RV64 modes

RISC-V challenge: code models

RISC-V minimizes one of the largest costs in implementing complex ISAs: ***addressing modes***

- RISC-V only has three addressing modes
  using aggressive linker relaxation to reduce the code size:
  - Absolute "**medlow**": restricts code to be linked around address 0
    - uses `lui` instruction (though arguably this is just x0-offset)
  - PC-relative "**medany**": allows the code to be linked at any address
    - uses `auipc`, `jal` and `br*` instructions
    - can have an appreciable performance effect
  - Both restrict the generated code to being linked within a 2 GB window
  - Register-offset:
    - uses `jalr`, `addi` and all memory instructions

- Specifically problematic for system software implementation
  - Requires understanding the linker behavior to debug errors

RISC-V challenge: emulation

RISC-V enforces the activation of physical memory protection (PMP) in systems implementing S/U mode

- Memory regions have to explicitly enabled for non M-mode code

However, **PMP was not enforced in emulation** (qemu < 7.0)

- Result: hard to debug error
- Enabling virtual memory via satp and returning from M→S via the `mret` instruction without configured PMP caused an exception in the OS on real HW (D1)
  - This worked perfectly in qemu…
- The exception handler ran into the same condition and called itself again…

RISC-V challenge: extension compatibility

Opcode ranges reserved for extensions are not protected

- Each core designer can implement proprietary extensions
- So far, no registration for opcode extensions

Possible problem:

- One opcode → different instructions depending on vendor
- Binary compatibility can become difficult
- Confusion as to compiler options required to support specific set of extensions

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 (>32*b*)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48*b*
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64*b*
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48*b*
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	≥80*b*

RISC-V challenge: standards conformance

Hardware vendors implement features based on draft specifications

- Example: RISC-V vector instruction extension
  - Version 1.0 only ratified in December 2021
  - C906 / Allwinner D1 implements version 0.7.1 since late 2020
  - Incompatible differences
- Example: RISC-V privileged instruction set
  - Kendryte K210 implements 2017 version of RISC-V privileged specification
  - Adaptations to Linux required to support virtual memory

RISC-V challenge: peripherals

Positive: Standard interrupt controllers defined early on

- Core-local interrupt controller CLINT
- Platform-level interrupt controller PLIC

This took ARM 20 years – vendor-specific interrupt controllers for ARM cores (up to ARM11) complicated porting of system code

However, there is **no standard for peripherals defined**

Example: 16550 UART problem on Allwinner D1 SoC

- Standard 16550 serial driver goes into an infinite loop when interrupts are enabled
- The reason is a residual "busy detect" interrupt generated on the used 16550 IP implementation not available on regular 16550 cores
- Fixes for Linux, Xen, xv6… required!

Vision: the 100% open source computer

Commercial hardware has hidden/undocumented features, e.g.

- Intel management engine
- GPU instruction sets

Open source hardware tries to provide completely open source (and spec) systems

- MNT Reform laptop (https://mntre.com)
- Purism Librem 5 smartphone (https://puri.sm)

Additional bonus: **Repairability** and **sustainability**

Can RISC-V support these efforts?

- Open source and spec SoCs slowly become available
- Open **GPGPU** in development

Also: Open source tooling for IC design

- Verilog/VHDL simulation and synthesis toolchains

MNT Reform ARM-based laptop
https://mntre.com

Purism Librem 5 smartphone
https://puri.sm

Conclusion

RISC-V shows a lot of promises…

- Mature base specifications and software tooling
- Many extensions recently ratified or still in development
- Scalable from microcontroller to high-performance accelerator

- Many different implementations available
  - Lots of choice, but also compatibility problems due to extensions
  - Verification of base ISA and standard extensions

- Numerous small problems remain
  - Not insurmountable, but can be roadblocks for developers

- Vision: a completely open source computer
  - Can it benefit teaching, research and industrial projects?

References

[1] Krste Asanović, *Instruction Sets Should be Free: The Case for RISC-V*, U.C. Berkeley Technical Reports, 2016, Regents of the University of California

[2] David Patterson and Andrew Waterman, *The RISC-V Reader: An Open Architecture Atlas*, Strawberry Canyon, 2017, ISBN-13: 978-0999249116

[3] Andrew Waterman, *Design of the RISC-V Instruction Set Architecture*, UCB Tech Report No. UCB/EECS-2016-1, 2016

[4] Daniel Mangum, *RISC-V Bytes: Privilege levels,* https://danielmangum.com/posts/risc-v-bytes-privilege-levels/

[5] Andrew Waterman, Krste Asanović and John Hauser, *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Document*, Version 20211203 https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

[6] https://github.com/riscvarchive/riscv-cores-list

[7] Jochen Liedtke, *On the realization of huge sparsely occupied and fine grained address spaces*, Oldenbourg, München & Wien 1996, ISBN 3-486-24185-0

[8] Tim Harris, James Larus and Ravi Rajwar, *Transactional Memory*, 2nd edition, Synthesis Lectures on Computer Architecture, Morgan-Claypool 2010, ISBN-13 : 978-1608452354

[9] Yoshihide Sato and Katsumi Maruyama, *LP49: Embedded system OS based on L4 and Plan 9,* Proceedings of the 4th International Workshop on Plan 9, 2006

[10] https://github.com/f9micro/f9-kernel

[11] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, *Plan 9 from Bell Labs*, Computing systems, 8(3), 221-254, 1995

[12] S. Dorward, R. Pike, D. Presotto, D. Ritchie, H. Trickey, P. Winterbottom, *The Inferno operating system*, Bell Labs Technical Journal, 2(1), 5-18, 1997

[13] https://github.com/riscv-non-isa/riscv-arch-test