QEMU Support for the RISC-V Instruction Set Architecture

Sagar Karandikar

sagark@eecs.berkeley.edu

KVM Forum 2016

https://github.com/riscv/riscv-qemu

Berkeley Architecture Research

RISC-V

Outline

- Why RISC-V?
- Benefits of an Open ISA
- RISC-V ISA Basics
- Virtualization Support
- QEMU RISC-V Target Support
- Work in Progress/TODOs for Upstreaming/Future Work

ISAs don’t matter

- Most of the performance and energy of running software on a computer is due to:
  - Algorithms
  - Application code
  - Compilers
  - OS/Runtimes
  - ISA
  - Microarchitecture (core and memory hierarchy)
  - Circuit design
  - Physical design
  - Fabrication process

+ In a *system*, there are also displays, radios, DC/DC converters, sensors, actuators…

Why Instruction Set Architecture Matters

- Why can't Intel sell mobile chips?
  - 99%+ of mobile phones/tablets based on ARM v7/v8 ISA
- Why can't ARM partners sell servers?
  - 99%+ of laptops/desktops/servers based on AMD64 ISA (over 95%+ built by Intel)
- How can IBM still sell mainframes?
  - IBM 360, oldest surviving ISA (50+ years)

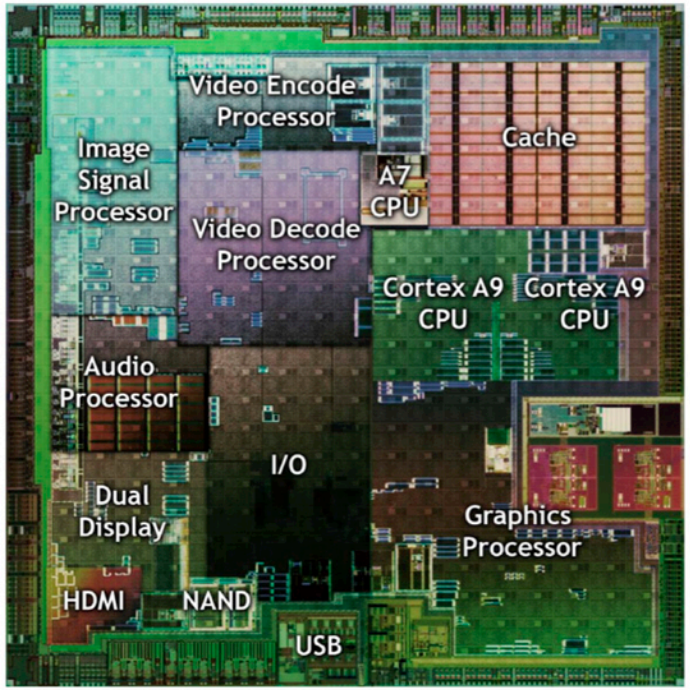
ISA is the most important interface in a computer system

where software meets hardware

RISC-V

Why so many ISAs on an SoC?

- Applications processor (usually ARM)
- Graphics processors
- Image processors
- Radio DSPs
- Audio DSPs
- Security processors
- Power-management processor
- ….
- Apps processor ISA (e.g. ARM) too large for most accelerators
- IP bought from different places, each proprietary ISA
- Home-grown ISA cores
- *Over a dozen ISAs on some SoCs – each with unique software stack*



NVIDIA Tegra SoC

RISC-V

Do we need all these different ISAs?

Must they be proprietary?

What if there were one free and open ISA everyone could use for everything?

ISAs should be Free and Open

- While ISAs may be proprietary for historical or business reasons, there is no good technical reason for the lack of free, open ISAs
  - It’s not an error of omission
  - Nor is it because the companies do most of the software development
  - Neither do companies exclusively have the experience needed to design a competent ISA
  - Nor are the most popular ISAs wonderful ISAs
  - Neither can only companies verify ISA compatibility
  - Finally, proprietary ISAs are not guaranteed to last

Benefits of a Viable Free and Open ISA

- Greater innovation via free-market competition from many core designers, closed-source and open-source
- Shared open core designs, shorter time to market, lower cost from reuse, fewer errors given more eyeballs
- Processors becoming affordable for more devices, which would help expand the Internet of Things (IoTs), which could cost as little as $1
- Software stacks survive for long time upgrade software on systems embedded in concrete 50 years ago
- Make architecture research and education more real with fully open hardware and software stacks

RISC-V

RISC-V Origins

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, it was time for the Computer Science team at UC Berkeley to look at what ISAs to use for their next set of projects
- Obvious choices: x86 and ARM
  - x86 impossible – too complex, IP issues

Intel x86 “AAA” Instruction

- ASCII Adjust After Addition
- AL register is default source and destination
- If the low nibble is > 9 decimal, or the auxiliary carry flag AF = 1, then
  - Add 6 to the low nibble of AL and discard overflow
  - Increment high byte of AL
  - Set CF and AF
- Else
  - CF = AF = 0
- A single-byte instruction

RISC-V Origins

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, it was time for the Computer Science team at UC Berkeley to look at what ISAs to use for their next set of projects
- Obvious choices: x86 and ARM
  - x86 impossible – too complex, IP issues
  - ARM mostly impossible – complex, IP issues
- UC Berkeley started a “3-month project” during summer of 2010 to develop their own clean-slate ISA
  - Andrew Waterman, Yunsup Lee, Dave Patterson, Krste Asanovic principal designers

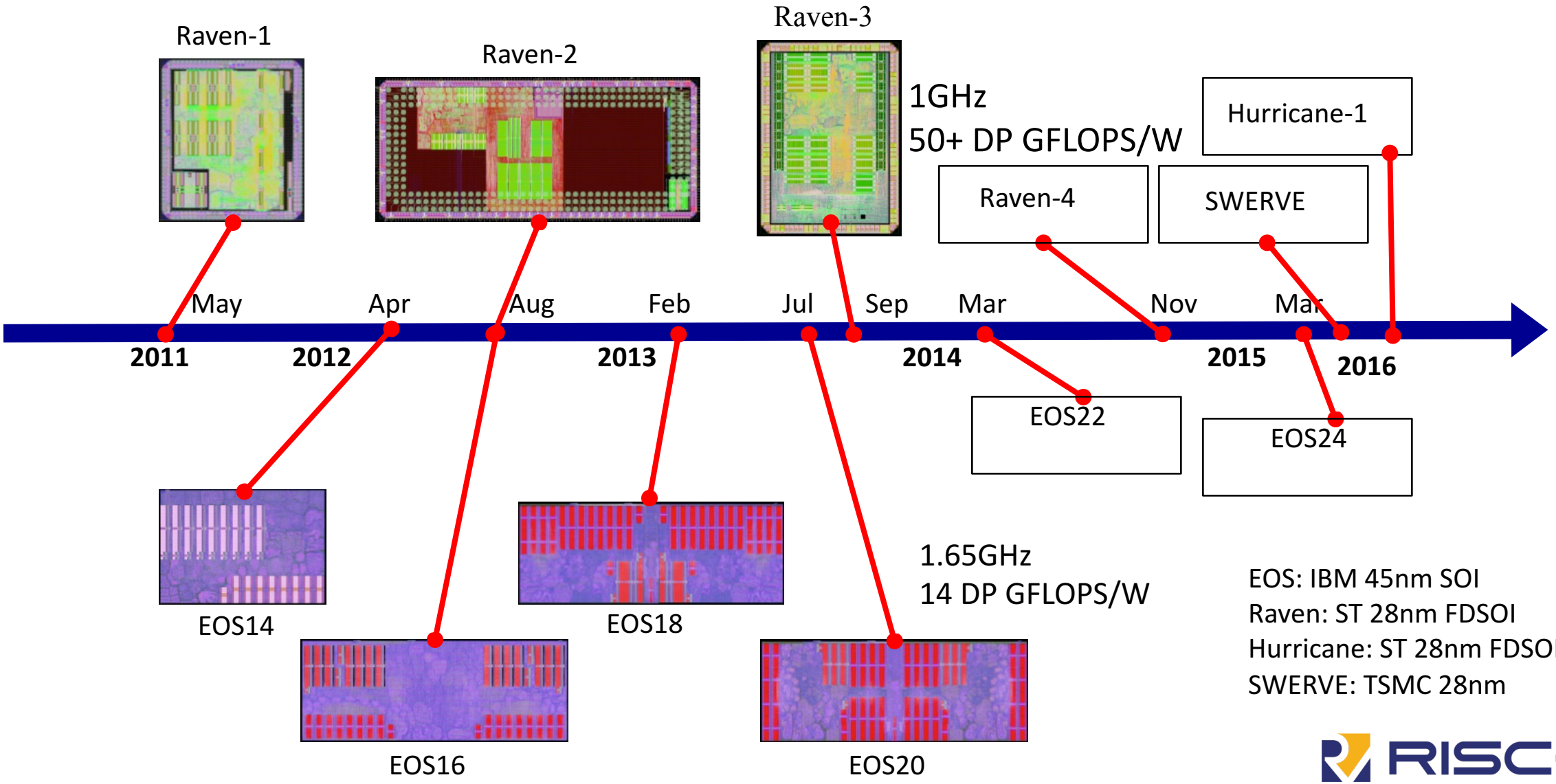
RISC-V Background (cont'd)

- Four years later, in May of 2014, UC Berkeley released frozen base user spec
  - Many chip tapeouts and several research publications along the way
- The name RISC-V (pronounced *risk-five*), was chosen to represent the fifth major RISC ISA design effort at UC Berkeley
  - RISC-I, RISC-II, SOAR, and SPUR were the first four with the original RISC-I publications dating back to 1981
- In August 2015, articles of incorporation were filed to create a non-profit RISC-V Foundation to govern the ISA

RISC-V is NOT an Open-Source Processor

- RISC-V is an ISA specification – NOT an open-source processor core
- Most of the cost of chip *design* is in software, so we want to make sure software can be reused across many chip designs
- The Foundation will encourage both open-source and proprietary implementations of the RISC-V ISA specification

UC Berkeley RISC-V Cores:



EOS: IBM 45nm SOI
Raven: ST 28nm FDSOI
Hurricane: ST 28nm FDSOI
SWERVE: TSMC 28nm

Industrial Support – Platinum Founding Members

RISC-V

Industrial Support – Gold, Silver, Auditor Founding Members

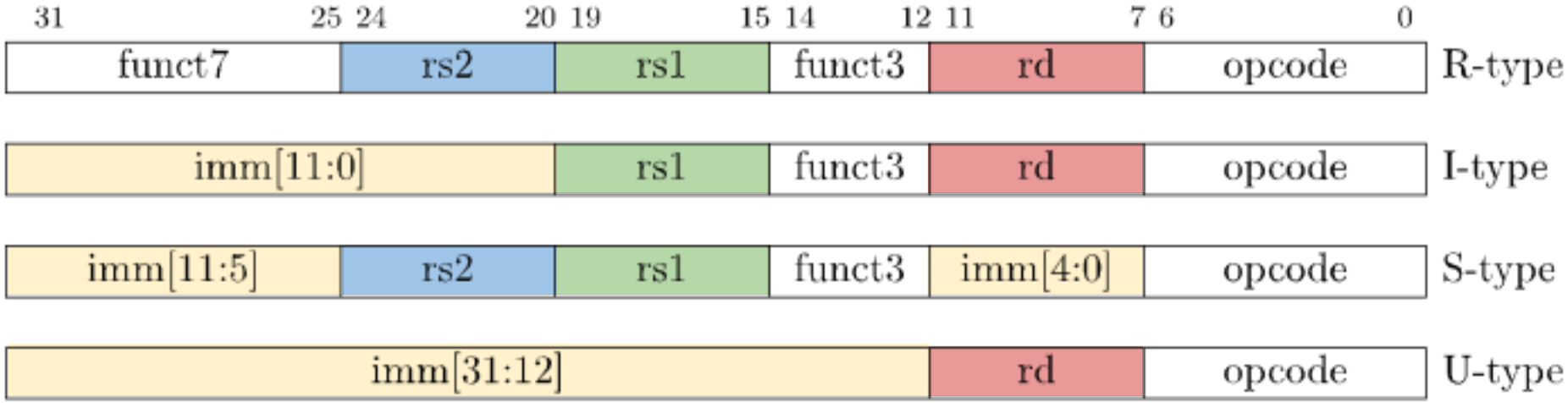
RISC-V

The RISC-V ISA

- RV32, RV64, RV128 variants for 32b, 64b, 128b address spaces defined
- Base ISA only <50 integer instructions, but supports compiler, linker, OS, etc.
- Extensions provide full general-purpose ISA, including IEEE-754/2008 floating-point
- Comparable ISA-level metrics to other RISCs
- Designed for extension, customization
- Twelve 64-bit silicon prototype implementations completed at Berkeley so far (45nm, 28nm)

RISC-V

RISC-V Standard Base ISA Details



- 32-bit, fixed-width, naturally aligned instructions
- 31 integer registers x1-x31, plus x0 zero register
- rd/rs1/rs2 in fixed location, no implicit registers
- Immediate field (instr[31]) always sign-extended
- Floating-point adds f0-f31 registers plus FP CSR, also fused mul-add four-register format
- Designed to support PIC and dynamic linking

RISC-V

RV64G Definition

- G = I, M, A, F, D
  - `I` = Base **I**nteger ISA
  - `M` = Standard Integer **M**ultiplication/Division Extension
  - `A` = Standard **A**tomics Extension
  - `F` = Standard Single-precision **F**loating-point extension
  - `D` = Standard **D**ouble-precision floating-point extension
- This is the standard, general purpose version of the ISA, what is implemented in QEMU

RISC-V Privileged Specification

- Four Privilege Modes: User, Supervisor, Hypervisor, Machine
- Machine Mode required
- Common: Provide M, S, U for running Unix-like OSes (QEMU does this)
- Virtual Memory Architecture designed to support current Unix-like operating systems
- Sv39 (RV64)
  - Demand-paged 39-bit virtual-address spaces
  - 3-level page table
  - 4 KiB pages, 2 MiB megapages, 1 GiB gigapages
- Also Sv32 (RV32) and Sv48, Sv57, Sv64 (RV64)

RISC-V Virtualization

- ISA designed with virtualization in-mind from the beginning, even when only using U + S + M modes
  - *“The privileged architecture is designed to simplify the use of classic virtualization techniques, where a guest OS is run at user-level, as the few privileged instructions can be easily detected and trapped.”* – RISC-V Privileged Architecture v1.9 Manual
- Avoiding Some Classical Virtualization Pitfalls…

Handling Sensitive, but Unprivileged Instructions

- In x86, for the original VMware – "*Table II lists the [19] instructions of the x86 architecture that unfortunately violated Popek and Goldberg's rule and hence made the x86 non-virtualizeable*"[1]
- In RISC-V, no "hidden" privileged state reads/writes
- Small set of privileged instructions that can modify space of privileged state (Control Status Registers, or CSRs)

1. E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, and E. Y. Wang. 2012. **Bringing Virtualization to the x86 Architecture with the Original VMware Workstation**. *ACM Trans. Comput. Syst.* 30, 4, Article 12 (November 2012), 51 pages.

Tracking Changes in Virtual Machine Memory

- In x86, for the original VMware – *"… privileged hardware registers contain the address of segment descriptor tables and page tables … but regular load and store instructions can access these structures in memory."*[1]
- In RISC-V, still use regular loads/stores to modify memory management state
- Privileged `SFENCE.VM` instruction required by spec. after modifying memory management state

1. E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, and E. Y. Wang. 2012. **Bringing Virtualization to the x86 Architecture with the Original VMware Workstation**. *ACM Trans. Comput. Syst.* 30, 4, Article 12 (November 2012), 51 pages.

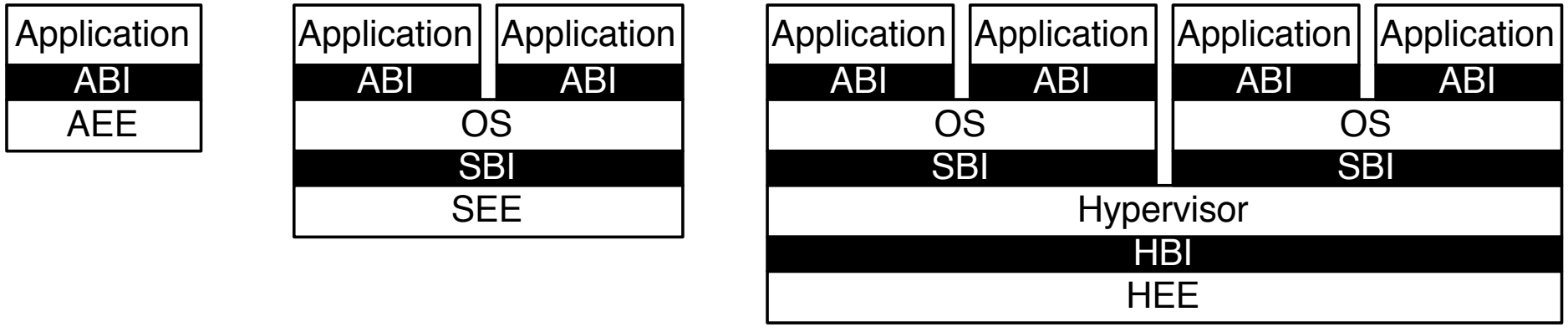
RISC-V

Virtualizing Segmentation

- In x86, for the original VMware – Complicated interactions between segment descriptor tables and segment registers, with visible and hidden fields. Hidden pieces updated by instructions or faults. Causes problems with extra faults introduced by a VMM.[1]
- In RISC-V, no x86-style segmentation
  - Limited base-and-bounds mode with 2 “segments”
  - Most software will use paging instead

1. E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, and E. Y. Wang. 2012. **Bringing Virtualization to the x86 Architecture with the Original VMware Workstation**. *ACM Trans. Comput. Syst.* 30, 4, Article 12 (November 2012), 51 pages.

RISC-V Virtualization Stacks



- Provide clean split between layers of the software stack
- Application communicates with Application Execution Environment (AEE) via Application Binary Interface (ABI)
- OS communicates via Supervisor Execution Environment (SEE) via System Binary Interface (SBI)
- Hypervisor communicates via Hypervisor Binary Interface (HBI) to Hypervisor Execution Environment (HEE)
- All levels of the ISA designed to support virtualization

RISC-V Hypervisor Specification - WIP

- Current privileged design can have an M-mode monitor that provides physical resource partitioning, can act as simple hypervisor
- Upcoming Hypervisor Extension Specification for “full” Hypervisors
  - Right now, an empty slot in the privileged specification
- Want to get involved?
  - Hypervisor Specification Draft will make the rounds soon on isa-dev@groups.riscv.org mailing list

The RISC-V Ecosystem

- Software Tools
  - GCC/glibc/GDB
  - LLVM/Clang
  - Linux
  - Yocto
  - Verification Suite
- Hardware Tools
  - Zynq FPGA Infrastructure
  - Chisel

- Software Implementations
  - Spike, “Golden-standard” ISA Simulator
  - ANGEL, JavaScript ISA Simulator
  - **QEMU**
- Hardware Implementations
  - Rocket Chip Generator
    - RV64G single-issue in-order pipeline
  - Sodor Processor Collection
  - BOOM (Berkeley Out-of-Order Machine)

github.com/riscv and github.com/ucb-bar

RISC-V

RISC-V Target support for QEMU

- Maintained at https://github.com/riscv/riscv-qemu
- QEMU full-system emulation
- QEMU on modern x86 is currently the fastest RISC-V implementation
- A big help in RISC-V software development

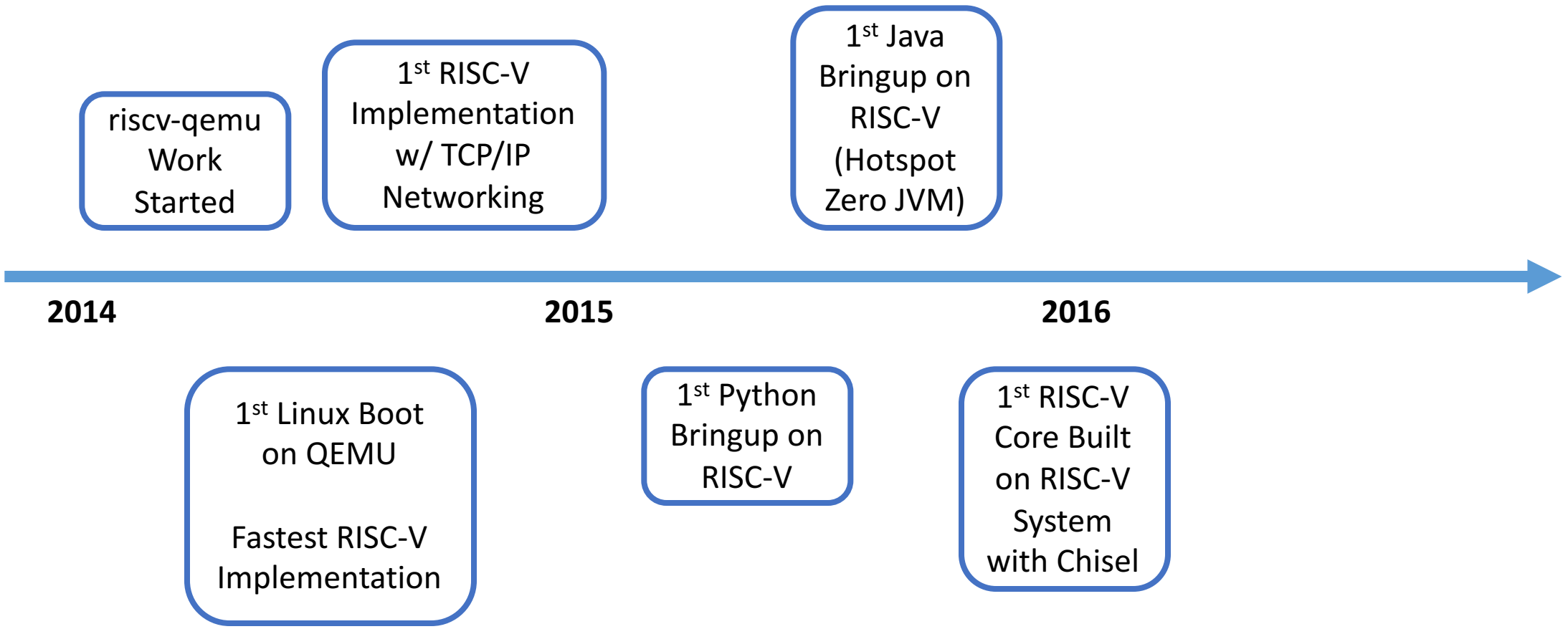
Spike

QEMU

```
This side will boot Linux in SPIKE


skarandikar@a8:/scratch/sagark$ spike bbl_vmlinux_initramfs
```

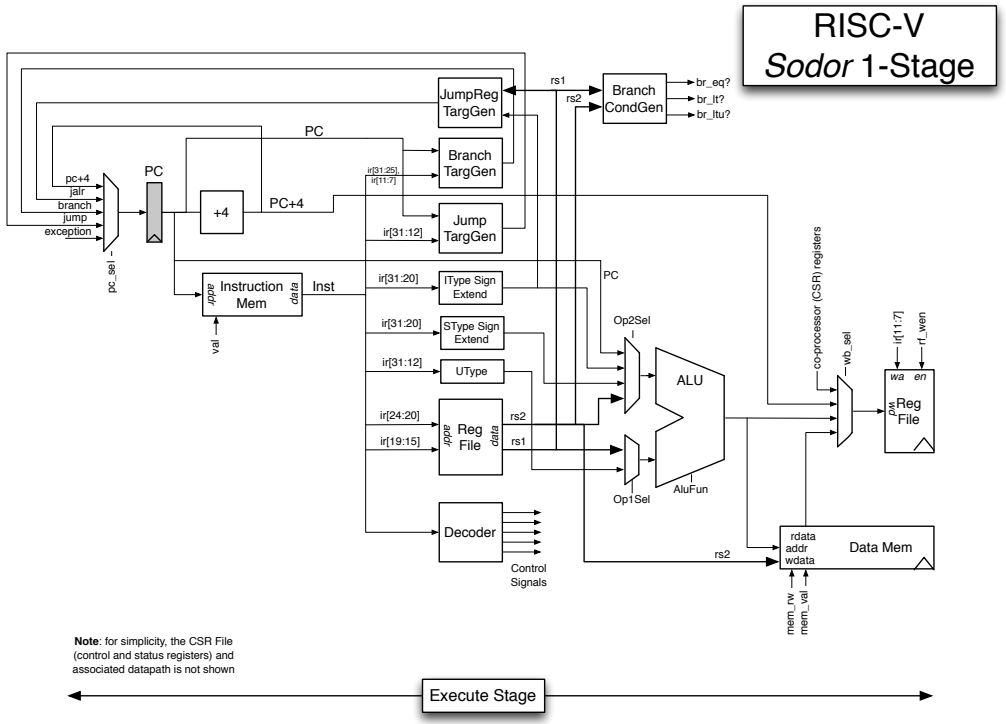
```
This side will boot Linux in QEMU


skarandikar@a8:/scratch/sagark$ qemu-system-riscv -kernel bbl_vmlinux_initram
fs -nographic
```

Timeline of RISC-V “Firsts”



… all on QEMU!

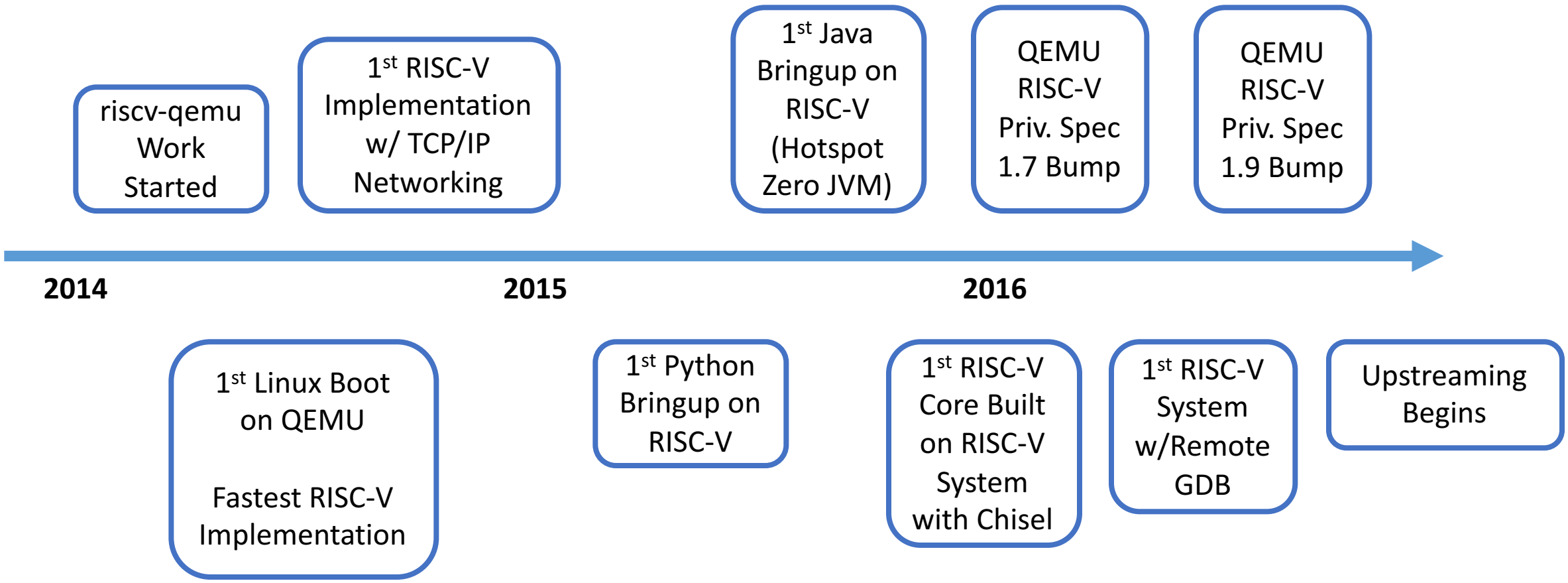
RISC-V Chip Development on RISC-V



Build screenshot courtesy Michael Knyszek

```
root@qemuriscv:~/riscv-sodor# make
make -C emulator/rv32_1stage/
make[1]: Entering directory '/home/root/riscv-sodor/emulator/rv32_1stage'
cd /home/root/riscv-sodor && java -Xmx200M -Xss8M -XX:MaxPermSize=128M -jar /home/r
zed"
[info] Loading project definition from /home/root/riscv-sodor/project
[info] Set current project to riscv-sodor (in build file:/home/root/riscv-sodor/)
[info] Set current project to rv32_1stage (in build file:/home/root/riscv-sodor/)
[info] Updating {file:/home/root/riscv-sodor/}common...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] downloading http://repo1.maven.org/maven2/edu/berkeley/cs/chisel_2.10/2.2.27
[info]  [SUCCESSFUL ] edu.berkeley.cs#chisel_2.10;2.2.27!chisel_2.10.jar (9505ms)
[info] downloading http://repo1.maven.org/maven2/org/scala-lang/scala-library/2.10.
[info]  [SUCCESSFUL ] org.scala-lang#scala-library;2.10.4!scala-library.jar (31686m
[info] downloading http://repo1.maven.org/maven2/org/scala-lang/scala-reflect/2.10.
[info]  [SUCCESSFUL ] org.scala-lang#scala-reflect;2.10.4!scala-reflect.jar (16265m
[info] Done updating.
[info] Updating {file:/home/root/riscv-sodor/}rv32_1stage...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 9 Scala sources to /home/root/riscv-sodor/common/target/scala-2.10
[warn] there were 1 deprecation warning(s); re-run with -deprecation for details
[warn] there were 72 feature warning(s); re-run with -feature for details
[warn] two warnings found
[info] Compiling 7 Scala sources to /home/root/riscv-sodor/rv32_1stage/target/scala
[warn] there were 30 feature warning(s); re-run with -feature for details
[warn] one warning found
[info] Running Sodor.elaborate --noIoDebug --wio --backend c --debug --ioDebug --ta

    Sodor Tile: creating Asynchronous Scratchpad Memory of size 2048 kB
```

Timeline of RISC-V “Firsts”



… all on QEMU!

RISC-V

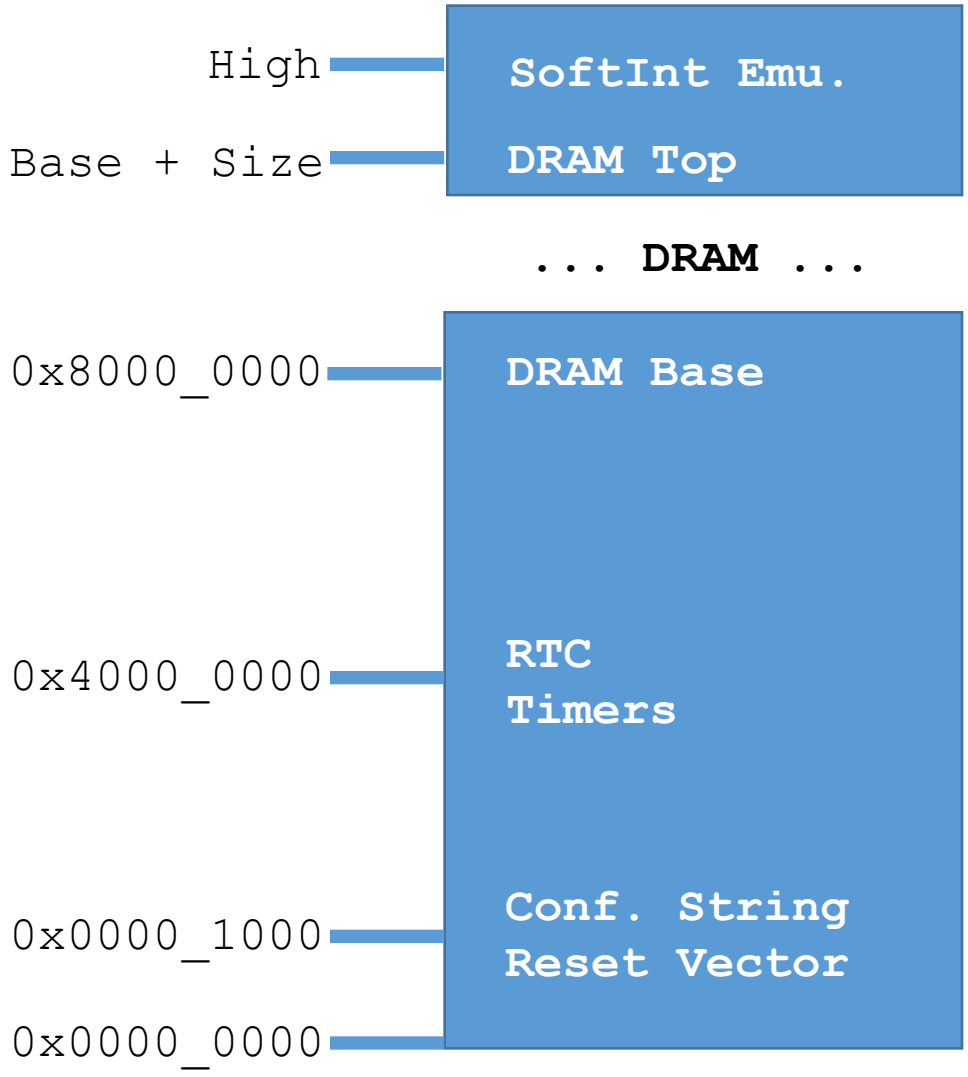
RISC-V Target support for QEMU

- RISC-V support started in 2014, evolves as the ISA does
  - Supports RV64IMAFD (“RV64G”) Full-system emulation
  - User ISA support largely unchanged since then (currently v2.0)
  - Privileged ISA nearing standardization (currently v1.9)
    - Future Priv. Spec upgrades to QEMU much simpler due to structural similarity of priv. mode emulation code with Spike
      - Pre-1.7 -> 1.7 bump, ~1 month
      - 1.7 -> 1.9 bump, 3 days
- I/O currently limited to “Host-Target Interface” (HTIF) devices
  - Enough to boot Linux, interact through console
  - Other devices previously shoehorned in (networking, displays, consoles)
    - Mainly waiting on platform standardization and software support

RISC-V

hw/riscv/riscv_board.c

- Reference “board” designed to match Spike
- Provides simple hardware, config:
  - HTIF-based console (simple, non-standard console device for early boot)
  - “Loopback” Software Interrupt Emulation
  - RTC/Timers compliant with RISC-V v1.9 privileged specification
  - Reset Vector (Boot ROM)
  - Configuration String

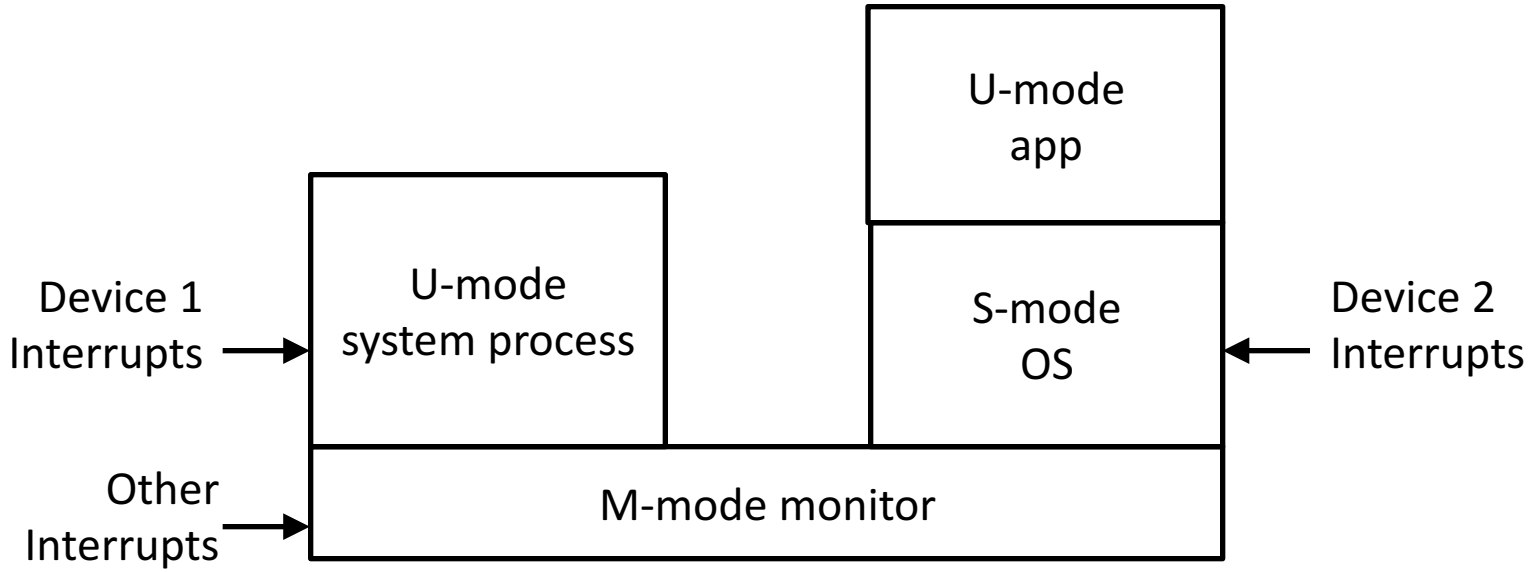


I/O: HTIF (old), Debug (new)

- Host-Target Interface (HTIF) is a relic of Berkeley Test Chips
- Two 64-bit registers, fromhost and tohost
- Formerly provided network, block device, console, now used only for console, signaling test completion
- Debugging: HTIF being phased out on real hardware, replaced with draft Debug Unit specification (will be standardized)
- I/O: Real hardware/software simulators will also phase out HTIF and move to standard devices as software support progresses

Software Stack inside RISC-V QEMU

- M-mode runs secure boot and monitor (currently bbl)
- S-mode runs OS (OS always runs virtualized)
- U-mode runs application on top of OS or M-mode



Boot Up

Binary supplied to QEMU contains:

- `bbl` – "Berkeley BootLoader" performs Machine-Mode management of the system, exposes SBI to OSes
- `vmlinux` – Linux kernel as payload for `bbl`
  - Includes a ramdisk for rootfs (currently limited device support)

System boots into hardcoded boot rom, jumps to `bbl`,
`bbl` initializes the system in M-mode, sets up Supervisor Execution Environment (SEE), then loads and runs supplied kernel, e.g. Linux

Testing/Debugging

- The usual GDB, brute force, etc
  - Passes the riscv-tests standard test suite
  - Boots Linux
    - With reasonably large software stacks on top – Python, Java, etc.
- Fuzz testing against the “Golden Standard”

Fuzz Testing with `riscv-torture`

- Generates a random sequence of instructions based on supplied parameters (% of mem instructions, floating point instructions, integer instructions, etc.)
- Runs code on Spike and other implementation of your choice
  - Spike is the “Golden Reference” RISC-V Simulator, written by the authors of the RISC-V Specs
- Dump register state at the end and compare
- On failure, binary search to pinpoint instruction where things go wrong
- Scripts for running riscv-torture on QEMU available at https://github.com/sagark/riscv-qemu-torture
- Accumulated 384 hours of failure-free testing since August Priv. 1.9 update!

`target-riscv` SLoC*

- ARM - 45,438
- MIPS - 37,501
- x86 - 30,437
- RISC-V - 5,074

Work in Progress/TODOs

- Functionality: Standard device support (combo of QEMU + Linux)
- Upstreaming! – planning to start in mid-September
  - Submitted a giant patch in February as proof-of-existence for gauging interest in GSoC
  - Some cleanup based on giant-patch feedback done (e.g. use built-in FPU)
  - Latest RISC-V Privileged Spec. v1.9 bump done
  - TODO:
    - More cleanup based on giant-patch feedback, checkpatch
    - Rebase to master (currently v2.5.0)
    - Small, logical patches
- Future:
  - Support other ISA variants, like RV32, Compressed ISA, QEMU Linux-User Mode

Thanks!
Questions?

Sagar Karandikar

sagark@eecs.berkeley.edu

KVM Forum 2016

https://github.com/riscv/riscv-qemu

Berkeley Architecture Research

RISC-V