



# Tutorial Introduction

Krste Asanović  
**krste@eecs.berkeley.edu**

**<http://www.riscv.org>**

RISC-V Tutorial, HPCA, SFO Marriot  
February 8, 2015





## ISAs don't matter

Most of the performance and energy running software on a computer is due to:

- Algorithms
- Application code
- Compiler
- OS/Runtimes
- ISA (Instruction Set Architecture)
- Microarchitecture (core + memory hierarchy)
- Circuit design
- Physical design
- Fabrication process

In a *system*, there's also displays, radios, DC/DC convertors, sensors, actuators, ...



## ISAs do matter

- Most important interface in computer system
- Large cost to port and tune all ISA-dependent parts of a modern software stack
- Large cost to recompile/port/QA all supposedly ISA-independent parts of stack
- Proprietary closed-source, don't have code
- Bit rot, lose ability to compile own source code
- Lost your own source code
  
- Most of the cost of developing a new chip is developing software for it
- Most big new chips have several ISAs...



So...

If choice of ISA doesn't have much impact on system energy/performance, and it costs a lot to use different ones,

why isn't there just one industry-standard ISA?



## 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 from Viable Freely 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, transparency makes it difficult for government agencies to add secret trap doors.
- **Processors becoming affordable for more devices**, which would help expand the Internet of Things (IoT), 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



## What Style of ISA?

- If ISA doesn't matter, then pick an easy-to-implement, efficient ISA
- Seymour Cray's 50-year-old load-store design, refined by RISC research 30+ years ago, still a good choice



## ISA innovation over last 30 years?

- Load-store ISAs predate Moore's Law
- Very little (nothing?) stuck for general-purpose code, failures:
  - CISC (microcode worse than instruction cache in VLSI)
  - Stack machines (compiler can target registers)
  - VLIW (dynamic predictors beat static scheduling)
- Industry has reconverged on simpler RISC ideas:
  - ARM v8, MIPS r6, both similar ideas to RISC-V
- Specialized Accelerators? Only way to improve performance until post-CMOS (20-30 years away)
  - G. Estrin, "Organization of computer systems: the fixed plus variable structure computer," Proc. WJCC, 1960
  - GPUs – bad version of Cray vectors





## Other Open ISAs

- **SPARC V8** - To its credit, Sun Microsystems made SPARC V8 an IEEE standard in 1994. Sun, Gaisler offered open-source cores. ISA now owned by Oracle.
- **OpenRISC** - GNU open-source effort started in 1999, based on DLX. 64-bit ISA was in progress.
- **Open Processor Foundation** - Patents expiring on Hitachi SH architecture, so open-sourcing tools and implementations
- **Lattice Micro 32 (LM32)** – Simple FPGA soft core.
- All RISC ISAs



## RISC-V Background

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, time to look at ISA for 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 low nibble of AL and discard overflow
  - Increment high byte of AL
  - Set CF and AF
- Else
  - $CF = AF = 0$
- Single byte instruction



## RISC-V Background

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, time to look at ISA for next set of projects
- Obvious choices: x86 and ARM
- x86 impossible – too complex, IP issues
- ARM mostly impossible - complex, IP issues
- So we started “3-month project” in summer 2010 to develop our own clean-slate ISA
- Four years later, we released frozen base user spec
  - But also many tapeouts and several research publications along the way



## 2015 RISC-V Project Goal

Become the industry-standard ISA for  
all computing devices



## RISC-V is NOT an Open-Source Processor

- RISC-V is an ISA specification
- Will be expanding to be specifications for SoCs including I/O and accelerators
- Most of cost of chip design is in software, so want to make sure software can be reused across many chip designs
- Want to encourage both open-source and proprietary implementations of the RISC-V ISA specification



## RISC-V Base Plus Standard Extensions

- Three base integer ISAs, one per address width
  - RV32I, RV64I, RV128I
  - Only 40 hardware instructions needed
- Standard extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations (AMOs + LR/SC)
  - F: Single-precision floating-point
  - D: Double-precision floating-point
  - G = IMAFD, “General-purpose” ISA
  - Q: Quad-precision floating-point
- All the above are a fairly standard RISC encoding in a fixed 32-bit instruction format
- Now frozen



# RISC-V Standard Base ISA Details

31	25 24	20 19	15 14	12 11	7 6	0	
funct7	rs2	rs1	funct3	rd	opcode		R-type
imm[11:0]		rs1	funct3	rd	opcode		I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode		S-type
imm[31:12]				rd	opcode		U-type

- 32-bit fixed-width, naturally aligned instructions
- 31 integer registers x1-x31, plus x0 zero register
- No implicit registers, rs1/rs2/rd in fixed location
- Floating-point adds f0-f31 registers plus FP CSR, also fused mul-add four-register format
- Designed to support PIC and dynamic linking





## “A”: Atomic Operations Extension

Two classes:

- Atomic Memory Operations (AMO)
  - Fetch-and-op, op=ADD,OR,XOR,MAX,MIN,MAXU,MINU
- Load–Reserved/Store Conditional
  - With forward progress guarantee for short sequences
- All atomic operations can be annotated with two bits (Acquire/Release) to implement release consistency or sequential consistency



## Why do SoCs have so many ISAs?

- Applications processor (usually ARM)
  - Graphics processors
  - Image processors
  - Radio DSPs
  - Audio DSPs
  - Security processors
  - Power management processor
- 
- IP bought from different places, each proprietary ISA
  - Home-grown ISA cores
  - Apps processor ISA too huge for base accelerator ISA
  - Rebuild whole software stack for each ISA to speed up <1% of the total code!!!



# Variable-Length Encoding

xxxxxxxxxxxxxxxxaa			16-bit (aa ≠ 11)
xxxxxxxxxxxxxxxx		xxxxxxxxxxxxbbb11	32-bit (bbb ≠ 111)
···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxxx011111	48-bit
···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxxx011111	64-bit
···xxxx	xxxxxxxxxxxxxxxx	xnnnxxxxx111111	(80+16*nnn)-bit, nnn≠111
···xxxx	xxxxxxxxxxxxxxxx	x111xxxxx111111	Reserved for ≥192-bits

Byte Address:      base+4                                  base+2                                  base

- Extensions can use any multiple of 16 bits as instruction length
- Branches/Jumps target 16-bit boundaries even in fixed 32-bit base

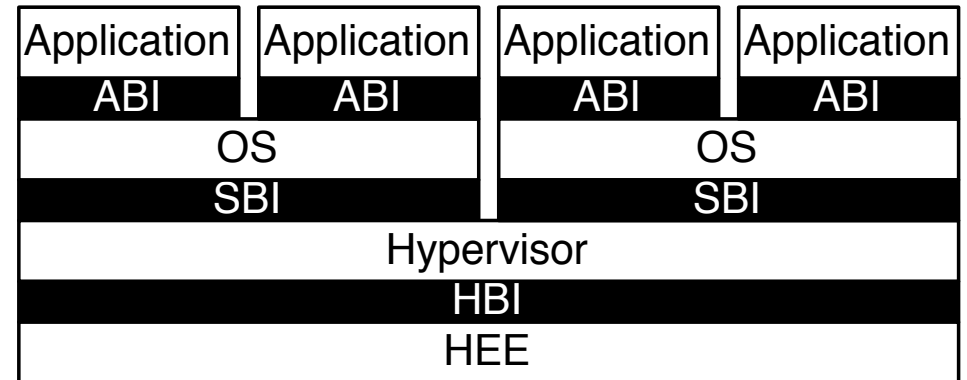
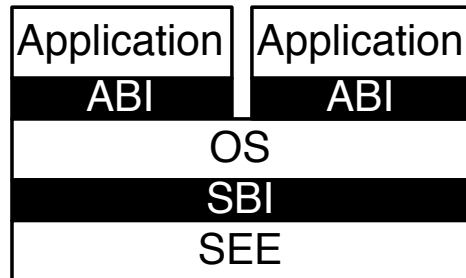
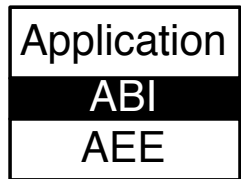


## “C”: Compressed Instruction Extension

- Compressed code important for:
  - low-end embedded to save static code space
  - high-end commercial workloads to reduce cache footprint
- Standard extension (still in draft, not frozen) adds 16-bit compressed instructions
  - 2-address forms with all 32 registers
  - 3-address forms with most frequent 8 registers
- Each C instruction expands to a single base ISA instruction
- All original 32-bit instructions retain encoding but now can be 16-bit aligned
- Approximately 25% reduction in code size, with performance improvement from reduced I\$ misses



# RISC-V Privileged Architecture



- 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 to Hypervisor Execution Environment
- All levels of ISA designed to support virtualization



# RISC-V Hardware Abstraction Layer

Application
ABI
AEE
HAL
Hardware

Application	Application
ABI	ABI
OS	
SBI	
SEE	
HAL	
Hardware	

Application	Application	Application	Application
ABI	ABI	ABI	ABI
OS		OS	
SBI		SBI	
Hypervisor			
HBI			
HEE			
HAL			
Hardware			

- Execution environments communicate with hardware platforms via Hardware Abstraction Layer (HAL)
- Details of execution environment and hardware platforms isolated from OS/Hypervisor ports



## Four Supervisor Architectures

- Mbare
  - Bare metal, no translation or protection
- Mbb
  - Base and bounds protection
- Sv32
  - Demand-paged 32-bit virtual address spaces
- Sv43
  - Demand-paged 43-bit virtual address spaces
- Designed to support current popular operating systems
- Where's the draft spec????
  - This is taking some time, very soon now.
  - It won't be finalized for a while, since we need feedback



▪ **Documentation**

- User-Level ISA Spec v2
- Reviewing Privileged ISA

▪ **Software Tools**

- GCC/glibc/GDB
- LLVM/Clang
- Linux
- Yocto
- Verification Suite

▪ **Hardware Tools**

- Zynq FPGA Infrastructure
- Chisel

▪ **Software Implementations**

- ANGEL, JavaScript ISA Sim.
- Spike, In-house ISA Sim.
- QEMU

▪ **Hardware Implementations**

- Rocket Chip Generator
  - RV64G single-issue in-order pipe
- Sodor Processor Collection





## RISC-V Outside Berkeley

- Adopted as “standard ISA” for **India**. IIT-Madras building 6 different open-source cores, from microcontrollers to servers
- **Bluespec** Inc. developing RISC-V cores for use in tightly integrated hardware/software IP blocks
- **LowRISC** project based in Cambridge, UK producing open-source RISC-V based SoCs. Led by one of the founders of Raspberry Pi, and privately funded
- First commercial RISC-V cores have already shipped!
- Multiple commercial silicon implementations for sale later this year



## RISC-V Foundation

- In progress, establishing a non-profit RISC-V foundation *“to standardize, protect, and promote the free and open RISC-V instruction set architecture and its hardware and software ecosystem for use in all computing devices.”*
- Expect to have RISC-V foundation up and running by time of second RISC-V workshop in Berkeley, summer 2015
- Sign up on mailing lists or twitter at [riscv.org](http://riscv.org) to get announcements



## HPCA/CGO/PPoPP Tutorial Goals

- Bring academic community up to speed on RISC-V developments and tools ecosystem
- Agenda:
  - Intro, Krste Asanovic
  - “Rocket Chip” SoC generator overview, Yunsup Lee
  - RISC-V software stack overview, Palmer Dabbelt
  - Break
  - RISC-V software bootcamp, Albert Ou
  - Rocket bootcamp, Colin Schmidt



## RISC-V Research Project Sponsors

- DoE Isis Project
- DARPA PERFECT program
- DARPA POEM program (Si photonics)
- STARnet Center for Future Architectures (C-FAR)
- Lawrence Berkeley National Laboratory
- Industrial sponsors (ParLab + ASPIRE)
  - Intel, Google, Huawei, LG, NEC, Microsoft, Nokia, NVIDIA, Oracle, Samsung