

## ISA Architecture

# RISC-V

Dr Jeff Drobman

website → [drjeffsoftware.com/classroom.html](https://drjeffsoftware.com/classroom.html)

email → [jeffrey.drobman@csun.edu](mailto:jeffrey.drobman@csun.edu)

# Index

- ❖ History → slide 3
- ❖ BSD → slide 31
- ❖ RISC-V ISA → slide 11
- ❖ R5 Foundation → slide 29
- ❖ Benchmarks → slide 32
- ❖ Cores → slide 35
- ❖ Design Tools → slide 43
- ❖ Software → slide 48
- ❖ BSD → slide 51
- ❖ RISC-V Cores → slide 56

# RISC-V



## RISC-V History

# RISC-V History



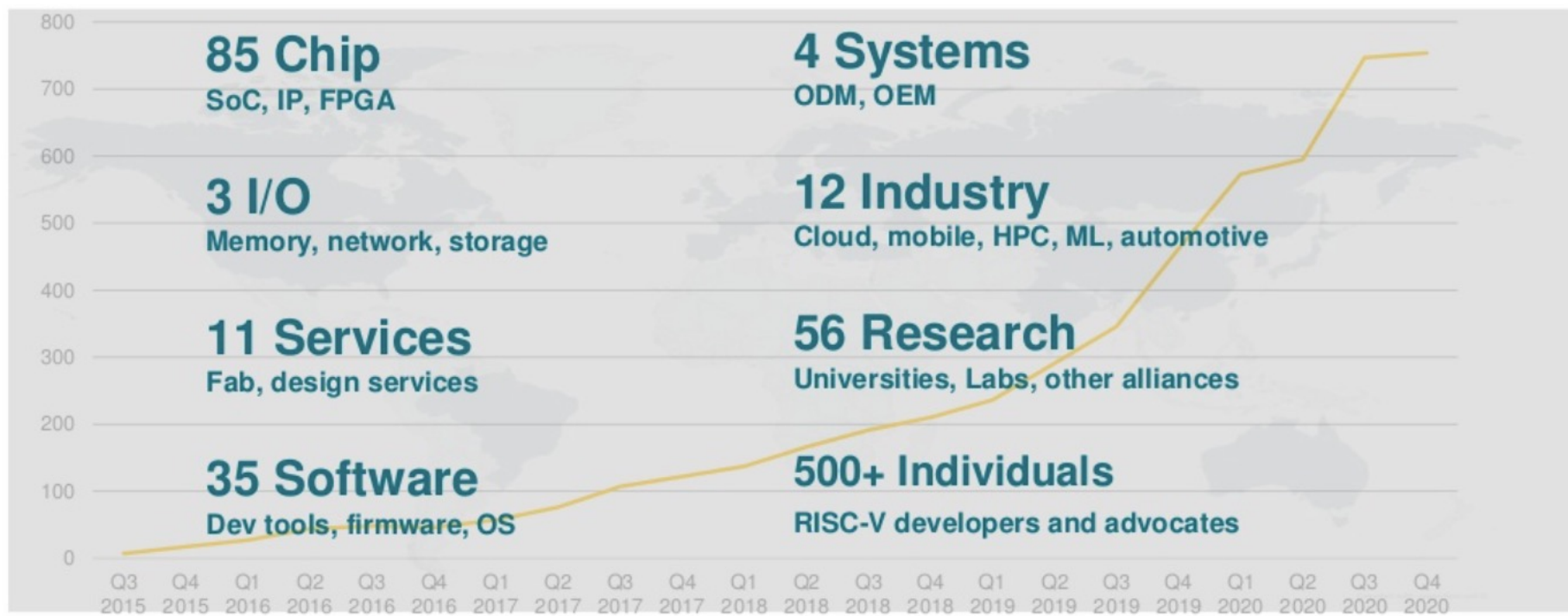
RISC-V is a high-quality, license-free, royalty-free instruction set architecture (ISA)

- 5<sup>th</sup> generation RISC design from UC Berkeley (started in 2010)
- Appropriate for all levels of computing system, from microcontrollers to supercomputers
- Multiple open-source and proprietary core implementations
- Custom extensions are encouraged
- Supported by growing software ecosystem
  - RISC-V support for binutils, gcc, newlib, glibc, Zephyr, Linux, FreeBSD mainlined
- Standard maintained by the non-profit RISC-V foundation



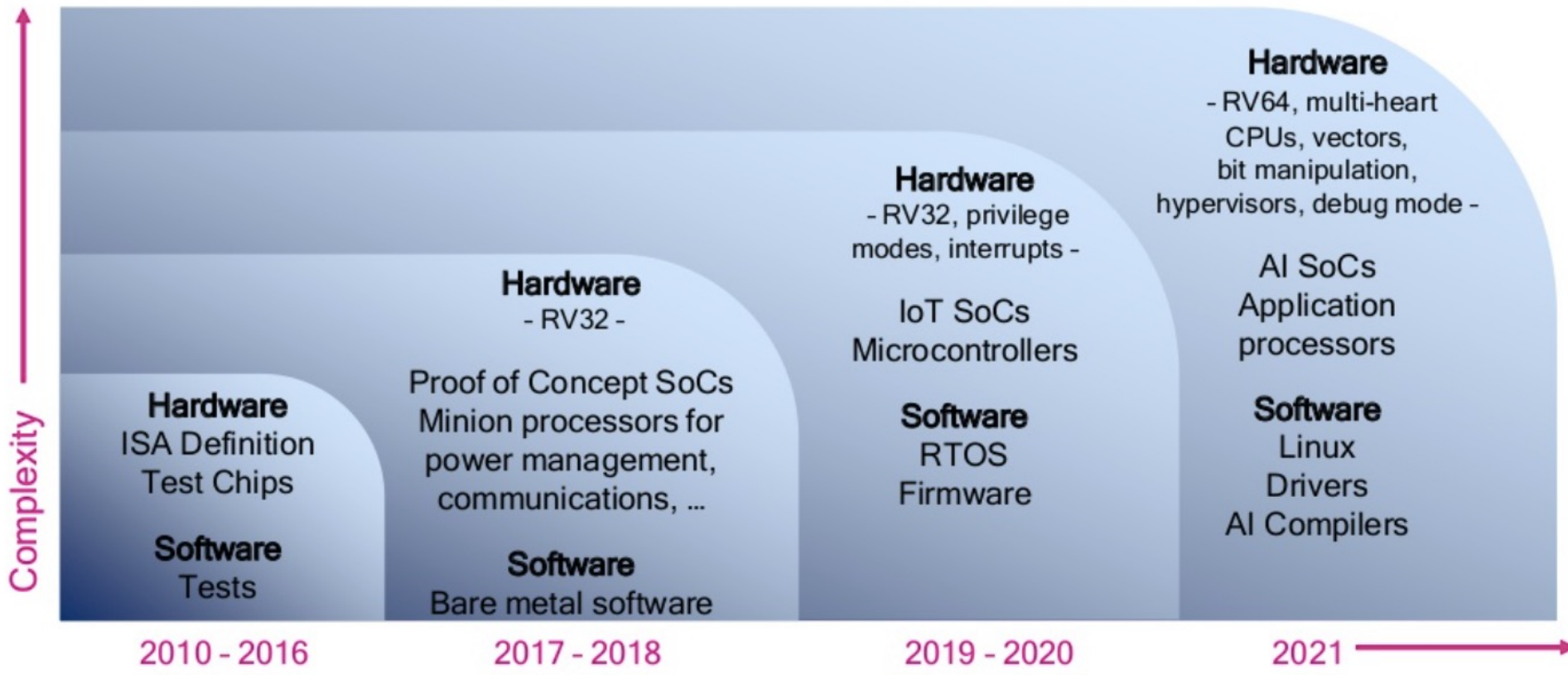
# RISC-V History

## More than 750 RISC-V Members across 50 Countries



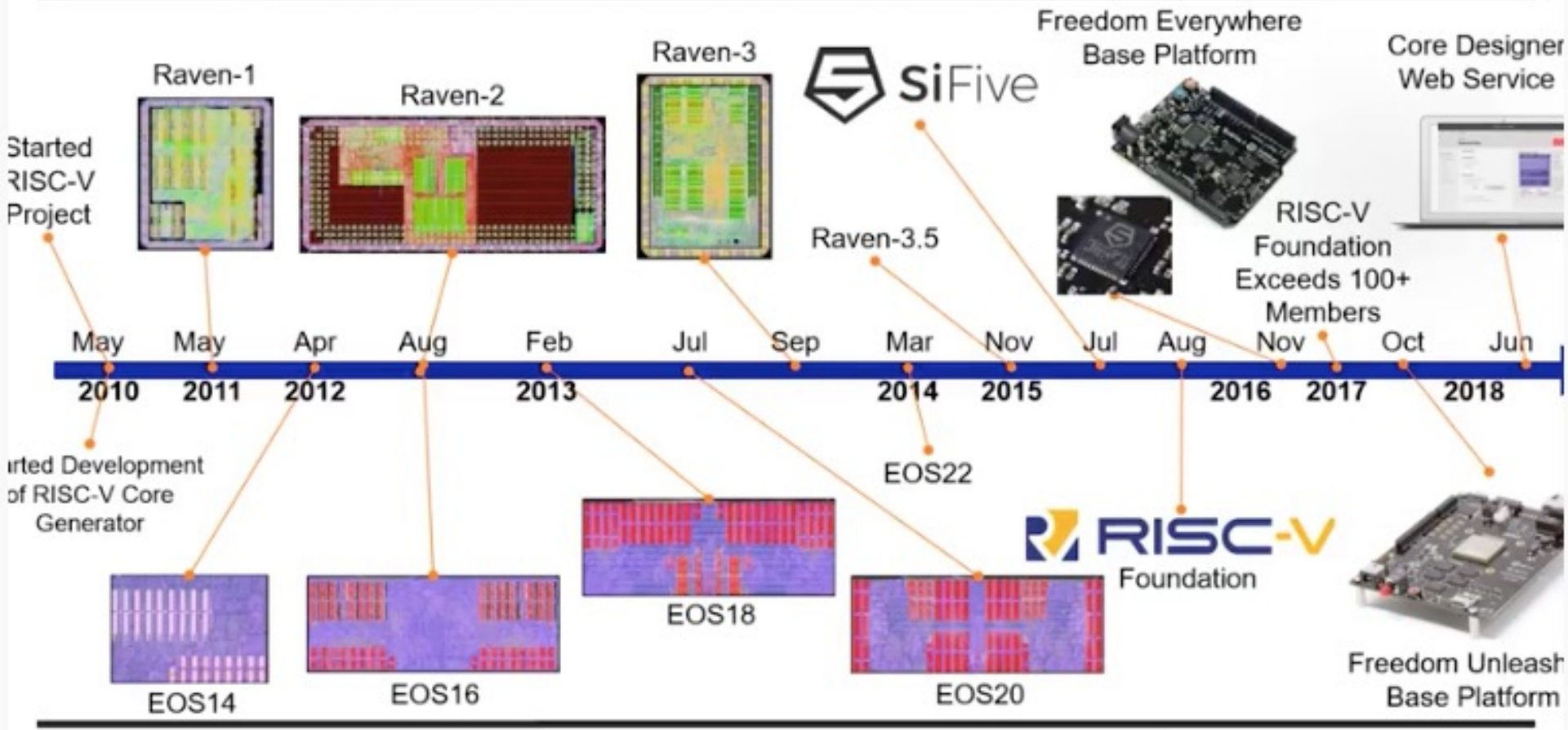
# RISC-V History

## Industry innovation on RISC-V



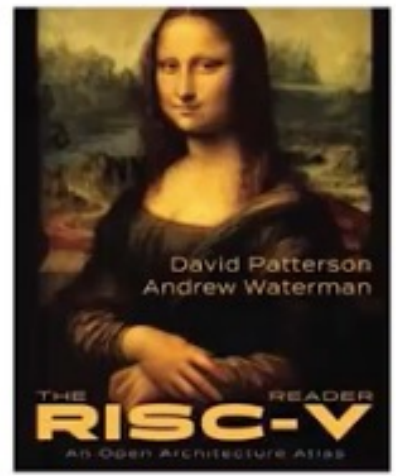
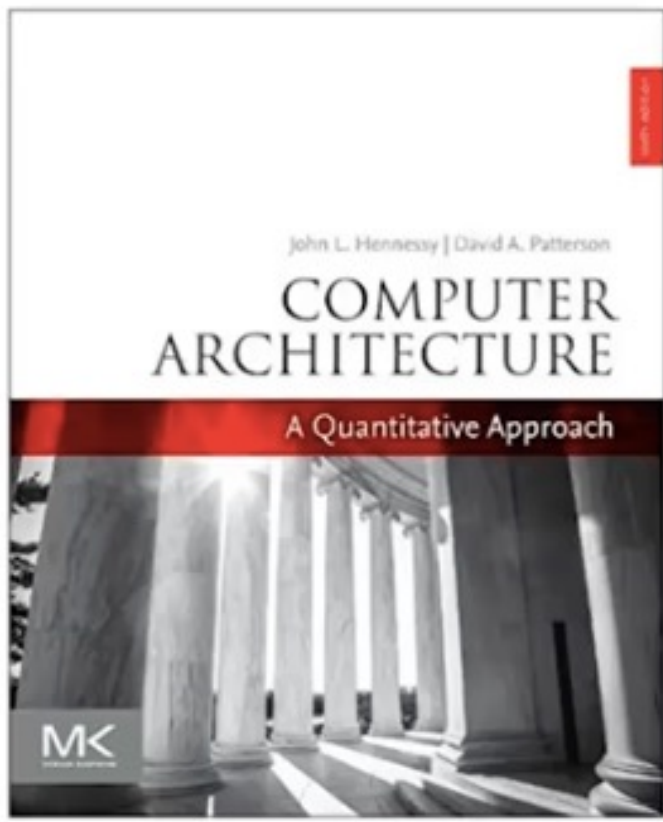
# RISC-V History Timeline

## RISC-V Development Timeline



# Books (Patterson & Hennessy)

RISC-V in Education





# RV vs ARM

Quora

Home

Answer <sup>177</sup>

Spaces

Notifications <sup>209</sup>

Search

Currently, the microcontroller (or -more generally- low end processor) IP space is very fragmented: ARM M series, MIPS M series, ANDES, Cortus, ARC, Cudasip and many others.

High end is exclusively ARM.



Microcontroller

Most of those are adopting RISC-V in a "consolidate but compete" manner: so Cortus, Andes, Cudasip have all stopped their proprietary ISAs in favour of RISC-V compatibility.

And then there are new entrants: SiFive most obviously, but Cloudbear, Syntacore, ROA etc

Western Digital has announced it is switching from ARC to RISC-V (one billion cores a year), Nvidia has announced all their on chip microcontrollers and "minion" cores will be RISC-V and there are many others.

Now, ARM is an excellent company with very good products and outstanding support.

As Microsoft and IBM can show it is entirely possible to succeed competing with open source or a radical industry shift. But it is not easy.

And I absolutely recognise that an SoC or an IP core is a different dynamic to software: if you try Libre Office and don't like it then it is a matter of moments to go back to MS Office. That is (cough, cough...) slightly harder if you decide you don't like the CPU in the SoC you just taped out....

BUT....

RISC-V is a very good architecture

# RV vs ARM

Quora



Home



Answer

177



Spaces



Notifications

209



[Watch More](#)



**Ramdas Mozhikunnath**, Experienced Microprocessor Verification engineer and Intel Alumni



Answered November 25, 2018

It may not "displace" the ARM architecture completely, but it is going to find some significant market share in embedded space.

ARM architecture is licensed by ARM (company) and involves a cost for any one who is making systems around same. (Either as an architecture license cost or as an IP cost).

On the other hand RISC V is an open source architecture - and there is no licence cost involved. This allows several of industry experts and academia to collaborate and innovate on this architecture and develop an entire ecosystem around same. (unlike limited to a single company like ARM)

Several companies have already made significant progress - SiFive is one the first companies already licensing IP cores based on RISC V and providing customized solutions. There are several other startups which are also building their own solutions.

Another interesting company is Esperanto technologies (lead by computer architecture veteran Dave Ditzel) which is building energy efficient powerful RISC V cores for the HPC market.

# RISC-V



ISA

**The RISC-V Instruction Set Manual**  
**Volume I: Unprivileged ISA**  
Document Version 20190608-Base-Ratified

Editors: Andrew Waterman<sup>1</sup>, Krste Asanović<sup>1,2</sup>

<sup>1</sup>SiFive Inc.,

<sup>2</sup>CS Division, EECS Department, University of California, Berkeley

andrew@sifive.com, krste@berkeley.edu

June 8, 2019

## The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1



*Andrew Waterman  
Yunsup Lee  
David A. Patterson  
Krste Asanović*

Electrical Engineering and Computer Sciences  
University of California at Berkeley

Technical Report No. UCB/EECS-2016-118  
<http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html>

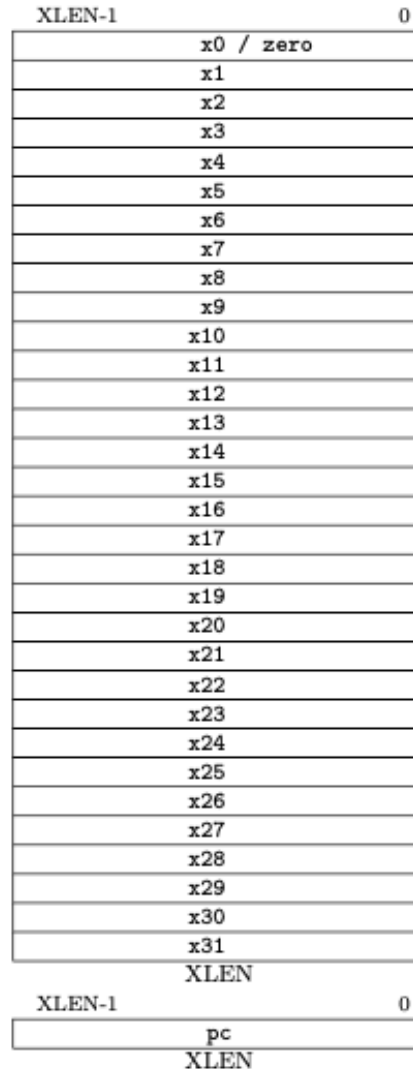
May 31, 2016

## Rationale

*We developed RISC-V to support our own needs in research and education, where our group is particularly interested in actual hardware implementations of research ideas (we have completed eleven different silicon fabrications of RISC-V since the first edition of this specification), and in providing real implementations for students to explore in classes (RISC-V processor RTL designs have been used in multiple undergraduate and graduate classes at Berkeley). In our current research, we are especially interested in the move towards specialized and heterogeneous accelerators, driven by the power constraints imposed by the end of conventional transistor scaling. We wanted a highly flexible and extensible base ISA around which to build our research effort.*

# RISC-V

Copyright © 2010–2016, The Regents of the University of California. All rights reserved.



XLEN = 32/64

Figure 2.1: RISC-V base unprivileged integer register state.

## 2.2 Base Instruction Formats

In the base RV32I ISA, there are four core instruction formats (R/I/S/U), as shown in Figure 2.2. All are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory. An instruction-address-misaligned exception is generated on a taken branch or unconditional jump if the target address is not four-byte aligned. This exception is reported on the branch or jump instruction, not on the target instruction. No instruction-address-misaligned exception is generated for a conditional branch that is not taken.

*The alignment constraint for base ISA instructions is relaxed to a two-byte boundary when instruction extensions with 16-bit lengths or other odd multiples of 16-bit lengths are added (i.e., IALIGN=16).*

*Instruction-address-misaligned exceptions are reported on the branch or jump that would cause instruction misalignment to help debugging, and to simplify hardware design for systems with IALIGN=32, where these are the only places where misalignment can occur.*

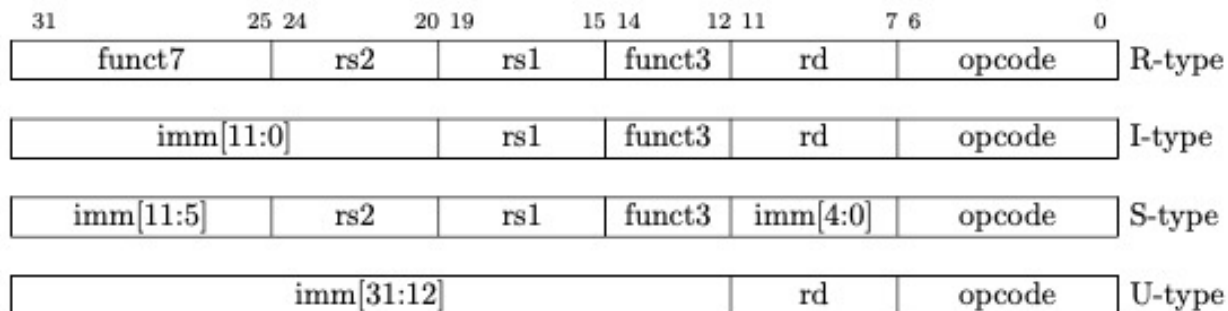


Figure 2.2: RISC-V base instruction formats. Each immediate subfield is labeled with the bit position (imm[x]) in the immediate value being produced, rather than the bit position within the instruction’s immediate field as is usually done.



# RISC-V

Copyright © 2010–2016, The Regents of the University of California. All rights reserved.

## Variable Length Instructions

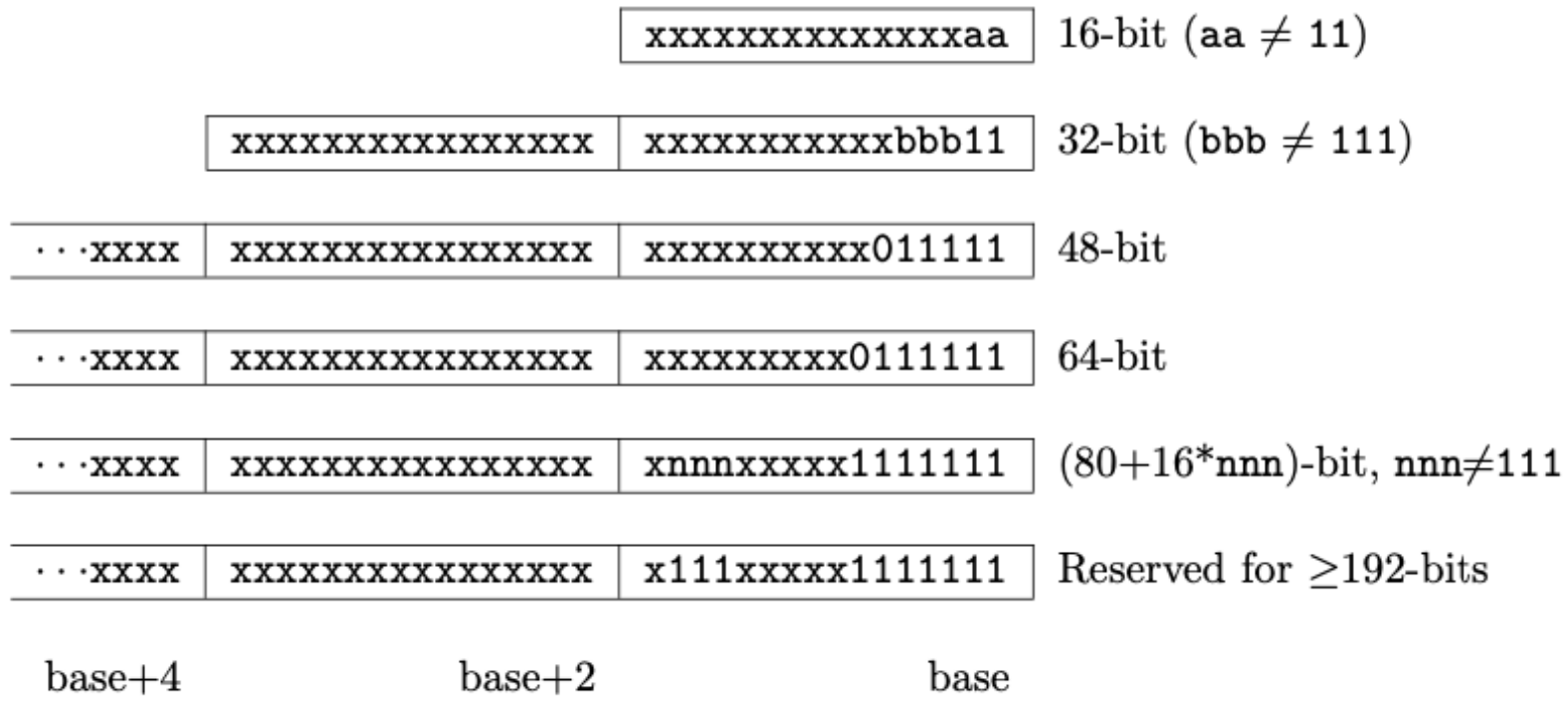
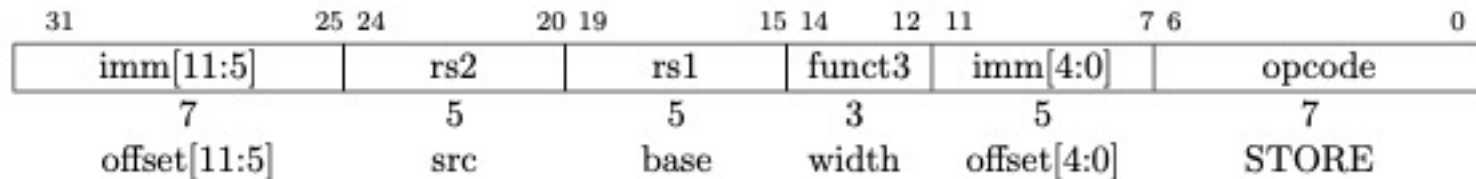
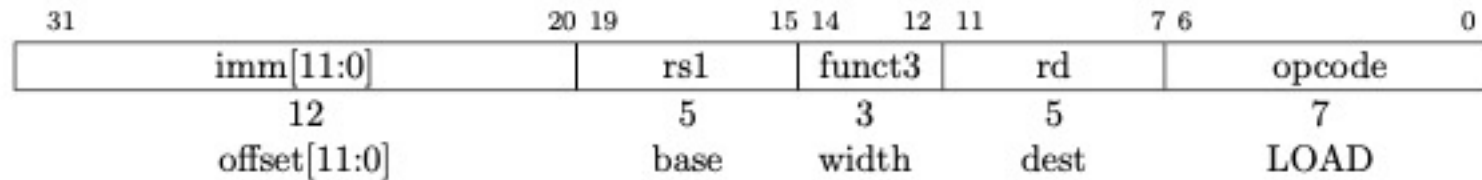


Figure 1.1: RISC-V instruction length encoding.

## 2.6 Load and Store Instructions

RV32I is a load-store architecture, where only load and store instructions access memory and arithmetic instructions only operate on CPU registers. RV32I provides a 32-bit address space that is byte-addressed and little-endian. The EEI will define what portions of the address space are legal to access with which instructions (e.g., some addresses might be read only, or support word access only). Loads with a destination of `x0` must still raise any exceptions and cause any other side effects even though the load value is discarded.



Load and store instructions transfer a value between the registers and memory. Loads are encoded in the I-type format and stores are S-type. The effective byte address is obtained by adding register

## Integer Register-Register Operations

### ALU

RV32I defines several arithmetic R-type operations. All operations read the *rs1* and *rs2* registers as source operands and write the result into register *rd*. The *funct7* and *funct3* fields select the type of operation.

31	25 24	20 19	15 14	12 11	7 6	0
funct7	rs2	rs1	funct3	rd	opcode	
7	5	5	3	5	7	
0000000	src2	src1	ADD/SLT/SLTU	dest	OP	
0000000	src2	src1	AND/OR/XOR	dest	OP	
0000000	src2	src1	SLL/SRL	dest	OP	
0100000	src2	src1	SUB/SRA	dest	OP	

ADD performs the addition of *rs1* and *rs2*. SUB performs the subtraction of *rs2* from *rs1*. Overflows are ignored and the low XLEN bits of results are written to the destination *rd*. SLT and SLTU perform signed and unsigned compares respectively, writing 1 to *rd* if  $rs1 < rs2$ , 0 otherwise. Note, SLTU *rd, x0, rs2* sets *rd* to 1 if *rs2* is not equal to zero, otherwise sets *rd* to zero (assembler pseudoinstruction SNEZ *rd, rs*). AND, OR, and XOR perform bitwise logical operations.

SLL, SRL, and SRA perform logical left, logical right, and arithmetic right shifts on the value in register *rs1* by the shift amount held in the lower 5 bits of register *rs2*.

## 2.3 Immediate Encoding Variants

### IMM

There are a further two variants of the instruction formats (B/J) based on the handling of immediates, as shown in Figure 2.3.

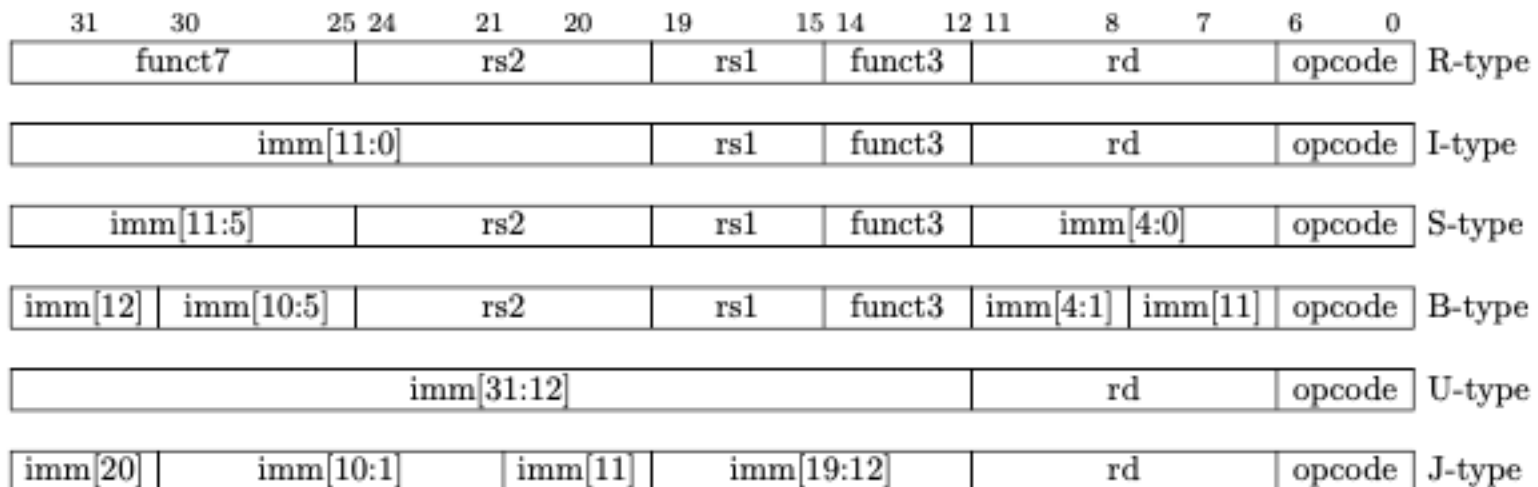


Figure 2.3: RISC-V base instruction formats showing immediate variants.

## 2.5 Control Transfer Instructions

BR

RV32I provides two types of control transfer instructions: unconditional jumps and conditional branches. Control transfer instructions in RV32I do *not* have architecturally visible delay slots.

Volume I: RISC-V Unprivileged ISA V20190608-Base-Ratified

23

31	30	25 24	20 19	15 14	12 11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode		
1	6	5	5	3	4	1	7		
offset[12 10:5]		src2	src1	BEQ/BNE	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BLT[U]	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BGE[U]	offset[11 4:1]		BRANCH		

Branch instructions compare two registers. BEQ and BNE take the branch if registers *rs1* and *rs2* are equal or unequal respectively. BLT and BLTU take the branch if *rs1* is less than *rs2*, using signed and unsigned comparison respectively. BGE and BGEU take the branch if *rs1* is greater than or equal to *rs2*, using signed and unsigned comparison respectively. Note, BGT, BGTU, BLE, and BLEU can be synthesized by reversing the operands to BLT, BLTU, BGE, and BGEU, respectively.

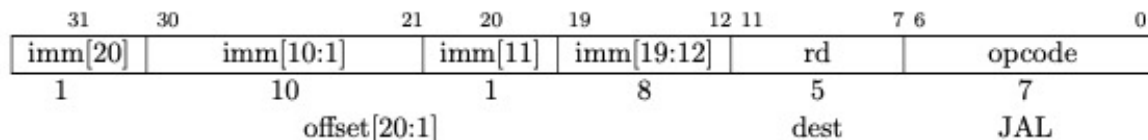
JMP

## Unconditional Jumps

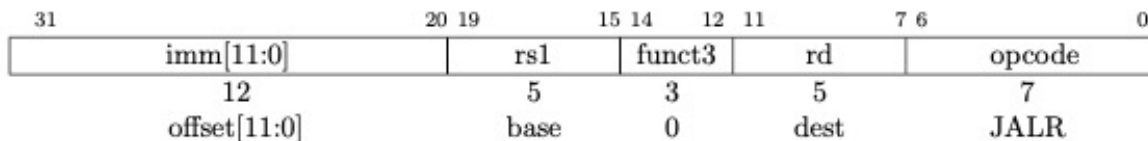
The jump and link (JAL) instruction uses the J-type format, where the J-immediate encodes a signed offset in multiples of 2 bytes. The offset is sign-extended and added to the address of the jump instruction to form the jump target address. Jumps can therefore target a  $\pm 1$  MiB range. JAL stores the address of the instruction following the jump ( $pc+4$ ) into register *rd*. The standard software calling convention uses *x1* as the return address register and *x5* as an alternate link register.

*The alternate link register supports calling millicode routines (e.g., those to save and restore registers in compressed code) while preserving the regular return address register. The register *x5* was chosen as the alternate link register as it maps to a temporary in the standard calling convention, and has an encoding that is only one bit different than the regular link register.*

Plain unconditional jumps (assembler pseudoinstruction J) are encoded as a JAL with *rd*=*x0*.



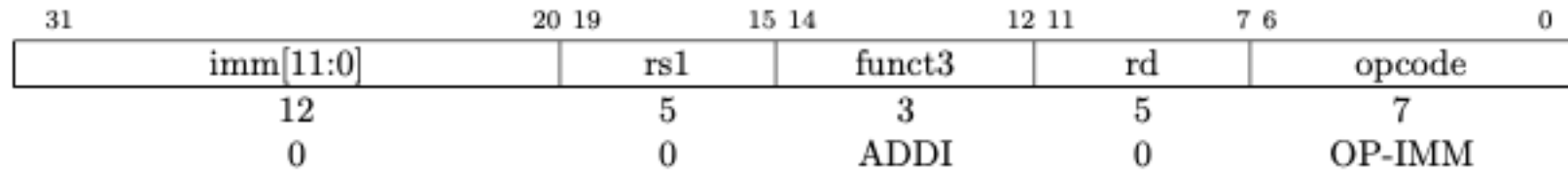
The indirect jump instruction JALR (jump and link register) uses the I-type encoding. The target address is obtained by adding the sign-extended 12-bit I-immediate to the register *rs1*, then setting the least-significant bit of the result to zero. The address of the instruction following the jump ( $pc+4$ ) is written to register *rd*. Register *x0* can be used as the destination if the result is not required.



*The unconditional jump instructions all use PC-relative addressing to help support position-independent code. The JALR instruction was defined to enable a two-instruction sequence to jump anywhere in a 32-bit absolute address range. A LUI instruction can first load *rs1* with the*

NOP

## NOP Instruction



The NOP instruction does not change any architecturally visible state, except for advancing the pc and incrementing any applicable performance counters. NOP is encoded as ADDI  $x0, x0, 0$ .

## 1.6 Exceptions, Traps, and Interrupts

We use the term *exception* to refer to an unusual condition occurring at run time associated with an instruction in the current RISC-V hart. We use the term *interrupt* to refer to an external asynchronous event that may cause a RISC-V hart to experience an unexpected transfer of control. We use the term *trap* to refer to the transfer of control to a trap handler caused by either an exception or an interrupt.

The instruction descriptions in following chapters describe conditions that can raise an exception during execution. The general behavior of most RISC-V EEIs is that a trap to some handler occurs when an exception is signaled on an instruction (except for floating-point exceptions, which, in the standard floating-point extensions, do not cause traps). The manner in which interrupts are generated, routed to, and enabled by a hart depends on the EEI.

---

*Our use of “exception” and “trap” is compatible with that in the IEEE-754 floating-point standard.*



## 2.9 Environment Call and Breakpoints

31		20 19		15 14	12 11		7 6	0
	funct12		rs1		funct3		rd	opcode
	12		5		3		5	7
	ECALL		0		PRIV		0	SYSTEM
	EBREAK		0		PRIV		0	SYSTEM

The ECALL instruction is used to make a request to the supporting execution environment, which is usually an operating system. The ABI for the system will define how parameters for the environment request are passed, but usually these will be in defined locations in the integer register file.

The EBREAK instruction is used by debuggers to cause control to be transferred back to a debugging environment.

---

*ECALL and EBREAK were previously named SCALL and SBREAK. The instructions have the same functionality and encoding, but were renamed to reflect that they can be used more generally than to call a supervisor-level operating system or debugger.*

## 2.7 Memory Model

The base RISC-V ISA supports multiple concurrent threads of execution within a single user address space. Each RISC-V thread has its own user register state and program counter, and executes an independent sequential instruction stream. The execution environment will define how RISC-V threads are created and managed. RISC-V threads can communicate and synchronize with other threads either via calls to the execution environment, which are documented separately in the specification for each execution environment, or directly via the shared memory system. RISC-V threads can also interact with I/O devices, and indirectly with each other, via loads and stores to portions of the address space assigned to I/O.

In the base RISC-V ISA, each RISC-V thread observes its own memory operations as if they executed sequentially in program order. RISC-V has a relaxed memory model between threads, requiring an explicit FENCE instruction to guarantee any specific ordering between memory operations from different RISC-V threads. Chapter [6](#) describes the optional atomic memory instruction extensions “A”, which provide additional synchronization operations.

31	28	27	26	25	24	23	22	21	20	19	15	14	12	11	7	6	0
0	PI	PO	PR	PW	SI	SO	SR	SW	rs1	funct3	rd	opcode					
4	1	1	1	1	1	1	1	1	5	3	5	7					
0	predecessor				successor				0	FENCE	0	MISC-MEM					

The FENCE instruction is used to order device I/O and memory accesses as viewed by other RISC-V threads and external devices or coprocessors. Any combination of device input (I), device output

## Conditional Branches

All branch instructions use the SB-type instruction format. The 12-bit B-immediate encodes signed offsets in multiples of 2, and is added to the current pc to give the target address. The conditional branch range is  $\pm 4$  KiB.

31	30	25 24	20 19	15 14	12 11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode		
1	6	5	5	3	4	1	7		
offset[12,10:5]		src2	src1	BEQ/BNE	offset[11,4:1]		BRANCH		
offset[12,10:5]		src2	src1	BLT[U]	offset[11,4:1]		BRANCH		
offset[12,10:5]		src2	src1	BGE[U]	offset[11,4:1]		BRANCH		

Branch instructions compare two registers. BEQ and BNE take the branch if registers *rs1* and *rs2* are equal or unequal respectively. BLT and BLTU take the branch if *rs1* is less than *rs2*, using

*The conditional branches were designed to include arithmetic comparison operations between two registers (as also done in PA-RISC and Xtensa ISA), rather than use condition codes (x86, ARM, SPARC, PowerPC), or to only compare one register against zero (Alpha, MIPS), or two registers only for equality (MIPS). This design was motivated by the observation that a combined compare-and-branch instruction fits into a regular pipeline, avoids additional condition code state or use of a temporary register, and reduces static code size and dynamic instruction fetch traffic. Another point is that comparisons against zero require non-trivial circuit delay (especially after the move to static logic in advanced processes) and so are almost as expensive as arithmetic magnitude compares. Another advantage of a fused compare-and-branch instruction is that branches are observed earlier in the front-end instruction stream, and so can be predicted earlier. There is perhaps an advantage to a design with condition codes in the case where multiple branches can be taken based on the same condition codes, but we believe this case to be relatively rare.*

*We considered but did not include static branch hints in the instruction encoding. These can reduce the pressure on dynamic predictors, but require more instruction encoding space and software profiling for best results, and can result in poor performance if production runs do not match profiling runs.*

*We considered but did not include conditional moves or predicated instructions, which can effectively replace unpredictable short forward branches. Conditional moves are the simpler of the two, but are difficult to use with conditional code that might cause exceptions (memory accesses and floating-point operations). Predication adds additional flag state to a system, additional instructions to set and clear flags, and additional encoding overhead on every instruction.*

# RISC-V



**RISC-V**  
**Foundation**

# Consortium



**RISC-V** Foundation: 100+ Members



# Foundation Officers

March 2021

## RISC-V Board of Directors: Officers



**Krste Asanović**  
Chairman of the Board

Professor, EECS Department, UC Berkeley

[Read More](#)  
in



**David Patterson**  
Vice Chair

Distinguished Engineer, Google

[Read More](#)  
in



**Zvonimir Bandić**  
Board Treasurer

Senior Director of Next Generation Platform Technologies,  
Western Digital Corporation

[Read More](#)  
in



[About RISC-V](#)

- Membership
- RISC-V Exchange
- Technical
- News & Events
- Community



**Calista Redmond**  
CEO

RISC-V International

[Read More](#)



**Jeffrey Osier-Mixon**  
Program Manager

RISC-V International

[Read More](#)



**Jenni McGinnis**  
Program Manager

RISC-V International

[Read More](#)



**Kim McMahon**  
Director of Marketing

RISC-V International

[Read More](#)



**Mark Himmelstein**  
CTO



**Megan Lehn**  
Program Manager



**Stephano Cetola**  
Technical Program Manager

# RISC-V

---



## Benchmarks

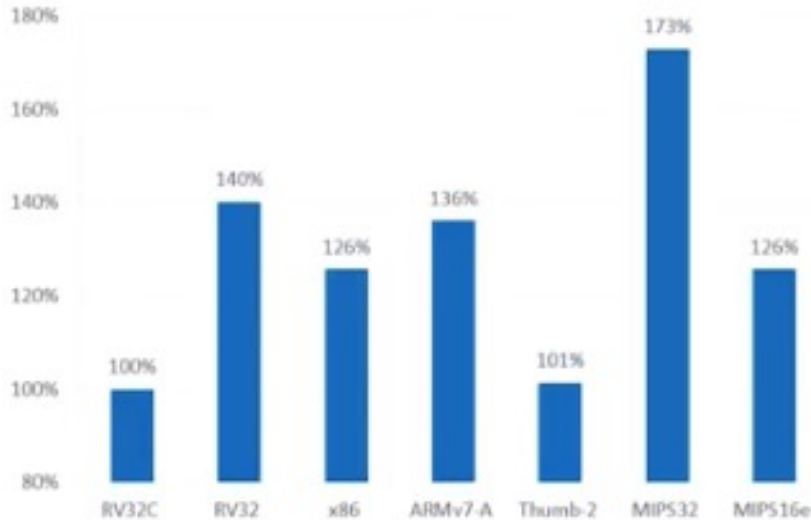


# Static Code Benchmarks

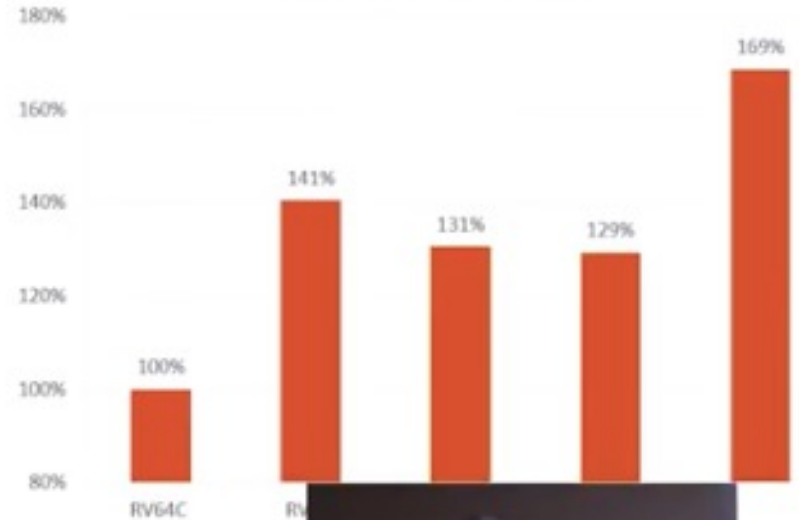
RISC-V: SPECint206 compressed code size (relative to "standard" RVC)



32-bit Architectures



64-bit Architectures

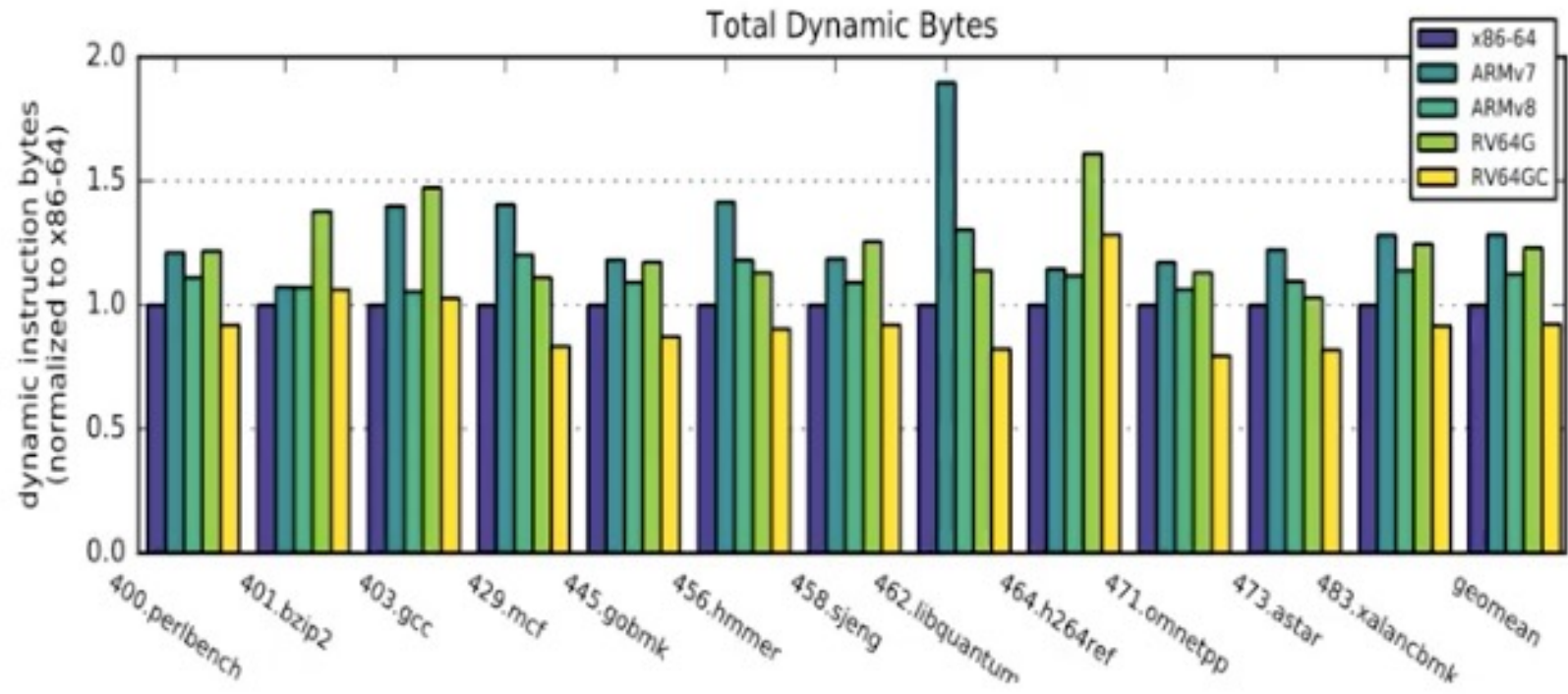


- RISC-V now smallest ISA for 32- and 64-bit addresses
- All results with same GCC compiler and options



# Dynamic Code Benchmarks

## RISC-V: Dynamic Bytes Fetched



- RV64GC is lowest overall in dynamic bytes fetched

# RISC-V



**Cores**

# NVIDIA (NVDLA)

## NVIDIA Open Sources Deep Learning Accelerator (NVDLA)

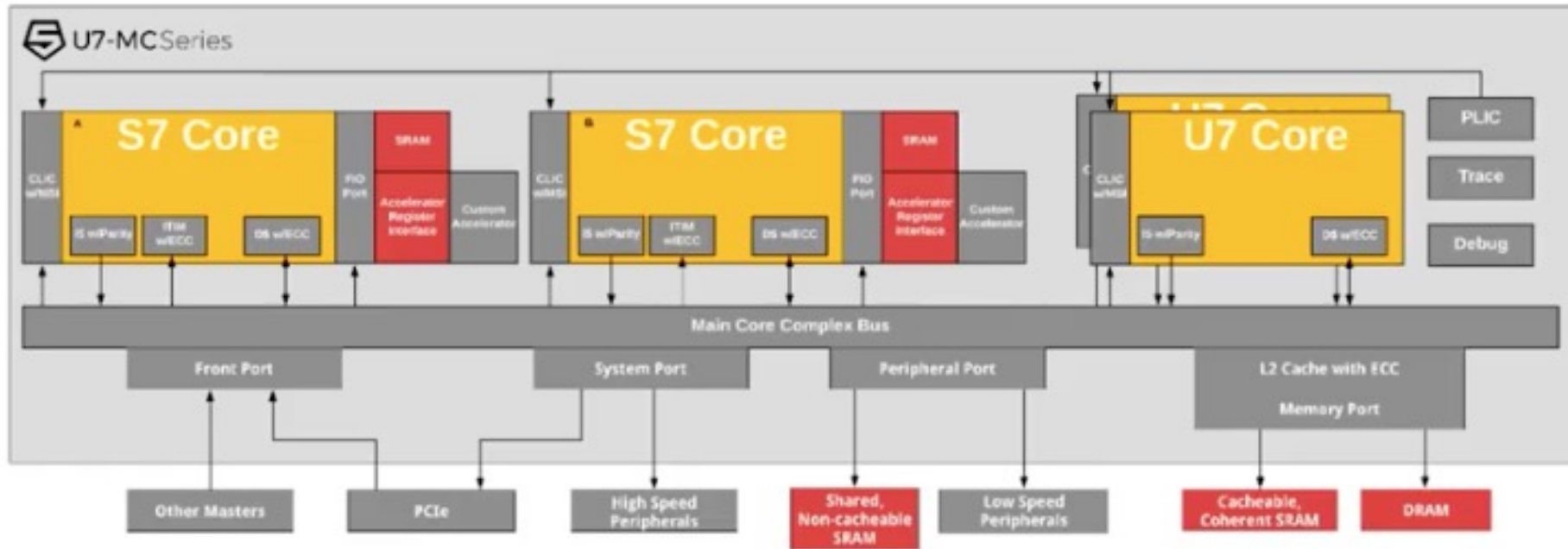
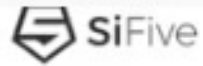


- Visit [nvdla.org](http://nvdla.org)
- The NVIDIA Deep Learning Accelerator (NVDLA) is a free and open architecture that promotes a standard way to design deep learning inference accelerators
- With its modular architecture, NVDLA is scalable, highly configurable, and designed to simplify integration and portability
- The hardware supports a wide range of IoT devices

<https://www.anandtech.com/show/11360/the-nvidia-gpu-tech-conference-2017-keynote-live-blog>

# Custom Multi-Core

## Custom RISC-V Core Complex

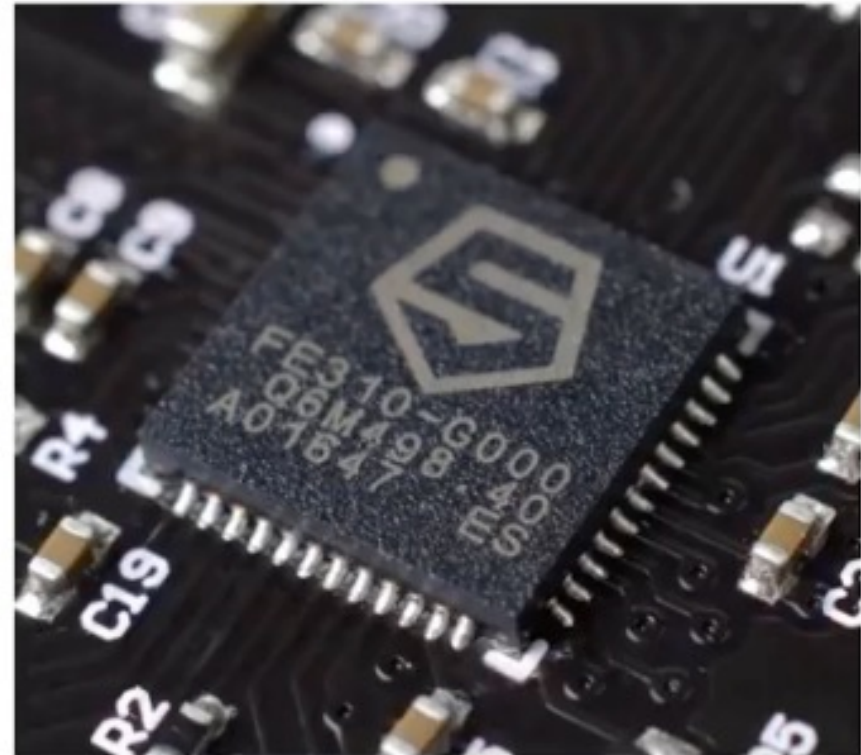
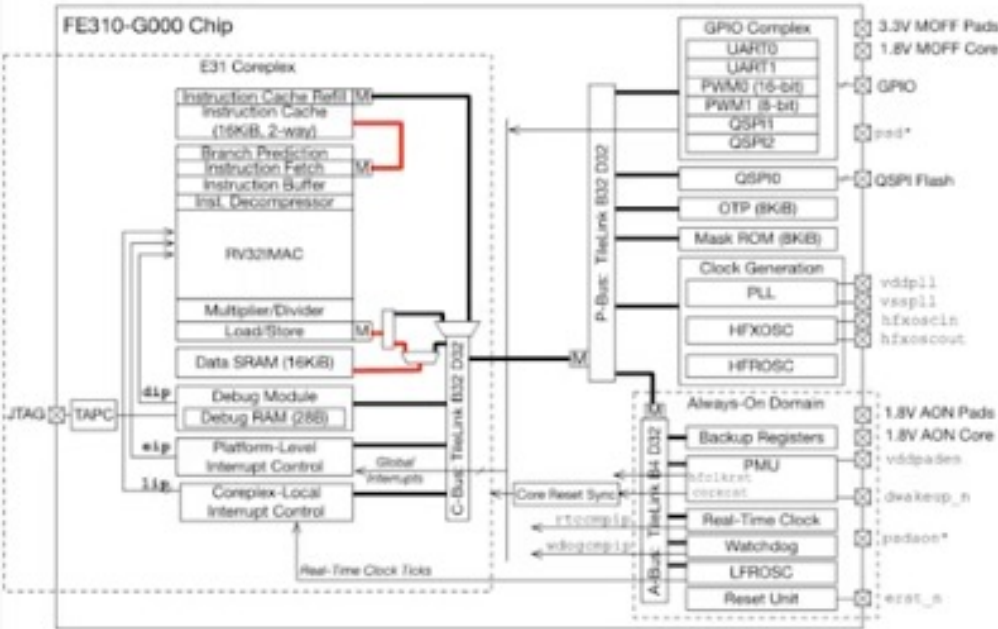


- ITIM and FIO Ports provide fast local SRAM and Accelerators
- Coherent System with a shared Level 2 Cache Controller
- Front Port allows other masters access to U7-MC Core Complex Memories

# MCU (Low Power)

 Freedom Everywhere **32/64-bit Low-power Microcontroller Platform**

 SiFive



Freedom E310, QFN48, manufactured in TSMC 180nm

- 320+ MHz SiFive E31 CPU
  - 16KB L1I\$, 16KB Data Scratchpad
  - Hardware Multiply/Divide, Debug Module
- Multiple Power Domains
- Low-Power Standby
- Wide Range of Clock Inputs

# Cheap (\$59) Dev Board

## HiFive1: Arduino-Compatible RISC-V Dev Board



- SiFive FE310-G000 (built in 180nm)
- Operating Voltage: 3.3 V and 1.8 V
- Input Voltage: 5 V USB or 7-12 VDC Jack
- IO Voltages: Both 3.3 V or 5 V supported
- Digital I/O Pins: 19
- PWM Pins: 9
- SPI Controllers/HW CS Pins: 1/3
- External Interrupt Pins: 19
- External Wakeup Pins: 1
- Flash Memory: 16 MB Quad SPI
- Host Interface (mi and Serial Comm

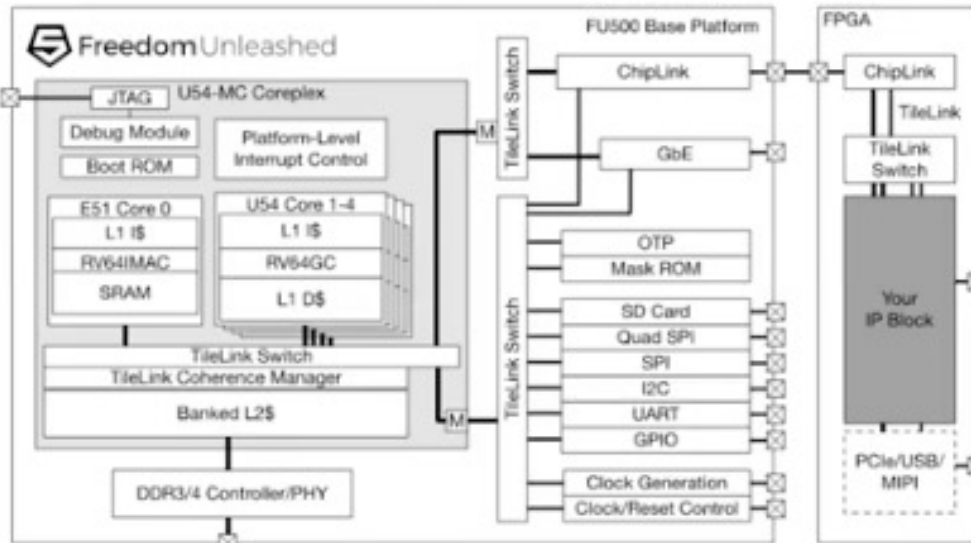
Order now at [crowdsupply.com](https://crowdsupply.com) for \$59



\$59

# 64-Bit Multi-Core Linux

FreedomUnleashed **64-bit Multi-Core RISC-V Linux Platform**



- 1.5+ GHz U54-MC SiFive CPU
  - 1x E51: 16KB L1I\$, 8KB DTIM with ECC support
  - 4x U54: 32KB L1I\$, 32KB L1D\$ with ECC support
  - Single- and Double-precision floating-point support
  - 2MB Banked L2\$ with directory-based cache-coherence & ECC support
- ChipLink
  - Serialized Chip-to-Chip Coherent TileLink Interconnect
- DDR3/4, GbE, Peripherals



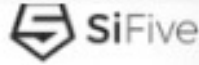
Freedom U540,





# Multi-Core Dev Board

HiFive Unleashed: World's First Multi-Core RISC-V Linux Dev Board



- SiFive FU540-C000 (built in 28nm)
- 8 GB 64-bit DDR4 with ECC
- Gigabit Ethernet Port
- 32 MB Quad SPI Flash
- MicroSD card for removable storage
- MicroUSB for debug and serial communication
- Digital GPIO pins
- FMC connector for future expansion with add-i

Order now at [crowdsupply.com](https://crowdsupply.com) for \$999



\$999

# FPGA Expansion Board

Develop Custom Accelerators with HiFive Unleashed Expansion Board



Brought to you by SiFive and Microsemi



# RISC-V



## Design Tools

# RISC-V Design



---

## Maven Silicon

---

### Maven Silicon's RISC-V Processor IP Verification Flow

by Sivakumar PR on 02-24-2023 at 6:00 am

Categories: Maven Silicon, RISC-V, Semiconductor Services

**RISC-V** is a general-purpose **license-free** open Instruction Set Architecture [ISA] with **multiple extensions**. It is an ISA separated into a **small base integer ISA**, usable as a base for customized accelerators and optional standard extensions to support general-purpose software development. RISC-V supports **both 32-bit and 64-bit** address space variants for applications, operating system kernels, and hardware implementations. So, it is suitable for all computing systems, from embedded microcontrollers to cloud servers.

# RISC-V Design

I have defined Maven Silicon's **RISC-V** verification flow using the correct by-construction approach. The approach is to build a pre-verified synthesizable RISC-V IP fundamental **IP building blocks library** and create any kind of **multi-stage pipeline** RISC-V processor using this library. Finally, the multi-stage pipeline RISC-V processor IP can be verified using Constrained Random Coverage Driven Verification [CRCDV] in **Universal Verification Methodology** [UVM] and **FPGA** prototyping.

# RISC-V Design

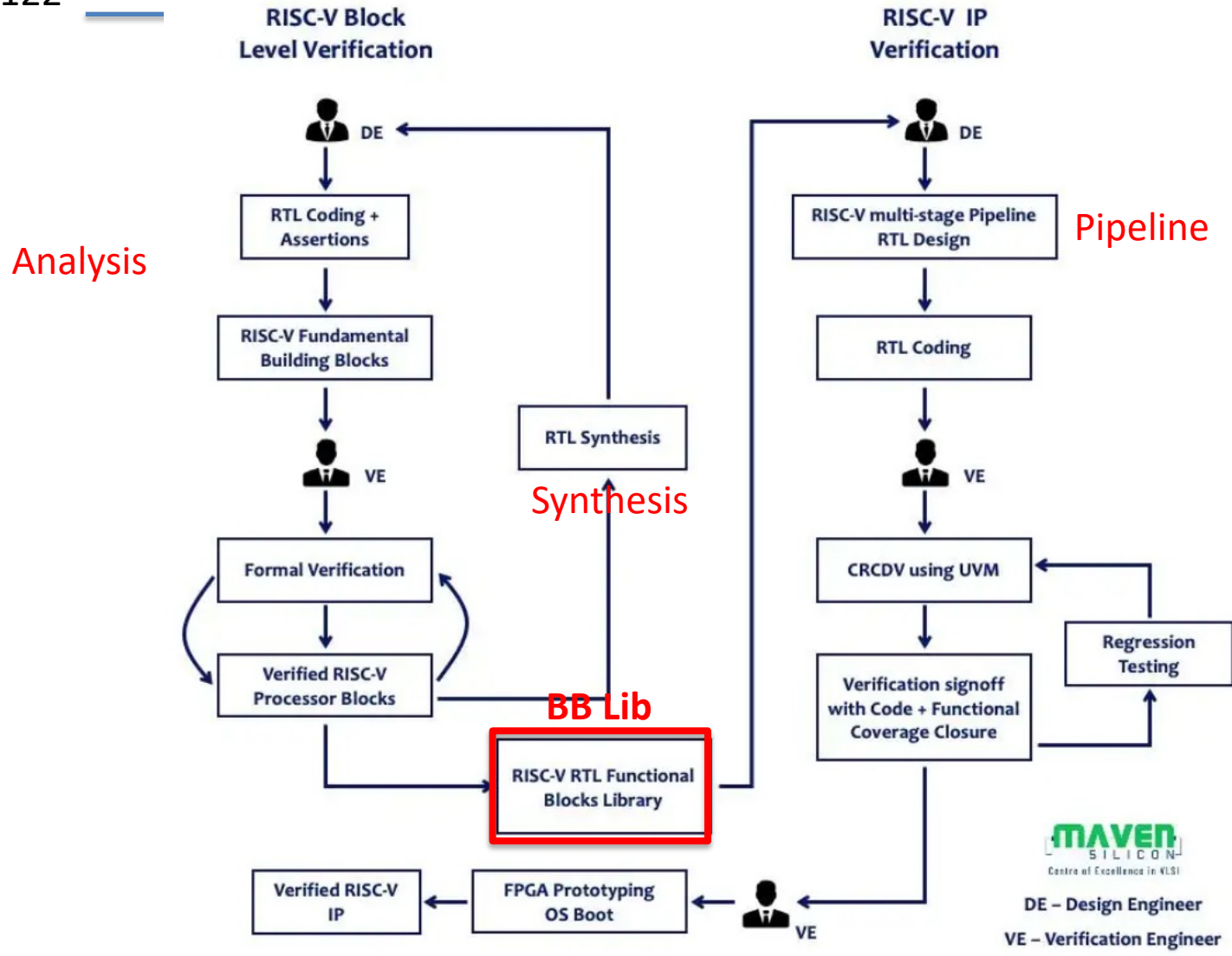


Figure1: Maven Silicon's RISC-V IP Verification Flow

# RISC-V Design

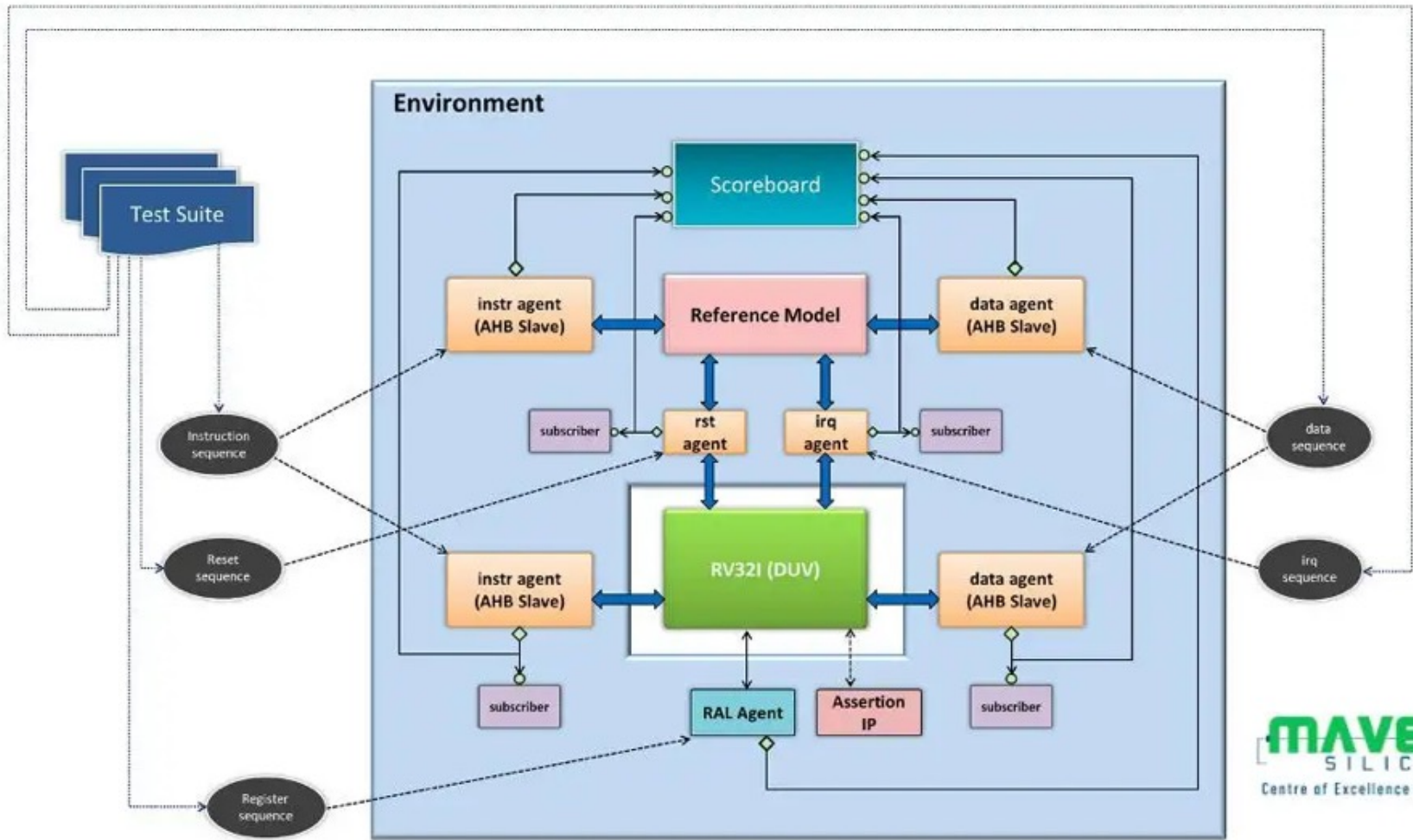


Figure 2: Maven Silicon's RISC-V IP UVM Verification Environment

# RISC-V



**Software**



# RISC-V Software

## RISC-V Software

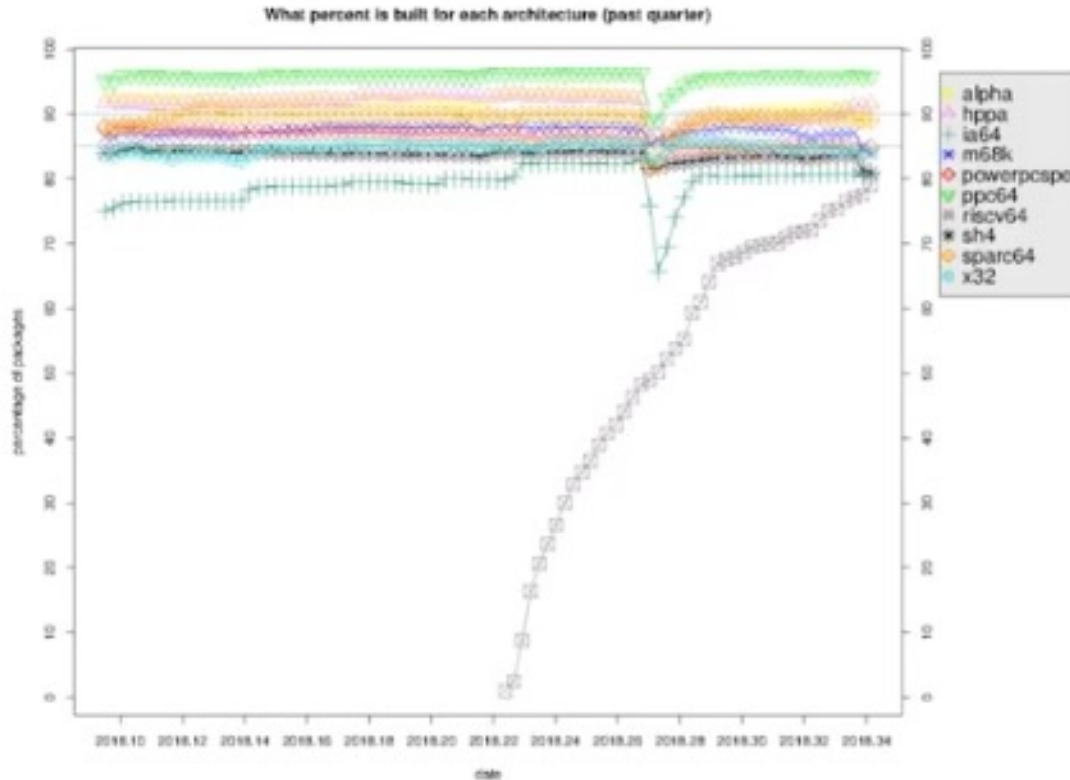


- RISC-V assembler, compiler, C library ports are upstreamed
  - binutils, GCC, newlib, glibc
- RISC-V OS ports are upstreamed
  - Zephyr, Linux, FreeBSD
- RISC-V emulator ports are upstreamed
  - QEMU
- RISC-V distribution ports are in progress
  - Debian, Fedora, OpenEmbedded, OpenWRT, Gentoo



# Unix Kernel (Debian)

## Debian RISC-V Distro



- RISC-V Debian Port Goals
  - Software-wise, this port will target the Linux kernel
  - Hardware-wise, the port will target the 64-bit variant, little-endian
- More packages than ia64
- Debian Distro runs on the HiFive Unleashed development board with the SiFive Freedom U540 SoC

<https://wiki.debian.org/RISC-V>



# ISA


---



**RISC-V**  
**BSD**

<https://www.youtube.com/watch?v=AOC7KmHvx9w>

# RV Origin




## RISC-V Origin Story

- x86 impossible –IP issues, too complex
- ARM mostly impossible – no 64-bit, IP issues, complex
- So we started “3-month project” in summer 2010 to develop our own clean-slate ISA
  - Andrew Waterman, Yunsup Lee, Dave Patterson, Krste Asanovic principal designers
- Four years later, we released frozen base user spec
  - First public specification released in May 2011
  - Many tapeouts and several publications along the way

*Why are outsiders complaining about changes to RISC-V in Berkeley classes?*

# ISA's




## Why Instruction Set Architecture matters

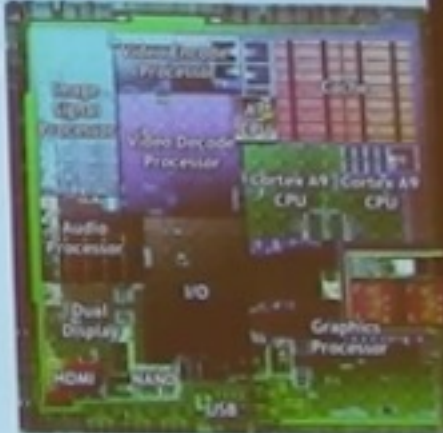
- **Why can't Intel sell mobile chips?**
  - 99%+ of mobile phones/tablets based on ARM v7/v8 ISA
- **Why can't ARM partners sell servers?**
  - 99%+ of laptops/desktops/servers based on AMD64 ISA (over 95%+ built by Intel)
- **How can IBM still sell mainframes?**
  - IBM 360, oldest surviving ISA (50+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

*ISA is most important interface in computer system  
where software meets hardware*

# Multi-ISA


 **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 too large for base accelerator ISA
- 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*

# Open Standards

 **Open Software/Standards Work!**

<i>Field</i>	<i>Standard</i>	<i>Free, Open Impl.</i>	<i>Proprietary Impl.</i>
Networking	Ethernet, TCP/IP	Many	Many
OS	Posix	Linux, FreeBSD	M/S Windows
Compilers	C	gcc, LLVM	Intel icc, ARMcc
Databases	SQL	MySQL, PostgreSQL	Oracle 12C, M/S DB2
Graphics	OpenGL	Mesa3D	M/S DirectX
ISA	??????	-----	x86, ARM, IBM360

- Why not successful free & open standards and free & open implementations, like other fields
- Dominant proprietary ISAs are *dismal* designs

# ISA

---



**RISC-V**  
**Cores**



# Semidynamic<sup>s</sup> RISC-V Cores

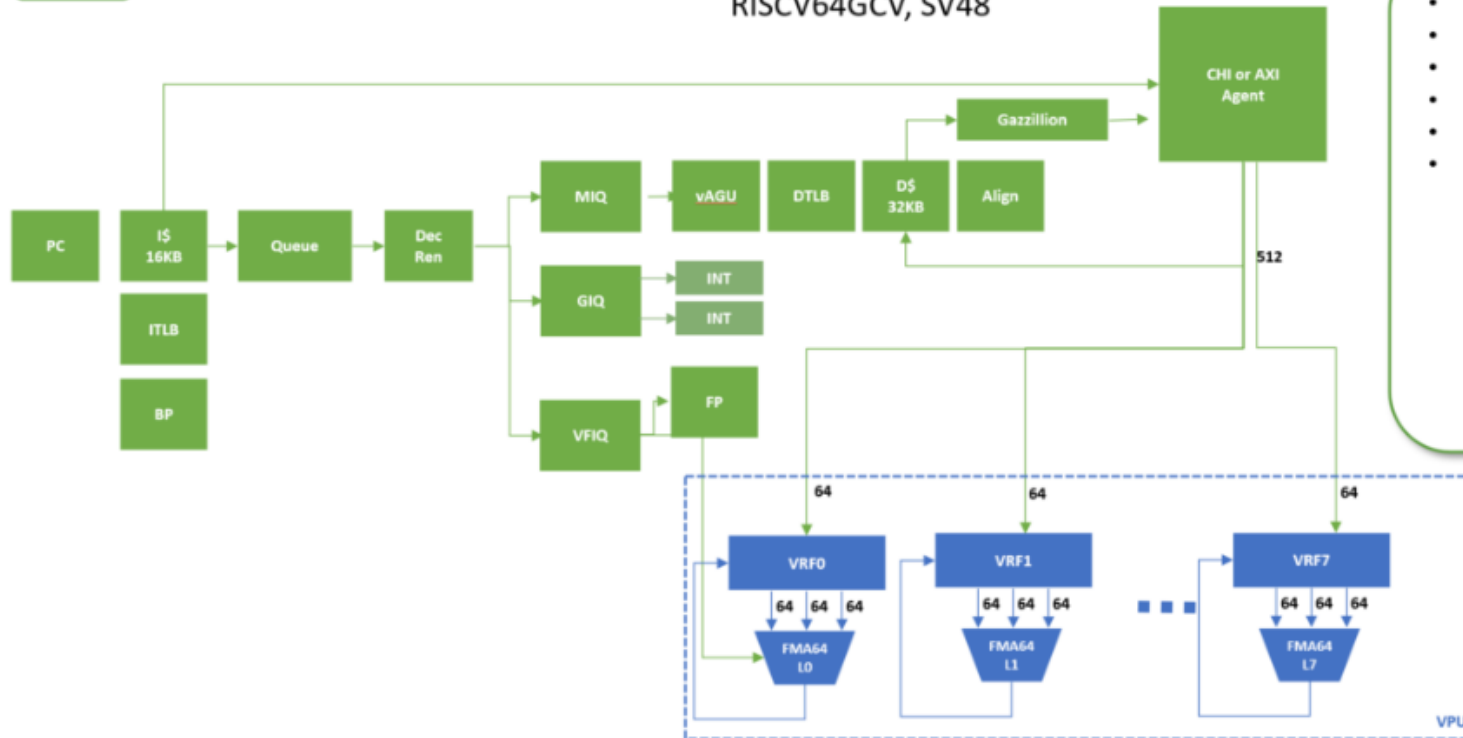
*Way beyond tailorable, fully configurable RISC-V cores match the requirements*

Most [RISC-V](#) vendors offer a good list of tailorable features for their cores. Atrevido has been envisioned from the ground up as a **fully customizable core** where everything is on the table. A customer interview phase determines the application's needs, and then the core is optimized for **power-performance-area** (PPA). Don't need a vector unit? No problem. Change the address space? Sure. Need custom instructions? Straightforward. Coherency, scheduling, or tougher needs? Semidynamics has carved out a unique space apart from the competition, providing customers with better differentiation as they can open up the core for changes – **Open Core Surgery**, as Espasa enthusiastically terms it. “We can include unique features in a few weeks, and have a customized core validated in a few months,” says Espasa.



## Atrevido 423 with VPU

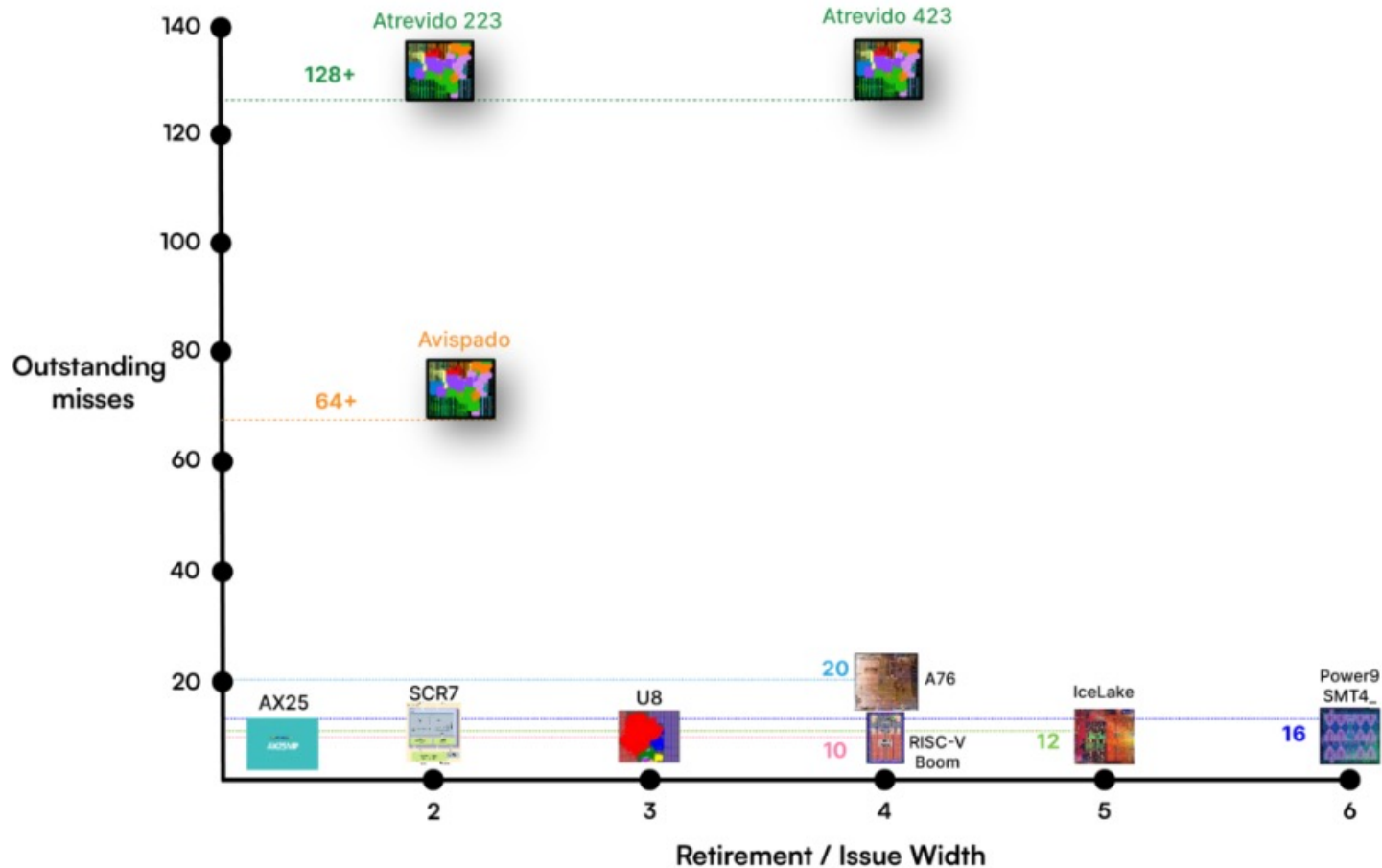
RISCV64GCV, SV48



- 64-bit Core
- Fast unaligned
- AXI/CHI
- SV48
- Linux Ready
- Available extensions:
  - Bit Manipulation
  - Single/Double
  - Crypto
  - CMO's
  - Zifencei

# semidynamic<sup>s</sup> RISC-V Cores

Comparison to other cores:



9-18-23

## SEMIDYNAMICS

### Deeper RISC-V pipeline plows through vector-scalar loops

by Don Dingee on 09-14-2023 at 10:00 am

Categories: IP, RISC-V, Semidynamics

Many modern processor performance benchmarks rely on as many as three levels of cache staying continuously fed. Yet, new data-intensive applications like multithreaded generative AI and 4K image processing often break conventional caching, leaving the expensive execution units behind them stalled. A while back, Semidynamics introduced us to their new highly customizable RISC-V core, Atrevido, with its Gazillion memory retrieval technology designed to solve more

Vector-Scalar

$$Z = \mathbf{a}X + \mathbf{b}$$

$$Z = \mathbf{a}X + Y$$

9-18-23

## *Some emerging use cases for a deeper RISC-V pipeline*

“We continuously get requests for new data types, and our answer is always yes, we can add that with some engineering time,” Espasa points out. int4 and fp8 additions say a lot about the type of application they are seeing: simpler, less training-intensive AI inference models, but hundreds or thousands of concurrent threads. Consider something like a generative AI query server where users hit it asynchronously with requests. One stream is no big deal, but 100 can overwhelm a conventional caching scheme. Gazillion fetches help achieve a deeper RISC-V pipeline scale not seen in other architectures.

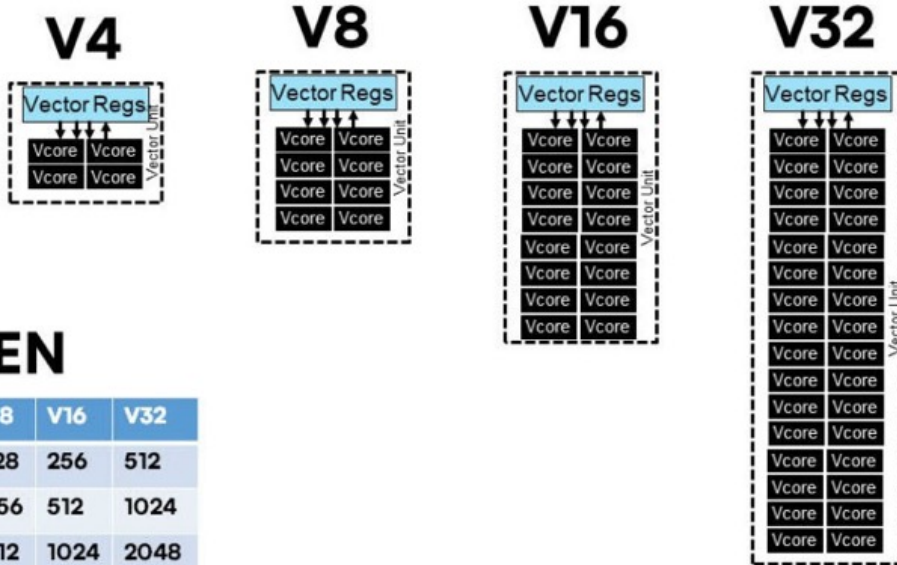
There’s also the near-far imaging problem – having to blast through high frame rates of 4K images looking for small-pixel fluctuations that may turn into targets of interest. Most AI inference engines are good once regions of interest take shape, but having to process the entire field of the image slows things down. When we mentioned one of the popular AI inference IP providers and their 24-core engine, Espasa blushed a bit. “Let’s just say we work with customers to adapt Atrevido to what they need rather than telling them what it has to look like.”

9-18-23

## Number of Vector Cores

Vector-Scalar

$$Z = aX + Y$$



### DLEN

VCore	V4	V8	V16	V32
ELEN=16	64	128	256	512
ELEN=32	128	256	512	1024
ELEN=64	256	512	1024	2048

9-18-23

## Atrevido 423 + V16 Vector Unit

Vector-Scalar

$$Z = aX + Y$$

