Syntacore™
Custom cores and tools

C/C++ Compilers and Software Optimization for RISC-V

Sergey Yakushkin
June 26, 2022

info@syntacore.com

Outline

- RISC-V Intro
- Example CPU IP
- LLVM Compilers
- Optimization Examples

RISC-V International

RISC-V is free and open RISC ISA:

- modular, extensible, allows co-design
- supports wide range from deeply embedded to high performance
- driven through collaboration
- new emerging standard already, influence comparable to USB or Linux.

RISC-V Foundation/International

- established in 2015 by industry leaders and startups
- 2500+ members from 70+ countries
- ~100 universities and labs
- promotes research and innovation

members

Google FOUNDING
Syntacore Custom cores and tools FOUNDING
SiFive FOUNDING
IBM FOUNDING
intel
NVIDIA
NXP
HUAWEI
ANDES TECHNOLOGY FOUNDING
SAMSUNG
SEAGATE
MICROCHIP FOUNDING
Qualcomm FOUNDING
codasip FOUNDING
XILINX
life.augmented
SONY

RISC-V ISA

- Small base integer ISA RV32/RV64
- Extensions (IMAFD)

Avoids over-architecting for:

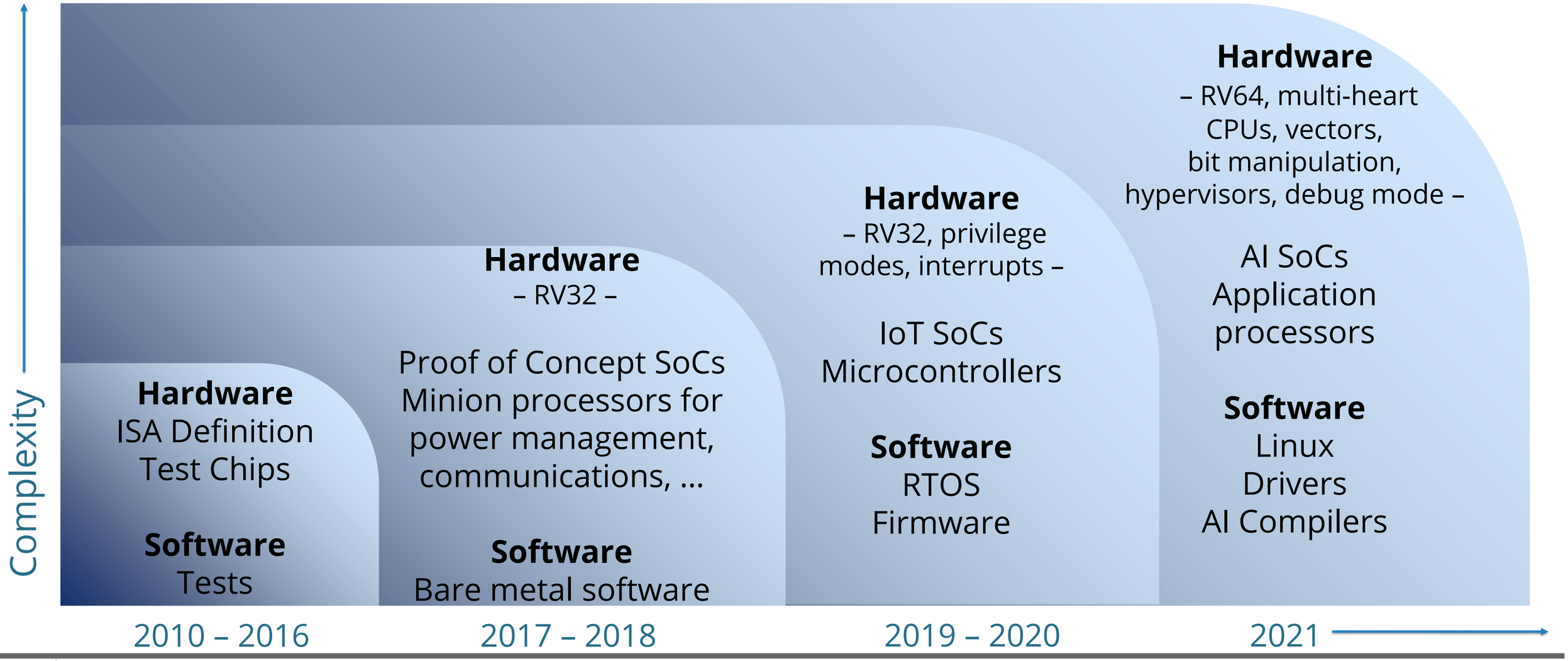
- specific usage scenario
- manufacturing technology
- microarchitecture

Design choices to omit:

- flags and conditional codes
- branch delay slots
- 8/16-bit arithmetic
- address mode write-back

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP; only memory address mode is 12-bit displacement+contents of a GPR*
`lb, lbu, sb`	Load byte, load byte unsigned, store byte (to/from integer registers)
`lh, lhu, sh`	Load half word, load half word unsigned, store half word (to/from integer registers)
`lw, lwu, sw`	Load word, store word (to/from integer registers)
`ld, sd`	Load doubleword, store doubleword
Arithmetic/logical	*Operations on data in GPRs. Word versions ignore upper 32 bits*
`add, addi, addw, addiw, sub, subi, subw, subiw`	Add and subtract, with both word and immediate versions
`slt, sltu, slti, sltiu`	set-less-than with signed and unsigned, and immediate
`and, or, xor, andi, ori, xori`	and, or, xor, both register-register and register-immediate
`lui`	Load upper immediate: loads bits 31..12 of a register with the immediate value. Upper 32 bits are set to 0
`auipc`	Sums an immediate and the upper 20-bits of the PC into a register; used for building a branch to any 32-bit address
`sll, srl, sra, slli, srli, srai, sllw,slliw, srli, srliw, srai, sraiw`	Shifts: logical shift left and right and arithmetic shift right, both immediate and word versions (word versions leave the upper 32 bit untouched)
`mul, mulw, mulh, mulhsu, mulhu, div,divw, divu, rem, remu, remw, remuw`	Integer multiply, divide, and remainder, signed and unsigned with support for 64-bit products in two instructions. Also word versions
Control	*Conditional branches and jumps; PC-relative or through register*
`beq, bne, blt, bge, bltu, bgeu`	Branch based on compare of two registers, equal, not equal, less than, greater or equal, signed and unsigned
`jal,jalr`	Jump and link address relative to a register or the PC

RISC-V Continuous Evolution



Recently Ratified RISC-V ISA Extensions

Specification name	Ratification date	New extension(s)
PMP Enhancements for memory access and execution prevention on Machine mode (Smepmp)	November 2021	Smepmp
RISC-V Base Cache Management Operation ISA Extensions	November 2021	Zicbom, Zicbop, Zicboz
RISC-V Bit-Manipulation ISA-extensions	November 2021	Zba, Zbb, Zbc, Zbs
RISC-V Count Overflow and Mode-Based Filtering Extension	November 2021	Sscofpmf
RISC-V Cryptography Extensions Volume I: Scalar & Entropy Source Instructions	November 2021	Zbkb, Zbkc, Zbkx, Zknd, Zkne, Zknh, Zksed, Zksh, Zkn, Zks, Zkt, Zk, Zkr
RISC-V State Enable Extension	November 2021	Smstateen
RISC-V "stimecmp / vstimecmp" Extension	November 2021	Sstc
RISC-V Vector Extension	November 2021	Zve32x, Zve32f, Zve64x, Zve64f, Zve64d, Zve, Zvl32b, Zvl64b, Zvl128b, Zvl256b, Zvl512b, Zvl1024b, Zvl, Zv
The RISC-V Instruction Set Manual Volume II: Privileged Architecture	November 2021	Sm1-12, Ss1-12, Sv57, Hypervisor, Svinval, Svnapot, Svpbmt
"Zfh" and "Zfhmin" Standard Extensions for Half-Precision Floating-Point	November 2021	Zfh, Zfhmin
"Zfinx", "Zdinx", "Zhinx", "Zhinxmin": Standard Extensions for Floating-Point in Integer Registers	November 2021	Zfinx, Zdinx, Zhinx, Zhinxmin

https://wiki.riscv.org/display/HOME/Recently+Ratified+Extensions

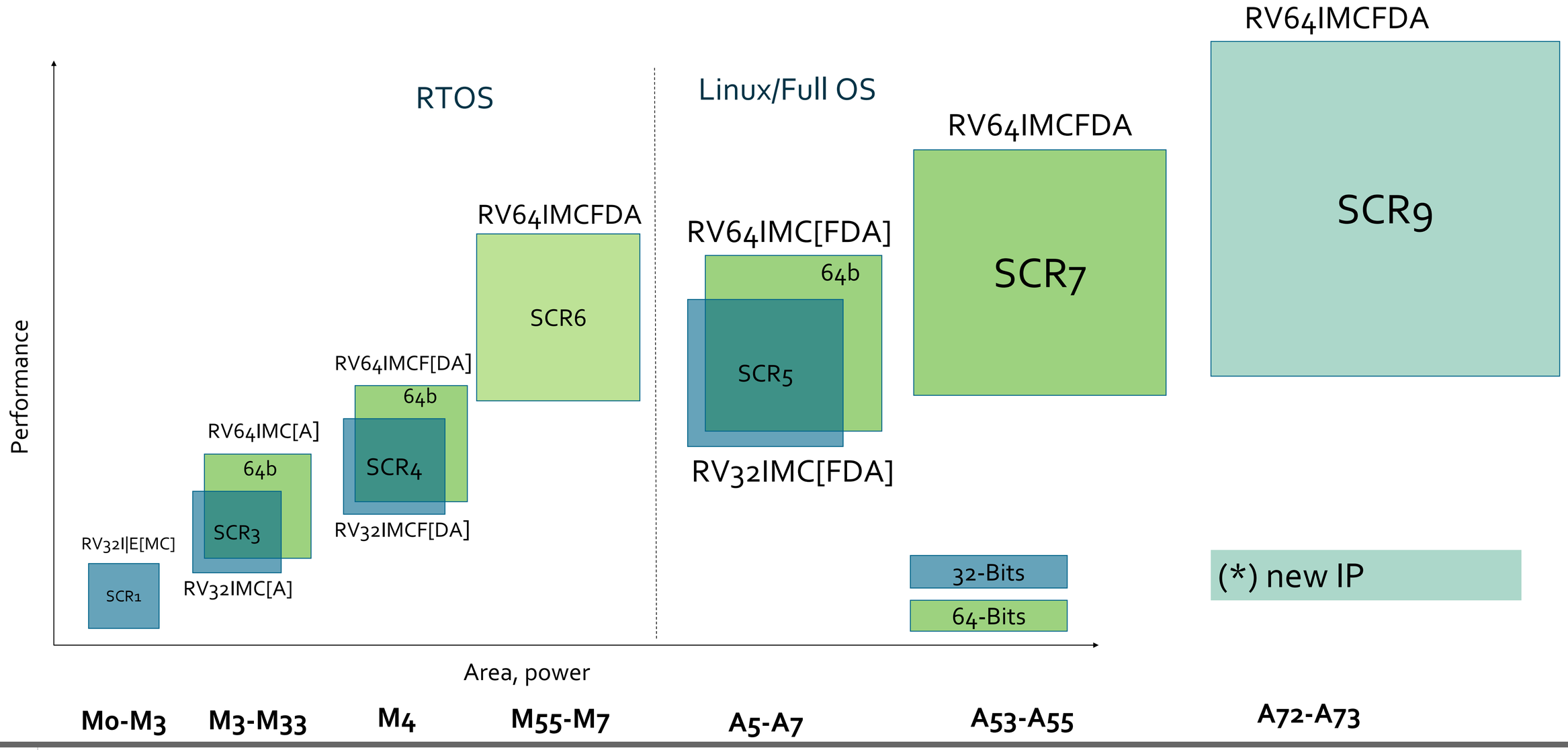
RISC-V ISA Profiles and Platforms

Profiles describe small common set of ISA extensions that capture the most value for most users (domain)

Platforms capture profile and complete execution environment for SW stack

extension name	ratification package name	IC/TG	description (what this does, in English)	ratified year	RVI20	RVA20 (64 only)			RVM20			RVA22 (64 only)			RVM22		
						Mode			Mode			Mode			Mode		
D	D	unpriv	floating point, double-precision (implies F)	2019	s	m	m	s	s	s	s	m	m	s	s	s	s
F	F	unpriv	floating point, single-precision	2019	s	m	m	s	s	s	s	m	m	s	s	s	s
H	Hypervisor	priv	h mode priv instructions and state	2021	x	x	x	x	x	x	x	s	s	s	u	u	u
I	RVI32	unpriv	integer base for RVI32	2019	m	m	m	m	m	m	m	m	m	m	m	m	m
E	RVE32	unpriv	integer base for RVI32	2019	m	i	i	i	m	m	m	i	i	i	m	m	m
I	RVI64	unpriv	integer base fro RVI64	2019	m	m	m	m	m	m	m	m	m	m	m	m	m
M	M	unpriv	multiply/divide	2019	s	m	m	m	s	s	s	m	m	m	s	s	s
Q	Q	unpriv	floating point, quad-precision (implies F&D)	2019	u	u	u	u	u	u	u	u	u	u	u	u	u
Sm1p11	priv 1.11	priv	m mode priv instructions and state	2019	s	n	n	m	n	n	m	n	n	i	n	n	i
Sm1p12	priv 1.12	priv	m mode priv instructions and state	2021	x	x	x	x	x	x	x	n	n	m	n	n	m
Smepmp	Enhanced PMP	TEE	trusted address map TEE extension	2021	x	x	x	x	x	x	x	n	n	s	n	n	s
Smstateen	State Enable	priv	Enable/disable state access by lower privielege modes (for security reasons)	2021	x	x	x	x	x	x	x	n	s	s	n	s	s
Ss1p11	priv 1.11	priv	s mode priv instructions and state	2019	s	n	m	m	n	s	s	n	i	i	n	i	i
Ss1p12	priv 1.12	priv	s mode priv instructions and state	2021	x	x	x	x	x	x	x	n	m	m	n	s	s
Sv57	Sv57	virtual memory	57-bit VA translation	2021	x	x	x	x	x	x	x	n	s	s	n	s	s
Sscofpmf	Count overlow & Filtering	priv	hpm counter overflow & privilege mode filtering	2021	x	x	x	x	x	x	x	n	s	s	n	s	s
Sstc	time cmp	priv	stimecmp/vstimecmp - s-mode timers so don't need M-mode sbi call	2021	x	x	x	x	x	x	x	n	s	s	n	s	s
Svinval	Fast TLB invalidation	virtual memory	fast multi-sfence.vma's	2021	x	x	x	x	x	x	x	n	s	s	n	s	s
Svnapot	NAPOT pages	virtual memory	naturally aligned power of two (for pages)	2021	x	x	x	x	x	x	x	n	s	s	n	s	s
Svpbmt	Page-based memory types	virtual memory	page-based memory types (e.g. cacheable, uncacheable, etc.)	2021	x	x	x	x	x	x	x	n	s	s	n	s	s
Zicntr	Counters	unpriv	XXX priv counter in priv spec. is this just of RVI?	2019	s	m	m	m	s	s	s	m	m	m	s	s	s
Zihpm	Priv 1.11	priv	s mode priv instructions and state	2019	s	m	m	m	s	s	s	m	m	m	s	s	s
V	V group 1	vector	all the basic instructions (beyond those below)	2021	x	x	x	x	x	x	x	s	s	s	s	s	s
Zaamo	A	unpriv	AMOs	2019	s	m	m	m	s	s	s	m	m	m	s	s	s
Zalrsc	A	unpriv	LR/Store Conditional atomics	2019	s	m	m	m	s	s	s	m	m	m	s	s	s

Example RISC-V CPU IP: SCRx

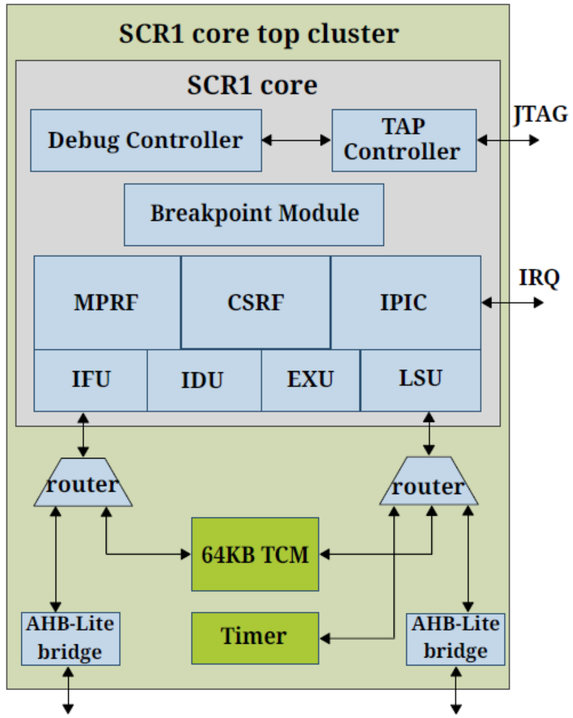


SCR1 overview

Industry-grade compact MCU core for deeply embedded applications and accelerator control

- RV32I|E[MC] ISA
- 2 to 4 stages pipeline
- M-mode only
- Optional configurable IPIC
- Optional integrated Debug Controller
- Choices of the optional MUL/DIV unit
- Open sourced under SHL (Apache 2.0 derivative) since 2017
  - **Unrestricted commercial use allowed**
- High quality, silicon-proven free MCU IP
- **In the top System Verilog Github repos in the world**
  - https://github.com/syntacore/scr1
- Best-effort support provided, commercial offered

Open & free



Standard configs: **10..40+** kGates
Default RV32IMC config: **32** kGates
Minimal RV32EC config: **11** kGates
250+ MHz @ tsmc90lp {typical, 1.0V, +25C}

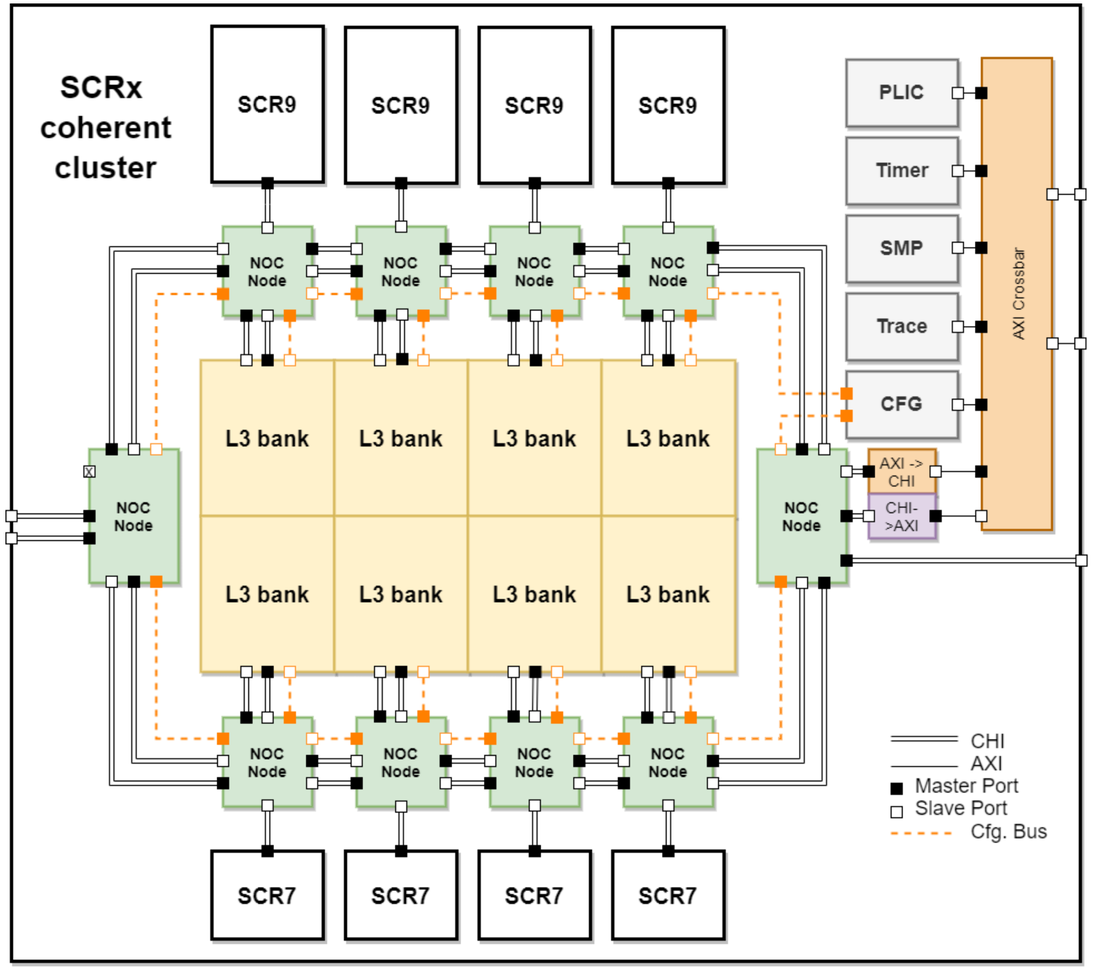
Performance*, per MHz	DMIPS	-O2	1.28
		-best**	1.89
	Coremark	-best**	2.95

* Dhrystone 2.1, Coremark 1.0, GCC 8.1 BM from TCM
** -O3 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -flto

Introducing SCR9

Linux-capable application CPU with entry-level server class features:

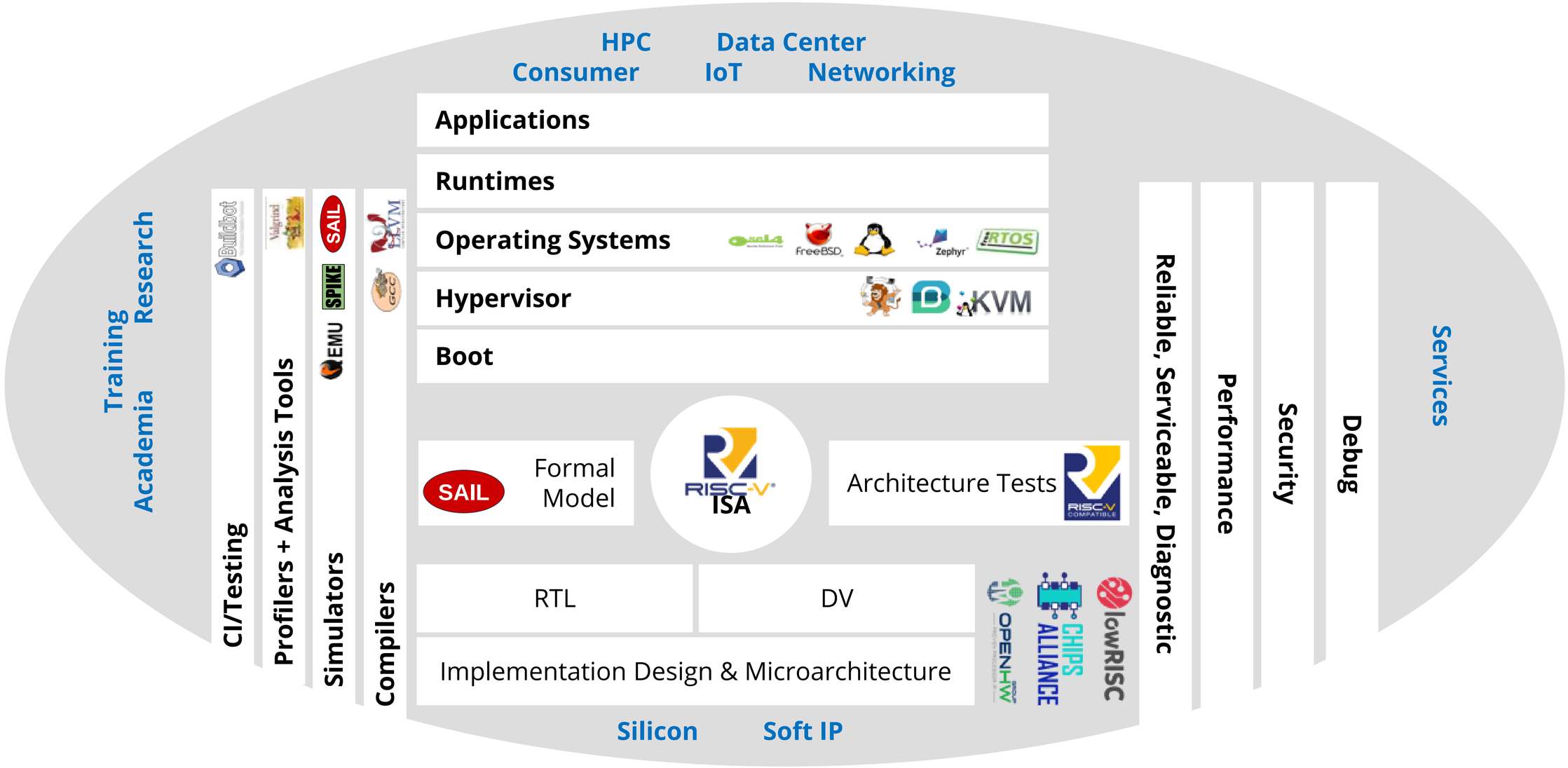
- 8-16 cores per cluster
  - SMP and heterogeneous
- Coherent NoC-based L3
- CHI external i/f
- SV39, SV48
- RVV
- Hypervisor
- AIA
- Accelerators support
- 7+ CM/MHz per core
- 2GHz @12nm



Early access program* starting Q3'22

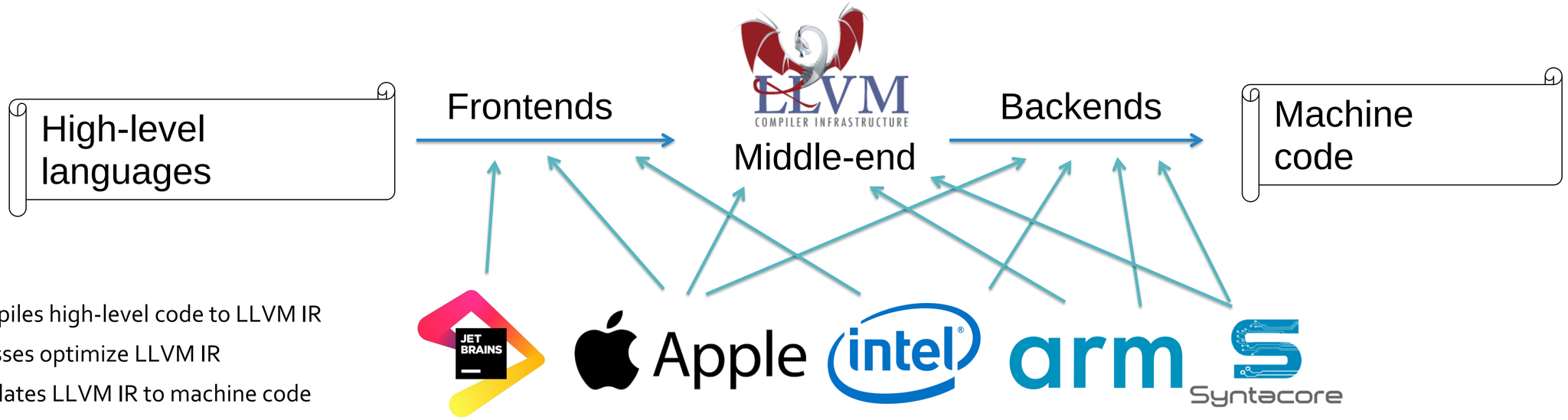
(*) some features may be not available in the initial release

RISC-V Ecosystem



LLVM Compiler Framework

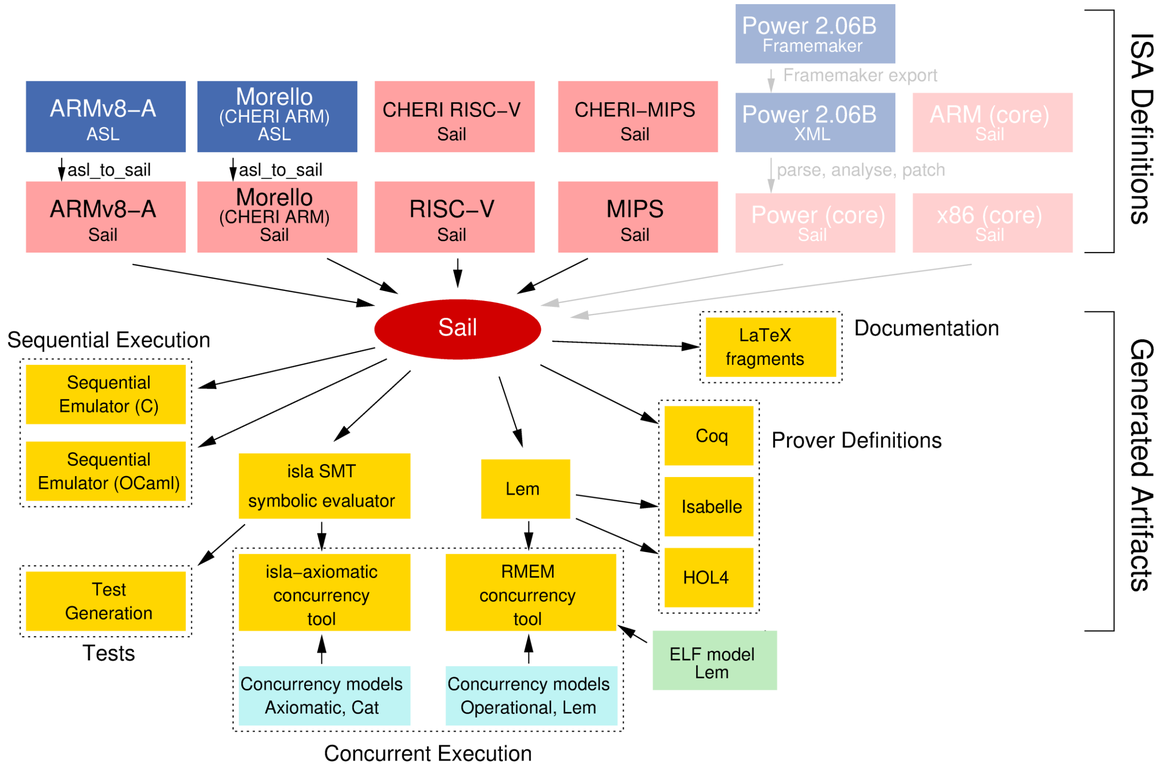
- LLVM is open-source compiler/framework: 20+ years, ~500,000 commits, 1500+ developers
- Used as primary compiler for many languages and companies developing languages, code analyzers, processor architectures, OS
- Depending on company profile, it might contribute to different LLVM components



- **Front-end** compiles high-level code to LLVM IR
- **Middle-end** passes optimize LLVM IR
- **Back-end** translates LLVM IR to machine code

Simulators

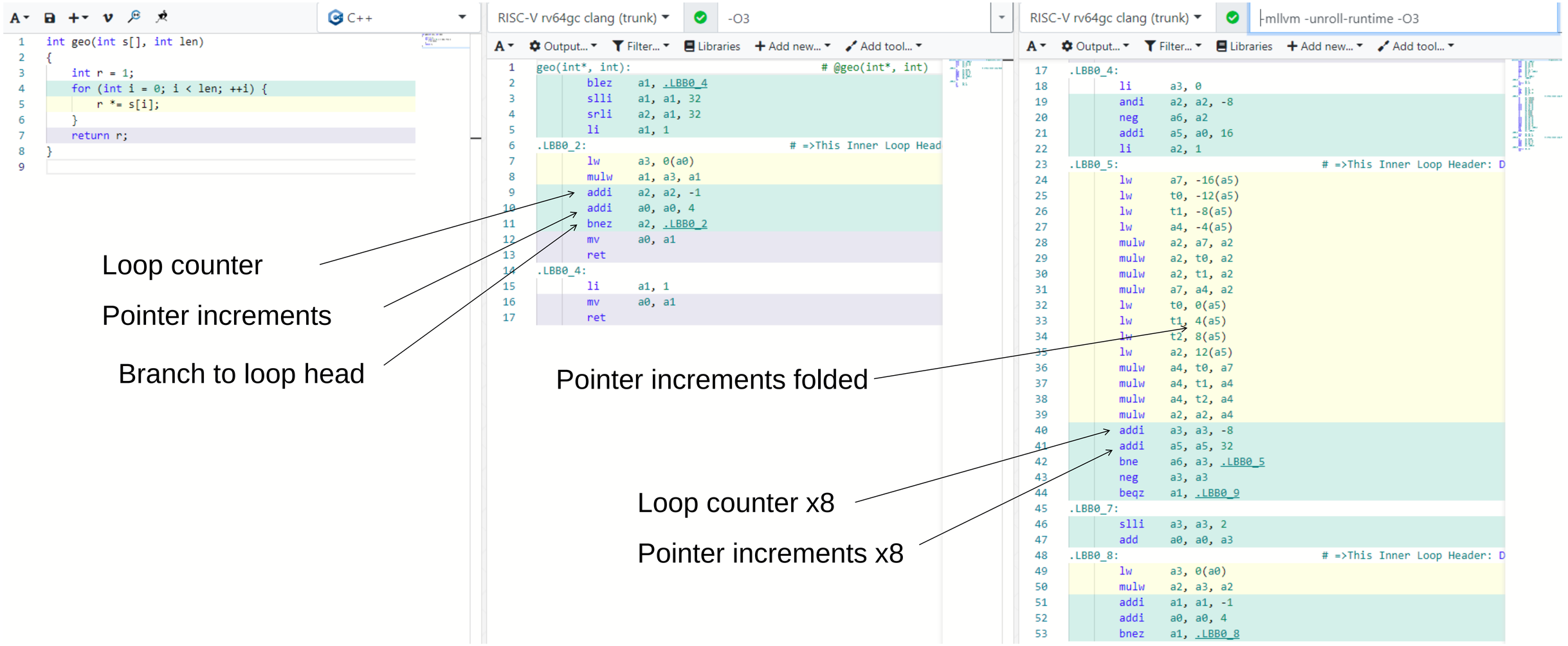
- ISA "Golden Model" for spec and HW Verification
  - Spike: legacy model
  - SAIL: new ISA-level
- Performance and Design Exploration
  - Sparta/Coyote on Spike/SAIL
  - MARSS++
  - GEM5
- System SW prototyping
  - QEMU



Code Optimization: Unroll Loops

Unrolling Runtime Loops: folds addi on address/counter

-mllvm -unroll-runtime




```
1  int geo(int s[], int len)
2  {
3      int r = 1;
4      for (int i = 0; i < len; ++i) {
5          r *= s[i];
6      }
7      return r;
8  }
9
```

```
1  geo(int*, int):                          # @geo(int*, int)
2          blez    a1, .LBB0_4
3          slli    a1, a1, 32
4          srli    a2, a1, 32
5          li      a1, 1
6  .LBB0_2:                                 # =>This Inner Loop Head
7          lw      a3, 0(a0)
8          mulw    a1, a3, a1
9          addi    a2, a2, -1
10         addi    a0, a0, 4
11         bnez    a2, .LBB0_2
12         mv      a0, a1
13         ret
14 .LBB0_4:
15         li      a1, 1
16         mv      a0, a1
17         ret
```

```
17 .LBB0_4:
18         li      a3, 0
19         andi    a2, a2, -8
20         neg     a6, a2
21         addi    a5, a0, 16
22         li      a2, 1
23 .LBB0_5:                                 # =>This Inner Loop Header: D
24         lw      a7, -16(a5)
25         lw      t0, -12(a5)
26         lw      t1, -8(a5)
27         lw      a4, -4(a5)
28         mulw    a2, a7, a2
29         mulw    a2, t0, a2
30         mulw    a2, t1, a2
31         mulw    a7, a4, a2
32         lw      t0, 0(a5)
33         lw      t1, 4(a5)
34         lw      t2, 8(a5)
35         lw      a2, 12(a5)
36         mulw    a4, t0, a7
37         mulw    a4, t1, a4
38         mulw    a4, t2, a4
39         mulw    a2, a2, a4
40         addi    a3, a3, -8
41         addi    a5, a5, 32
42         bne     a6, a3, .LBB0_5
43         neg     a3, a3
44         beqz    a1, .LBB0_9
45 .LBB0_7:
46         slli    a3, a3, 2
47         add     a0, a0, a3
48 .LBB0_8:                                 # =>This Inner Loop Header: D
49         lw      a3, 0(a0)
50         mulw    a2, a3, a2
51         addi    a1, a1, -1
52         addi    a0, a0, 4
53         bnez    a1, .LBB0_8
```

Code Optimization: Vectorization

Vectorize scalar and loops using RVV 1.0 ISA

https://github.com/riscv/riscv-v-spec

- RVV 1.0 ISA – high-performance and efficient vector
- Scales from small to server/accelerator cores
- Strip-mined loops, strided loads, scatter-gather, reductions
- Can be extended for specific domains

```
void sum_slp(int *in1, int *in2, int *out) {
    int a0 = in1[0];
    int a1 = in1[1];
    int a2 = in1[2];
    int a3 = in1[3];

    int b0 = in2[0];
    int b1 = in2[1];
    int b2 = in2[2];
    int b3 = in2[3];

    out[0] = a0 + b0;
    out[1] = a1 + b1;
    out[2] = a2 + b2;
    out[3] = a3 + b3;
}

void sum_loop(int * __restrict in1,
              int  * __restrict in2, int * out) {
    for (int i = 0; i < 4; i++)
        out[i] = in1[i] + in2[i];
}
```

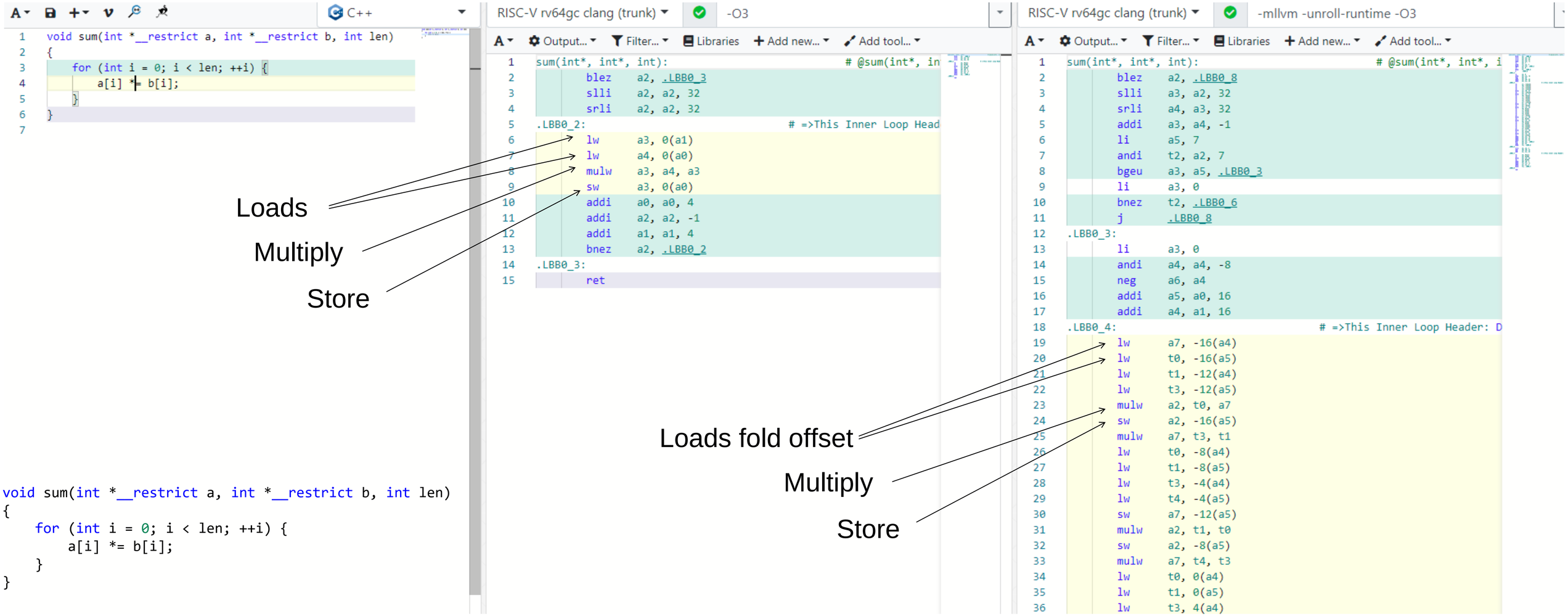
```
define void @sum_slp(...) {
  %4 = load <4 x i32>, ptr %0
  %5 = load <4 x i32>, ptr %1
  %6 = add nsw <4 x i32> %5, %4
  store <4 x i32> %6, ptr %2
  ret void
}

define void @sum_loop(...) {
  %4 = load <4 x i32>, ptr %0
  %5 = load <4 x i32>, ptr %1
  %6 = add nsw <4 x i32> %5, %4
  store <4 x i32> %6, ptr %2
  ret void
}
```

```
sum_slp(int*, int*, int*):
        # VLEN/SEW is set
        vlw.v       v0, (a0)
        vlw.v       v1, (a1)
        vadd.vv     v2, v0, v1
        vsw.v       v2, (a2)
        ret
sum_loop(int*, int*, int*):
        # VLEN/SEW is set
        vlw.v       v0, (a0)
        vlw.v       v1, (a1)
        vadd.vv     v2, v0, v1
        vsw.v       v2, (a2)
        ret
```

https://godbolt.org

Code Optimization: Vectorization

Vectorize scalar and loops using RVV 1.0 ISA




```
1  void sum(int *__restrict a, int *__restrict b, int len)
2  {
3      for (int i = 0; i < len; ++i) {
4          a[i] *= b[i];
5      }
6  }
7
```

```
1  sum(int*, int*, int):                              # @sum(int*, in
2          blez    a2, .LBB0_3
3          slli    a2, a2, 32
4          srli    a2, a2, 32
5  .LBB0_2:                                           # =>This Inner Loop Head
6          lw      a3, 0(a1)
7          lw      a4, 0(a0)
8          mulw    a3, a4, a3
9          sw      a3, 0(a0)
10         addi    a0, a0, 4
11         addi    a2, a2, -1
12         addi    a1, a1, 4
13         bnez    a2, .LBB0_2
14 .LBB0_3:
15         ret
```

```
1  sum(int*, int*, int):                              # @sum(int*, int*, i
2          blez    a2, .LBB0_8
3          slli    a3, a2, 32
4          srli    a4, a3, 32
5          addi    a3, a4, -1
6          li      a5, 7
7          andi    t2, a2, 7
8          bgeu    a3, a5, .LBB0_3
9          li      a3, 0
10         bnez    t2, .LBB0_6
11         j       .LBB0_8
12 .LBB0_3:
13         li      a3, 0
14         andi    a4, a4, -8
15         neg     a6, a4
16         addi    a5, a0, 16
17         addi    a4, a1, 16
18 .LBB0_4:                                           # =>This Inner Loop Header: D
19         lw      a7, -16(a4)
20         lw      t0, -16(a5)
21         lw      t1, -12(a4)
22         lw      t3, -12(a5)
23         mulw    a2, t0, a7
24         sw      a2, -16(a5)
25         mulw    a7, t3, t1
26         lw      t0, -8(a4)
27         lw      t1, -8(a5)
28         lw      t3, -4(a4)
29         lw      t4, -4(a5)
30         sw      a7, -12(a5)
31         mulw    a2, t1, t0
32         sw      a2, -8(a5)
33         mulw    a7, t4, t3
34         lw      t0, 0(a4)
35         lw      t1, 0(a5)
36         lw      t3, 4(a4)
```

```
void sum(int *__restrict a, int *__restrict b, int len)
{
    for (int i = 0; i < len; ++i) {
        a[i] *= b[i];
    }
}
```

Code Optimization: Vectorization

Vectorize scalar and loops using RVV 1.0 ISA

-march=rv64gcv (`v` enables RVV)
-mllvm -riscv-v-vector-bits-min=128
-mllvm -riscv-v-fixed-length-vector-lmul-max=1

```
1  void sum(int *__restrict a, int *__restrict b, int len)
2  {
3      for (int i = 0; i < len; ++i) {
4          a[i] *= b[i];
5      }
6  }
7
```

```
14          mv      a3, a1
15  .LBB0_4:                                # =>This Inner Loop Head
16          addi    a2, a5, 16
17          addi    t0, a3, 16
18          vle32.v v8, (a3)
19          vle32.v v9, (t0)
20          vle32.v v10, (a5)
21          vle32.v v11, (a2)
22          vmul.vv v8, v10, v8
23          vmul.vv v9, v11, v9
24          vse32.v v8, (a5)
25          vse32.v v9, (a2)
26          addi    a3, a3, 32
27          addi    a4, a4, -8
28          addi    a5, a5, 32
29          bnez    a4, .LBB0_4
30          beq     a7, a6, .LBB0_8
31  .LBB0_6:
32          slli    a2, a7, 2
33          add     a0, a0, a2
34          add     a1, a1, a2
35          sub     a2, a6, a7
36  .LBB0_7:                                # =>This Inner Loop Head
37          lw      a3, 0(a1)
38          lw      a4, 0(a0)
39          mulw    a3, a4, a3
40          sw      a3, 0(a0)
41          addi    a0, a0, 4
42          addi    a2, a2, -1
43          addi    a1, a1, 4
44          bnez    a2, .LBB0_7
45  .LBB0_8:
46          ret
```

Loads

Multiply

Store

```
24  .LBB0_6:                                # =>This Inner Loop Header:
25          add     a5, a1, t3
26          vle32.v v8, (a5)
27          addi    a4, a5, 16
28          vle32.v v9, (a4)
29          add     a4, a0, t3
30          vle32.v v10, (a4)
31          addi    a2, a4, 16
32          vle32.v v11, (a2)
33          vmul.vv v8, v10, v8
34          vmul.vv v9, v11, v9
35          vse32.v v8, (a4)
36          vse32.v v9, (a2)
37          addi    a2, a5, 32
38          vle32.v v8, (a2)
39          addi    a2, a5, 48
40          vle32.v v9, (a2)
41          addi    a2, a4, 32
42          vle32.v v10, (a2)
43          addi    a3, a4, 48
44          vle32.v v11, (a3)
45          vmul.vv v8, v10, v8
46          vmul.vv v9, v11, v9
47          vse32.v v8, (a2)
48          vse32.v v9, (a3)
49          addi    a2, a5, 64
50          vle32.v v8, (a2)
51          addi    a2, a5, 80
52          vle32.v v9, (a2)
53          addi    a2, a4, 64
54          vle32.v v10, (a2)
55          addi    a3, a4, 80
56          vle32.v v11, (a3)
```

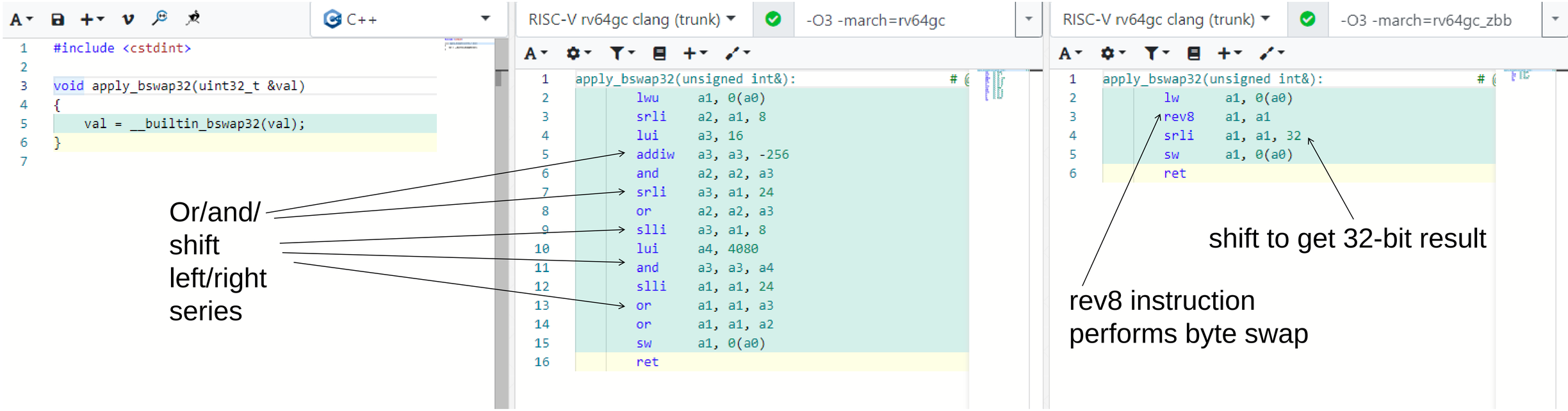
Pointer inc

Loads

Code Optimization: Bit Manipulation

Bit Manipulation ISA extension (B, Zba/Zbb/etc.)

-march=rv64g_Zbb

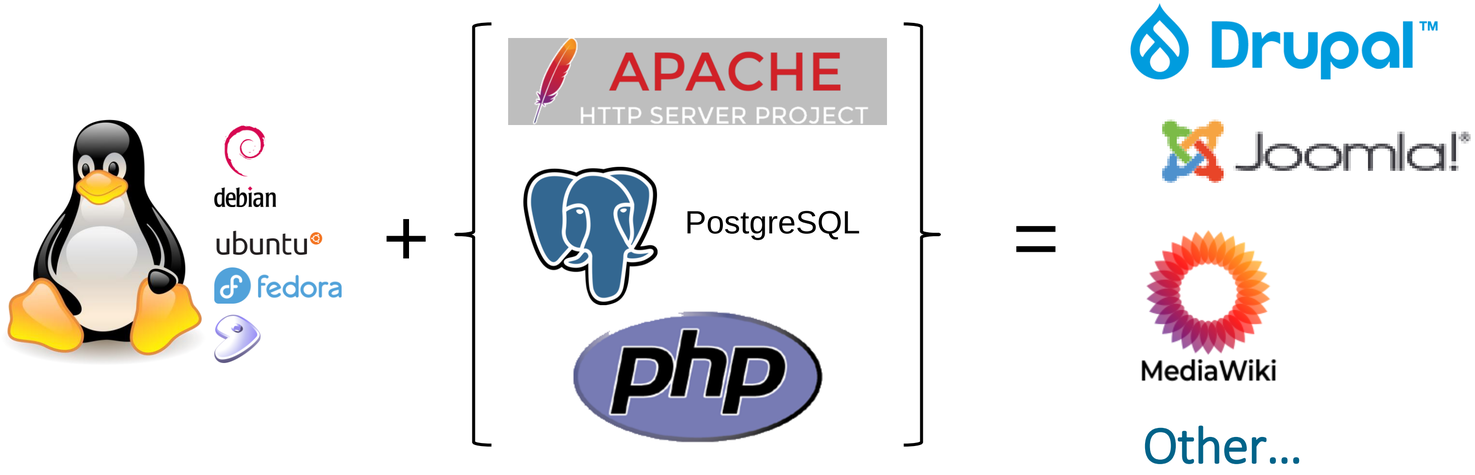


Example DC-class SW stack

LAPP = Linux + Apache + PostgreSQL + PHP

✓ pillar enabling technology for web server applications

✓ running on SCRx cores in the lab



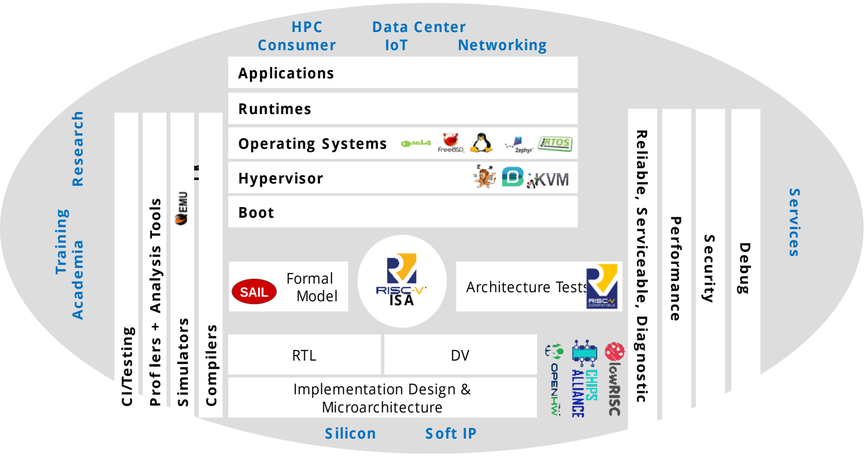
Thank you!

www.syntacore.com

info@syntacore.com

Development Tools and System Software

Modular and open ISA architecture creates many challenges for tools development, system SW and OS



Toolchains (GCC, LLVM, binutils)	GCC toolchain, LLVM optimizations for new ISA (RVV vectors) General performance, benchmarks CoreMark/SPEC Retargeting using ADL (architecture description)
Java VM and runtime	OpenJDK for RISC-V, optimizations for SCR cores RISC-V J extension
Debugging and Profiling	GDB, OOCD, LLDB, perf JVM tooling
Simulator (SW, verif, perf)	ISA-level golden model simulator (Spike, SAIL) Performance modelling (MARSS), SW dev (QEMU)
IDE	Eclipse, basic VSCode basic CLion
Virtualization	Investigation & optimizations (KVM, etc.)
Apps & Bench	CoreMark/SPEC benchmark analysis, customer workloads Focus on storage, servers, tablet, comms and AI
BSP, Libs and OS	Libs, Linux, Debian/Ubuntu, OpenSBI, Uboot FreeRTOS, Yocto/OpenWRT
Testing, Benchmarking, Verification and Automation	CI and automation, testing and benchmarking frameworks Verification tools.

Example RISC-V CPU IP:

Features			SCR1* (FREE!)	SCR3	SCR4	SCR6	SCR5	SCR7
			RTOS/ Bare Metal				Linux/ "Full" OS	
Width		32bit	●	●	●		●	
		64bit		●	●	●	●	●
ISA			RV32I|E[MC]	RV[32|64]IMC[A]	RV[32|64]IMCF[AD]	RV64IMCAFD	RV[32|64]IMC[AFD]	RV64IMCAFD
Pipeline type			In-order	In-order	In-order	Dual-issue, In-order	In-order	Superscalar
Pipeline, stages			2-4	3-5	3-5	10-12	7-9	10-12
Branch prediction				Static BP, RAS	Static BP, RAS	Dynamic BP, BTB, BHT, RAS	Static BP, BTB, BHT, RAS	Dynamic BP, BTB, BHT, RAS
Execution priority levels			Machine	User, Machine	User, Machine	User, Machine	User, Supervisor, Machine	User, Supervisor, Machine
Extensibility/customization			●	●	●	●	●	●
Execution units	MUL/DIV	area-opt	●	○	○	○		
		hi-perf	○	●	●	●	●	●
	FPU				●	● [hi-perf opt]	●	● [hi-perf opt]
Memory subsystem	TCM [w/ECC|parity]		●	●	●	●	●	○
	L1$ [w/ECC|parity]			○	○	●	●	●
	L2$ w/ECC			○	○	○	○	●
	MPU/PMP			●	●	●	●	●
	MMU, virtual memory						●	●
Debug	Integrated JTAG debug		●	●	●	●	●	●
	HW BP		1-2	1-8 adv ctrl	1-8 adv ctrl	1-8 adv ctrl	1-8 adv ctrl	1-8 adv ctrl
	Performance counters		○	●	●	●	●	●
Interrupt Controller	IRQs		8-32	8-1024	8-1024	8-1024	8-1024	8-1024
	Features		basic	advanced	advanced	advanced	advanced+	advanced+
SMP support				up to 4 cores with coherency				up to 8-16 cores
I/F options	AHB		●	○	○	○	○	○
	AXI		○	●	●	●	●	●
	ACE / CHI							○

* Download SCR1 free at www.github.com/syntacore/scr1

● – default, ○ – configurable option; ISA options: I – Integer instruction set; E – Embedded subset (16 registers); M – Integer multiply and divide; A – Atomic memory operations, load-reserve/store conditional; C – Compressed integer instructions, reduces size to 16 bits; F/D – single/double precision (32/64 bit) floating point.

Baseline cores:

- Clean-slate designs in System Verilog
- Configurable and extensible
- 100% compatible with major EDA flows
- Silicon-proven at the customers