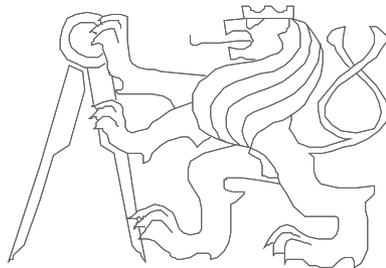


Advanced Computer Architectures

RISC architectures examples – ARM, AArch64 a RISC-V



Czech Technical University in Prague, Faculty of Electrical Engineering
Slides authors: Pavel Píša, Michal Štepanovský

ARM architecture - registers

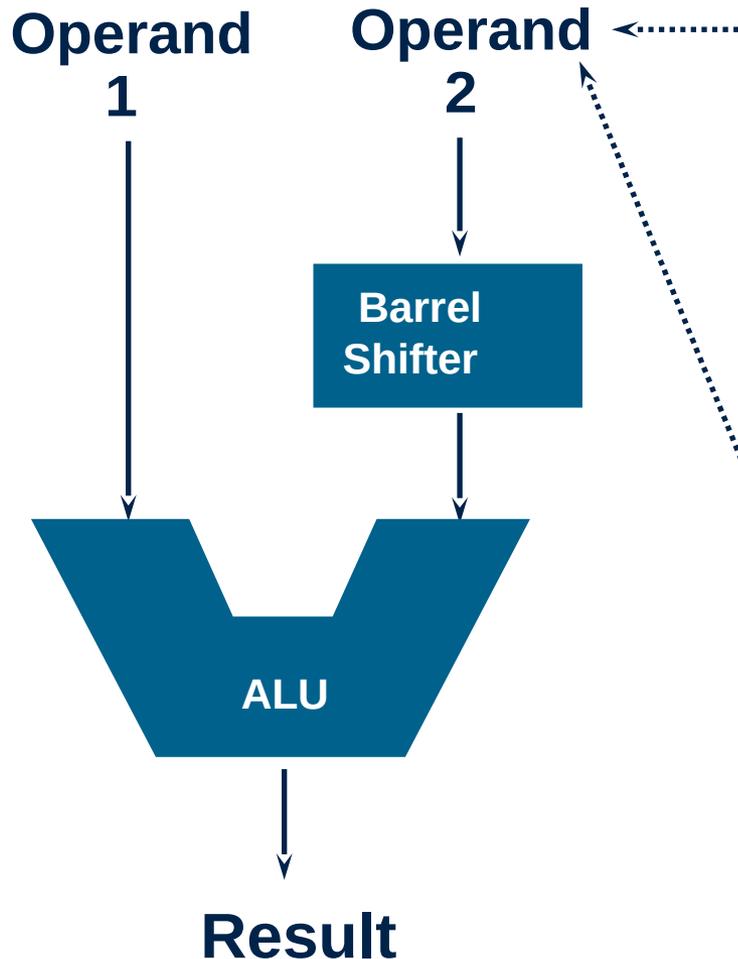
Current Visible Registers

Abort Mode	r0
	r1
	r2
	r3
	r4
	r5
	r6
	r7
	r8
	r9
	r10
	r11
	r12
	r13 (sp)
	r14 (lr)
	r15 (pc)
cpsr	
spsr	

Banked out Registers

User	FIQ	IRQ	SVC	Undef
	r8			
	r9			
	r10			
	r11			
	r12			
r13 (sp)				
r14 (lr)				
	spsr	spsr	spsr	spsr

ARM architecture – ALU and operands encoding



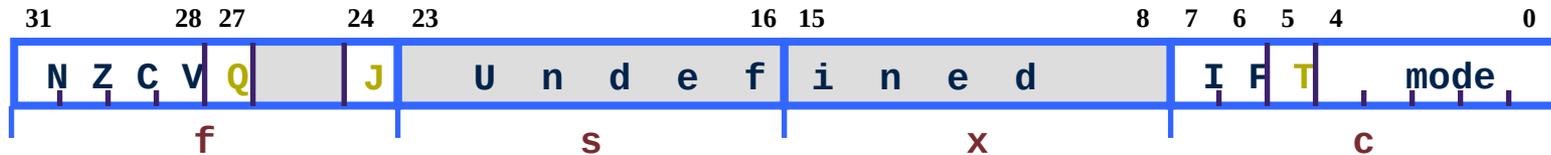
Register, optionally with shift operation

- Shift value can be either be:
 - 5 bit unsigned integer
 - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value

- 8 bit number, with a range of 0-255.
 - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers

ARM architecture – program status word



- **Condition code flags**
 - **N** = **N**egative result from ALU
 - **Z** = **Z**ero result from ALU
 - **C** = ALU operation **C**arried out
 - **V** = ALU operation **o**Verflowed
- **Sticky Overflow flag - Q flag**
 - Architecture 5TE/J only
 - Indicates if saturation has occurred
- **J bit**
 - Architecture 5TEJ only
 - **J** = 1: Processor in Jazelle state
- **Interrupt Disable bits.**
 - **I** = 1: Disables the IRQ.
 - **F** = 1: Disables the FIQ.
- **T Bit**
 - Architecture xT only
 - **T** = 0: Processor in ARM state
 - **T** = 1: Processor in Thumb state
- **Mode bits**
 - Specify the processor mode

ARM architecture – CPU execution modes

- User : unprivileged mode under which most tasks run
- FIQ : entered when a high priority (fast) interrupt is raised
- IRQ : entered when a low priority (normal) interrupt is raised
- Supervisor : entered on reset and when a Software Interrupt instruction is executed
- Abort : used to handle memory access violations
- Undef : used to handle undefined instructions
- System : privileged mode using the same registers as user mode

ARM 64-bit – AArch64

- Calling uses LR, no register banking, ELR for exceptions
- PC is separate register (not included in general purpose registers file)
- 31 64-bit registers R0 to R30 (R30 = X30 \cong LR)
 - Symbol Wn ($W0$) used for 32-bit access, Xn ($X0$) for 64-bit
 - Reg. code 31 same role as MIPS 0, WZR/XZR in code
 - Reg. code 31 special meaning as WSP, SP for some opcodes
- Immediate operand 12-bit with optional LS 12 for arithmetic operations and repetitive bit masks generator for logic ones
- 32-bit operations ignore bits 32–63 for source and zeroes these in the destination register

AArch64 – Branches and conditional operations

- Omitted conditional execution in all instructions as well as Thumb IT mechanism
- Conditional register retain, CBNZ, CBZ, TBNZ, TBZ added
- Only couple of conditional instructions
 - add and sub with carry, select (move C?A:B)
 - set 0 and 1 (or -1) according to the condition evaluation
 - conditional compare instruction
- 32-bit and 64-bit multiply and divide (3 registers), multiply with addition
64×64+64 ÷ 64 (four registers), high bits 64 to 127 from 64×64 multiplication

AArch64 – Memory access

- 48+1 bit address, sign extended to 64 bits
- Immediate offset can be multiplied by access size optionally
- If register is used in index role, it can be multiplied by access size and can be limited to 32 bits
- PC relative $\pm 4\text{GB}$ can be encoded in 2 instructions
- Only pair of two independent registers LDP and STP (omitted LDM, STM), added LDNP, STNP
- Unaligned access support
- LDX/STX(RBHP) for 1,2,4,8 and 16 bytes exclusive access

AArch64 – Address modes

- Simple register (exclusive)
[base{,#0}]
- Offset
[base{,#imm}] – Immediate Offset
[base,Xm{,LSL #imm}] – Register Offset
[base,Wm,(S|U)XTW {#imm}] – Extended Register Offset
- Pre-indexed
[base,#imm]!
- Post-indexed
[base],#imm
- PC-relative (literal) load
label

Bits	Sign	Scaling	WBctr	LD/ST type
0	-	-	-	LDX, STX, acquire, release
9	signed	scaled	option	reg. pair
10	signed	unscaled	option	single reg.
12	unsig.	scaled	no	single reg.

RISC-V – optimize and simplify RISC again

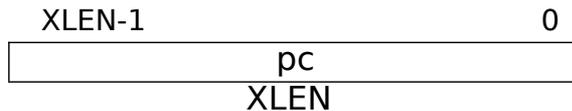
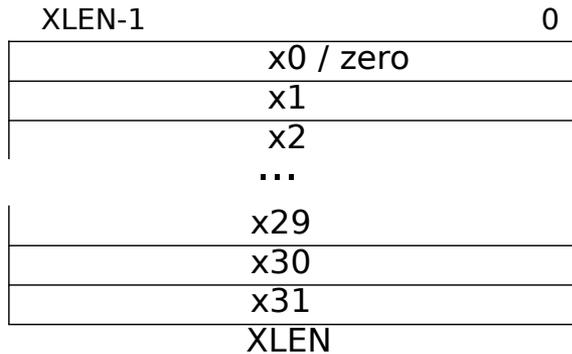
- Patterson, Berkeley RISC 1984 → initiation of RISC era, evolved into **SPARC** (Hennessy **MIPS**, Stanford University)
- Commercialization and extensions results in too complex CPUs again, with license and patents preventing even the original inventors to use real/actual implementations in silicon to be used for education and research
- MIPS is model architecture for prevalent amount of base courses and implementation of similar processor is part of follow up courses (A4M36PAP)
- Krste Asanovic and other Dr. Patterson's students initiated development of new architecture (start of 2010)
- BSD License to ensure openness in the future
- Supported by GCC, binutils, Linux, QEMU, etc.
- Simpler than SPAC, more like MIPS but optimized on gate level load (fanout) and critical paths lengths in future designs
- Some open implementations already exist: **Rocket** (SiFive, BOOM), project **lowRISC** contributes to research in security area, in ČR Codasip
- Already more than 15 implementations in silicon

RISC-V – architecture specification

- ISA specification can be found at
 - The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0
 - Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic
 - Not only architecture description but even choices analysis with pros&cons of each selection and cost source description/analysis of alternatives
- classic design, 32 integer registers, the first tied to zero. regsrc1, regsrc2, regdest operands, uniqueness, rule kept strictly even for SaveWord, leads to non-continuous immediate operands encoding, PC not part of base register file, PC-relative addressing
- variants for 32, 64 a 128-bit registers and address-space defined
- high code density (16-bit instruction encoding variant planned)
- encoding reserves space for floating point (single, double, quad) and multimedia SIMD instructions in a systematic way, etc.

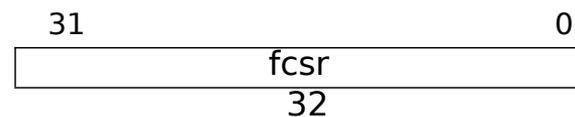
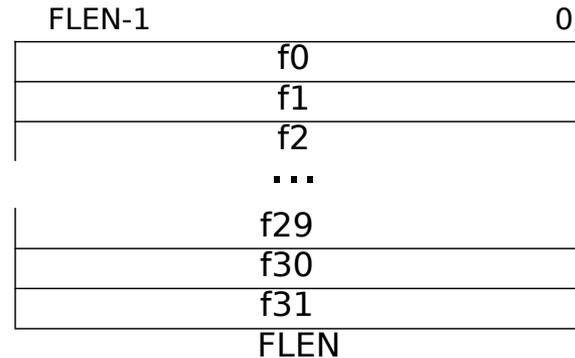
RISC-V – registers

Integer registers



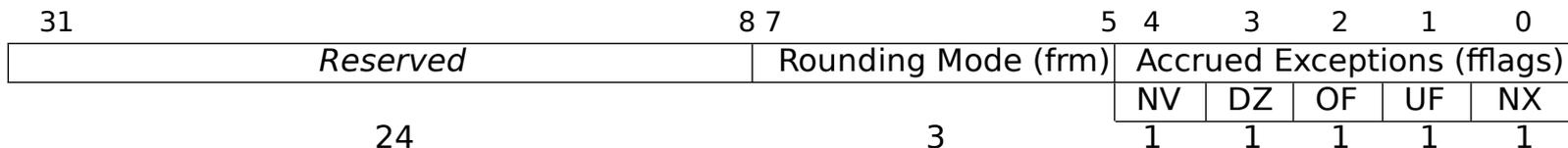
Variant	XLEN
RV32	32
RV64	64
RV128	128

Floating point registers



Variant	FLEN
F	32
D	64
Q	128

Floating-point control and status register



Source: <https://riscv.org/specifications/>

RISC-V – instruction length encoding

xxxxxxxxxxxxxxxxaa 16-bit (aa \neq 11)

xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxbbb11 32-bit (bbb \neq 111)

· · ·xxxx xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxx011111 48-bit

· · ·xxxx xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxx0111111 64-bit

· · ·xxxx xxxxxxxxxxxxxxxxxxxx xnnnxxxxx1111111 (80+16*nnn)-bit, nnn \neq 11:

· · ·xxxx xxxxxxxxxxxxxxxxxxxx x111xxxxx1111111 Reserved for \geq 192-bits

Address:

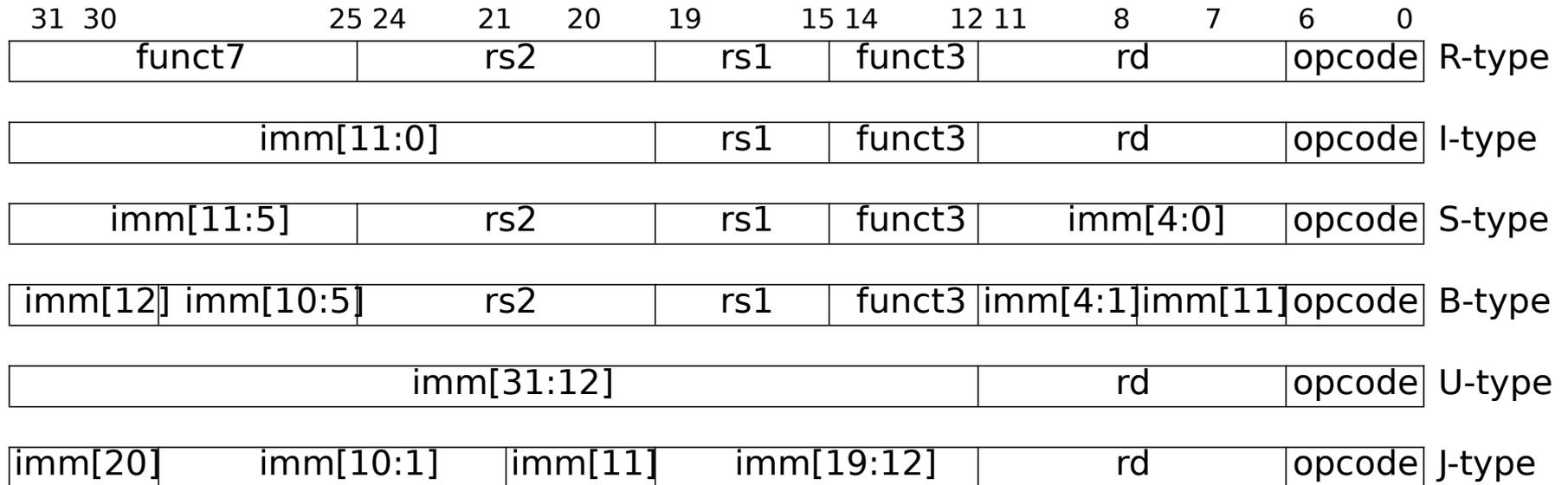
base+4

base+2

base

Source: <https://riscv.org/specifications/>

RISC-V – 32-bit instructions encoding

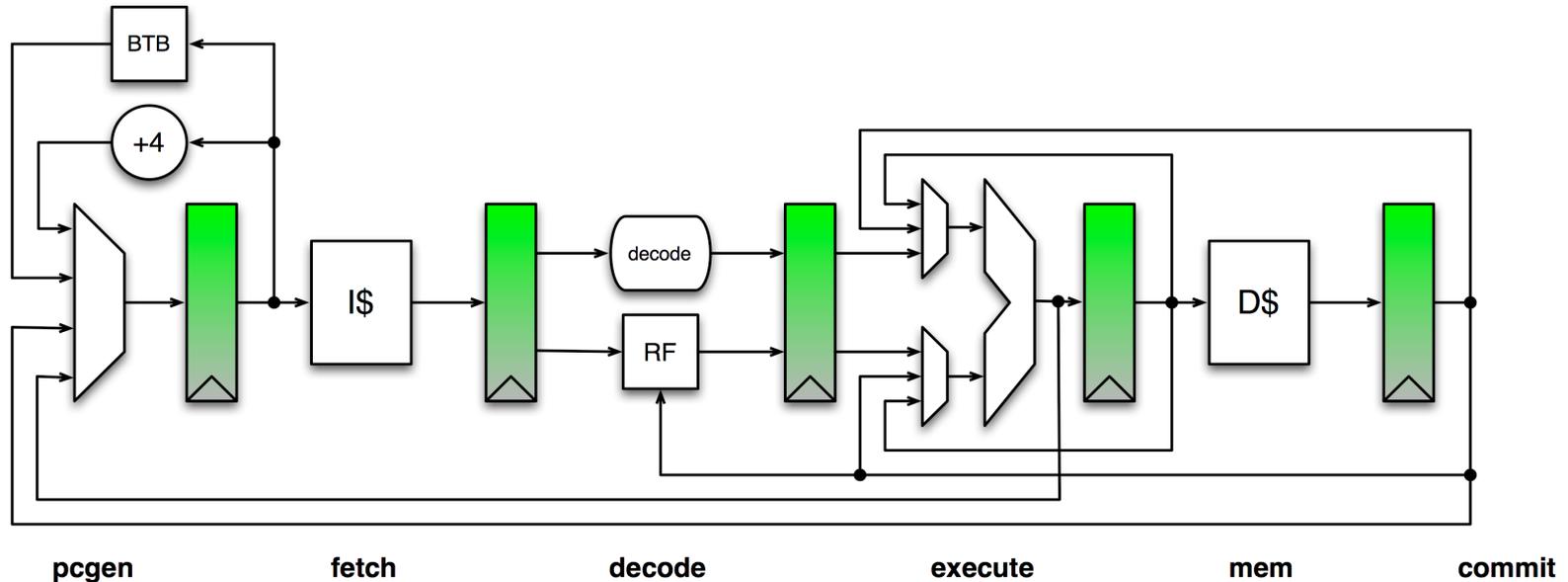


Source: <https://riscv.org/specifications/>

RISC-V – calling convention

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	–
x4	tp	Thread pointer	–
x5	t0	Temporary/alternate link register	Caller
x6–7	t1– 2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10 – 11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

RISC-V Rocket Core



Source: <http://www-inst.eecs.berkeley.edu/~cs250/fa13/handouts/lab2-riscv.pdf>

Implemented in chisel

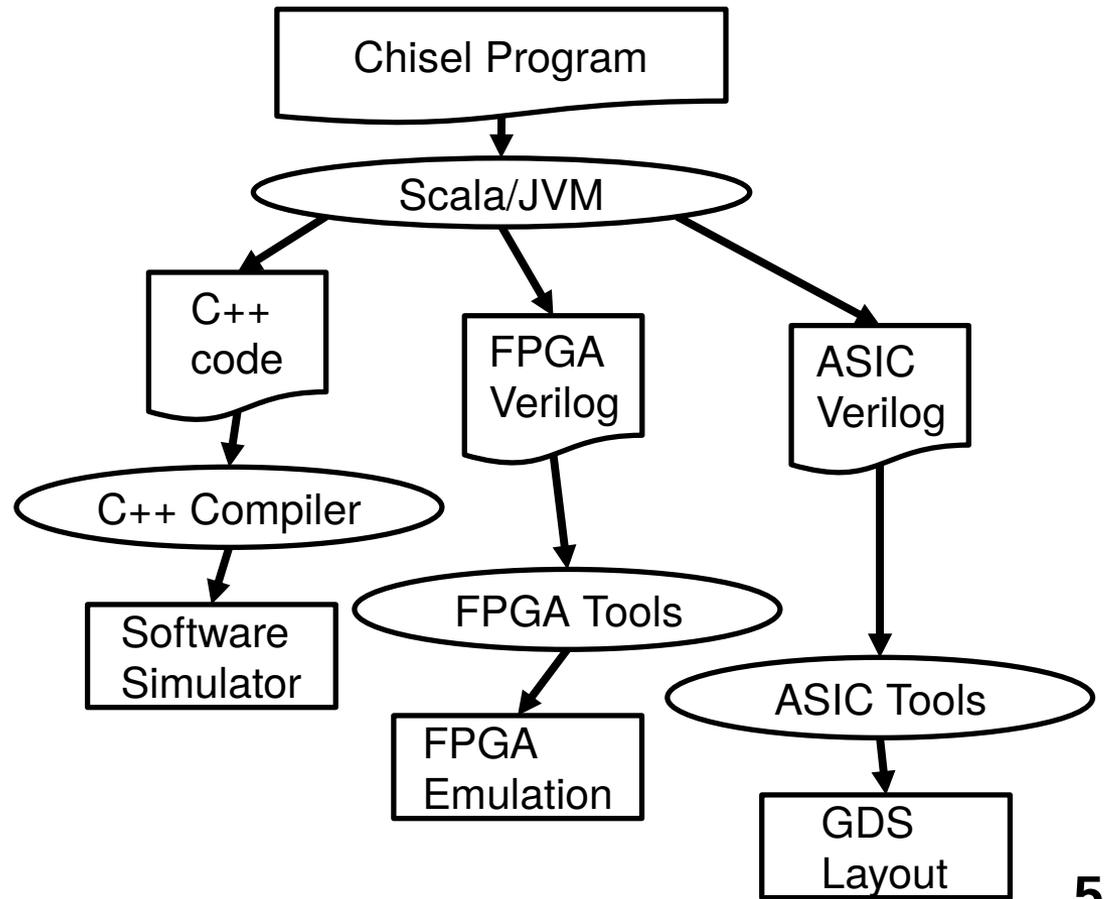
<https://github.com/freechipsproject/rocket-chip>

git clone git://github.com/freechipsproject/rocket-chip.git

branches boom, boom-devel, boom2 ...

Why Chisel?

- RTL generator written in Chisel
 - HDL embedded in Scala
- Full power of Scala for writing generators
 - object-oriented programming
 - functional programming



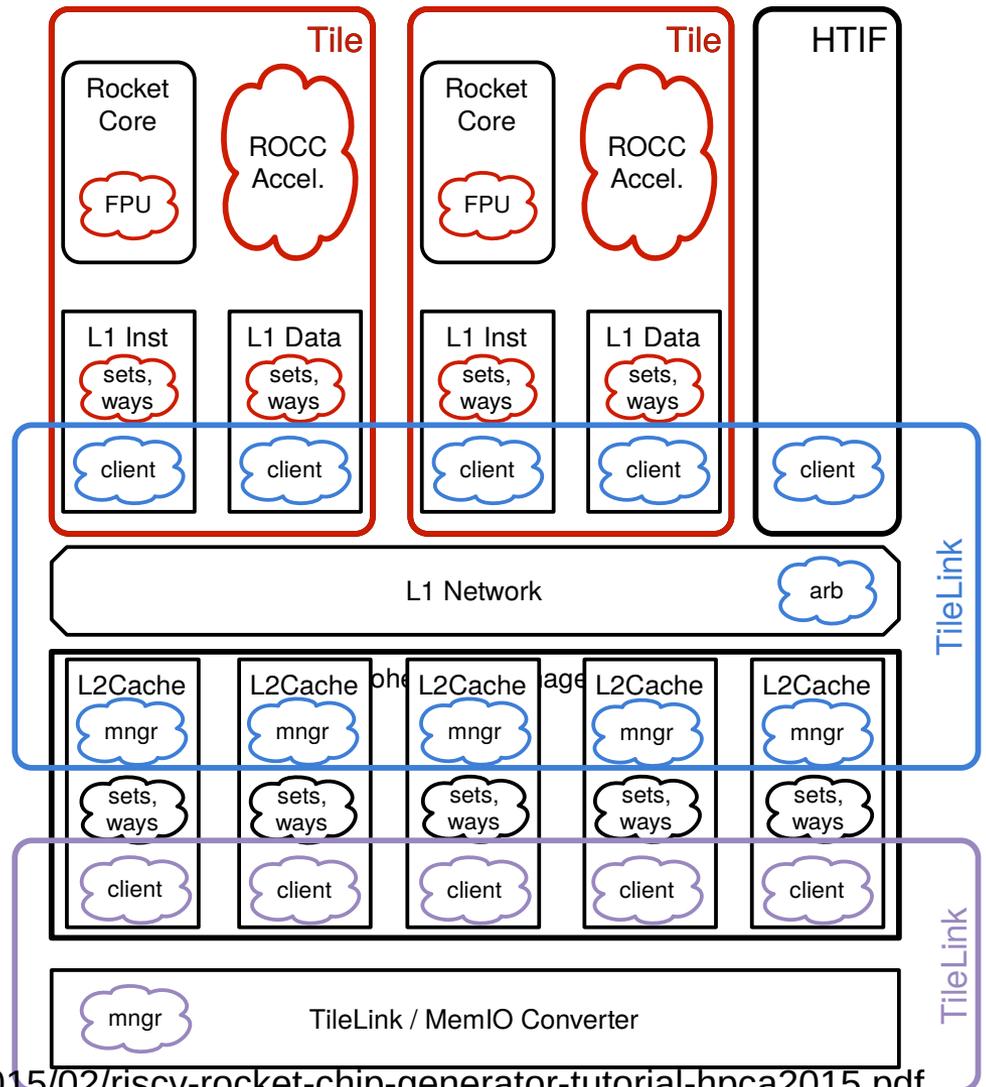
CHISEL

- Open-source hardware construction language
- UC Berkeley
- Supports advanced hardware design
- Highly parameterized generators
- Layered domain-specific hardware languages.
- Embedded in the Scala programming language
- Algebraic construction and wiring
- Hierarchical + object oriented + functional construction
- Highly parameterizable using metaprogramming in Scala
- Generates low-level Verilog designed to pass on to standard ASIC or FPGA tools
- Multiple clock domains

Source: <https://chisel.eecs.berkeley.edu/>

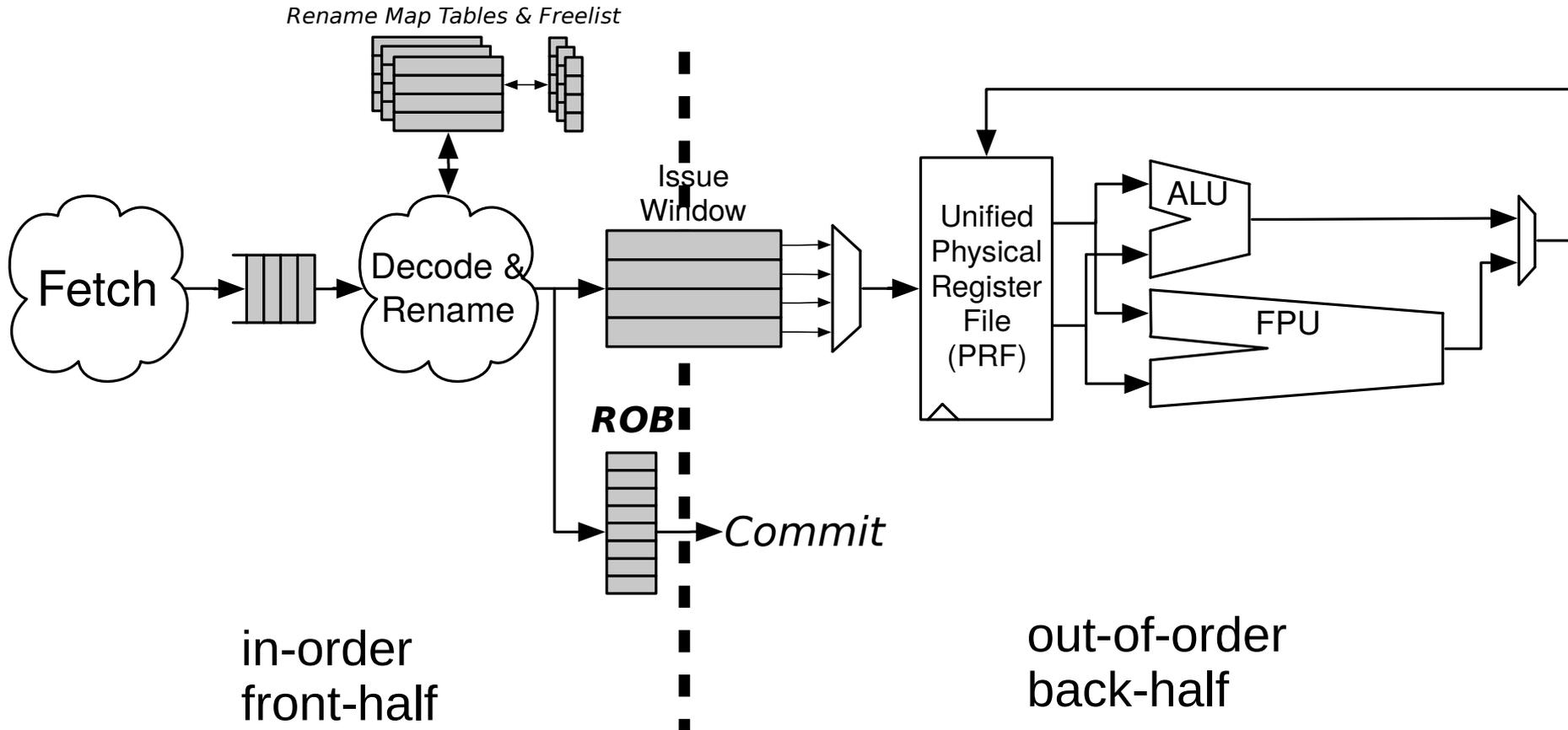
“Rocket Chip” SoC Generator

- Generates n Tiles
 - (Rocket) Core
 - RoCC Accelerator
 - L1 I\$
 - L1 D\$
- Generates HTIF (The host-target interface)
 - Host DMA Engine
- Generates Uncore
 - L1 Crossbar
 - Coherence Manager
 - Exports
 - MemIO
 - Interface



Source: <https://riscv.org/wp-content/uploads/2015/02/riscv-rocket-chip-generator-tutorial-hpca2015.pdf>

BOOM Superscalar RISC-V into Rocket Chip

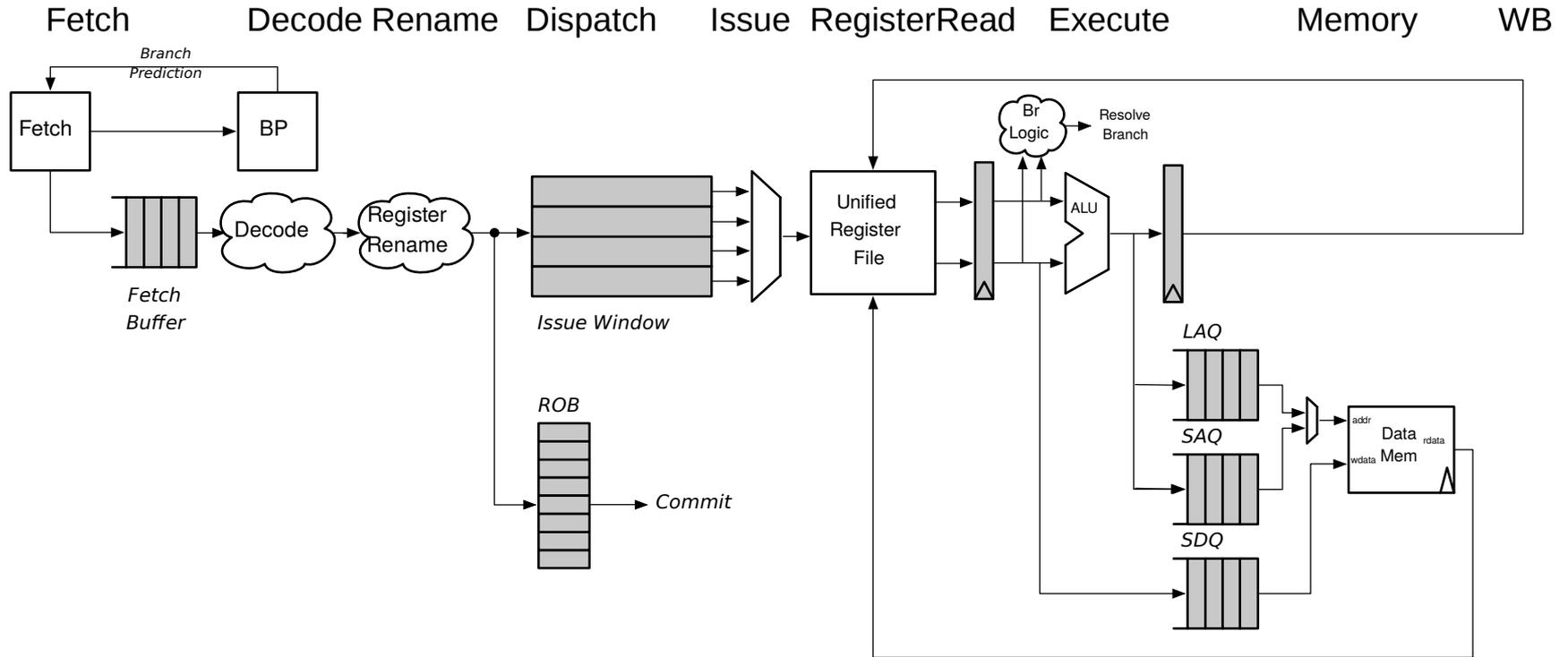


Main developer: Christopher Celio

9k source lines + 11k from Rocket

Source: <https://riscv.org/wp-content/uploads/2016/01/Wed1345-RISCV-Workshop-3-BOOM.pdf>

BOOM Stages

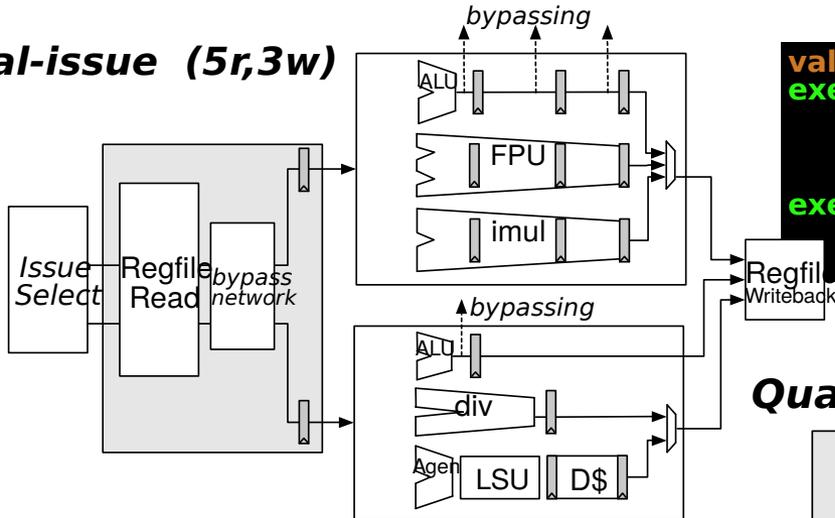


Source: <https://github.com/ccelio/riscv-boom-doc>

BOOM Parametrized Superscalar

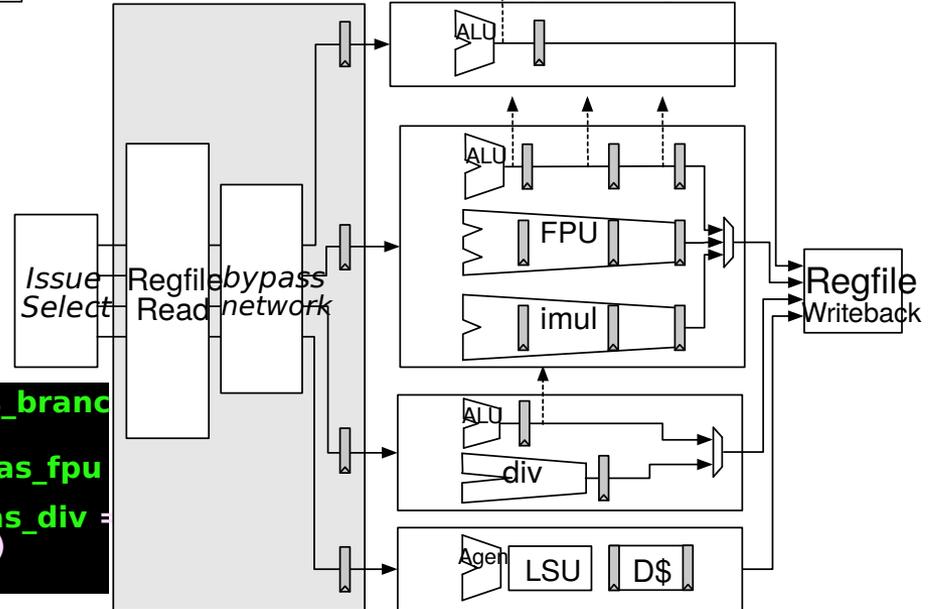


dual-issue (5r,3w)



```
val exe_units = ArrayBuffer[ExecutionUnit]()
exe_units += Module(new ALUExeUnit(
    is_branch_unit = true
))
exe_units += Module(new ALUMemExeUnit(
    fp_mem_support
))
```

Quad-issue (9r,4w)

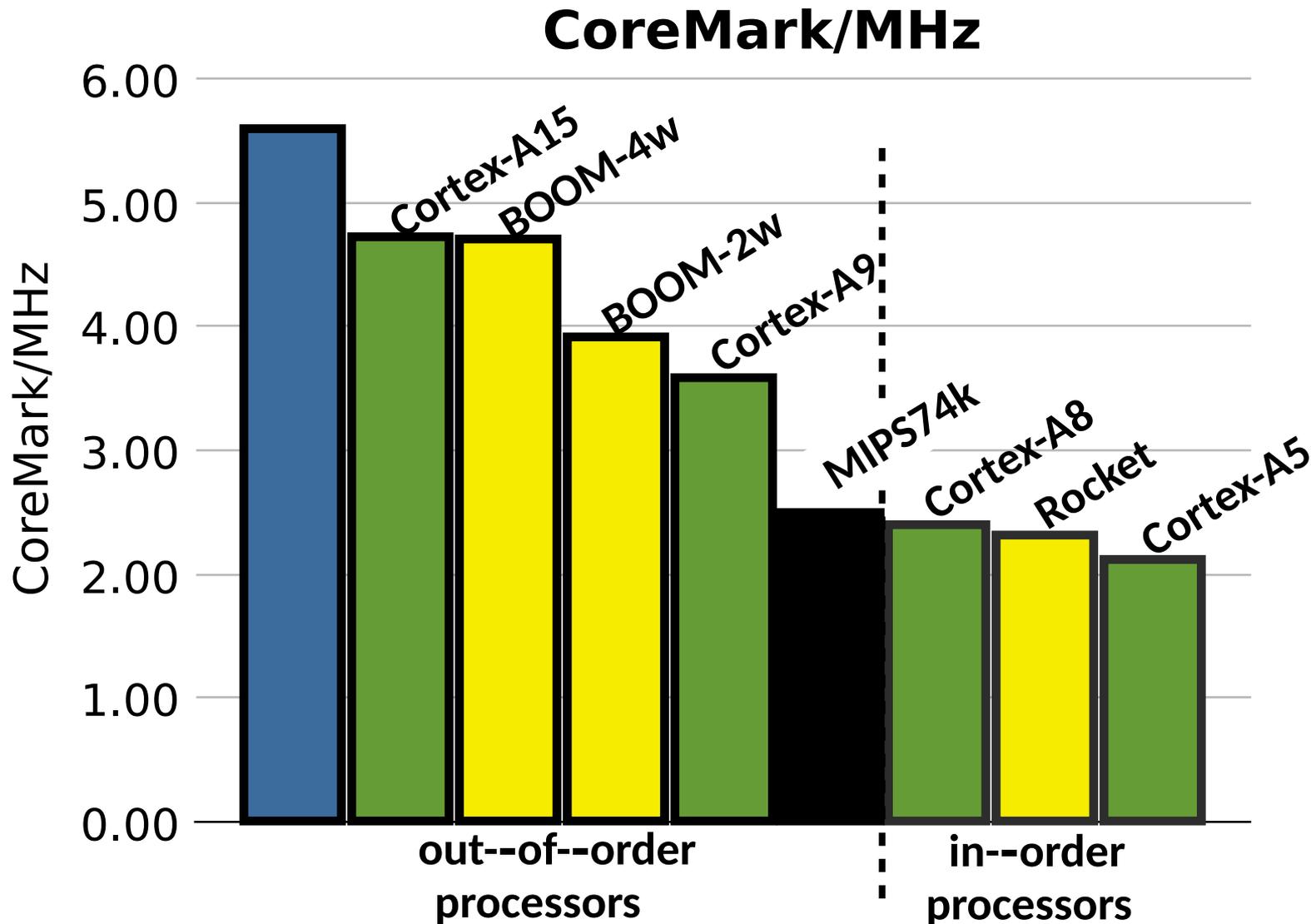


OR

```
exe_units += Module(new ALUExeUnit(is_branch
    true))
exe_units += Module(new ALUExeUnit(has_fpu
    true))
exe_units += Module(new ALUExeUnit(has_div
    true))
exe_units += Module(new MemExeUnit())
```

Source: <https://riscv.org/wp-content/uploads/2016/01/Wed1345-RISCV-Workshop-3-BOOM.pdf>

BOOM – Expected CoreMark Results

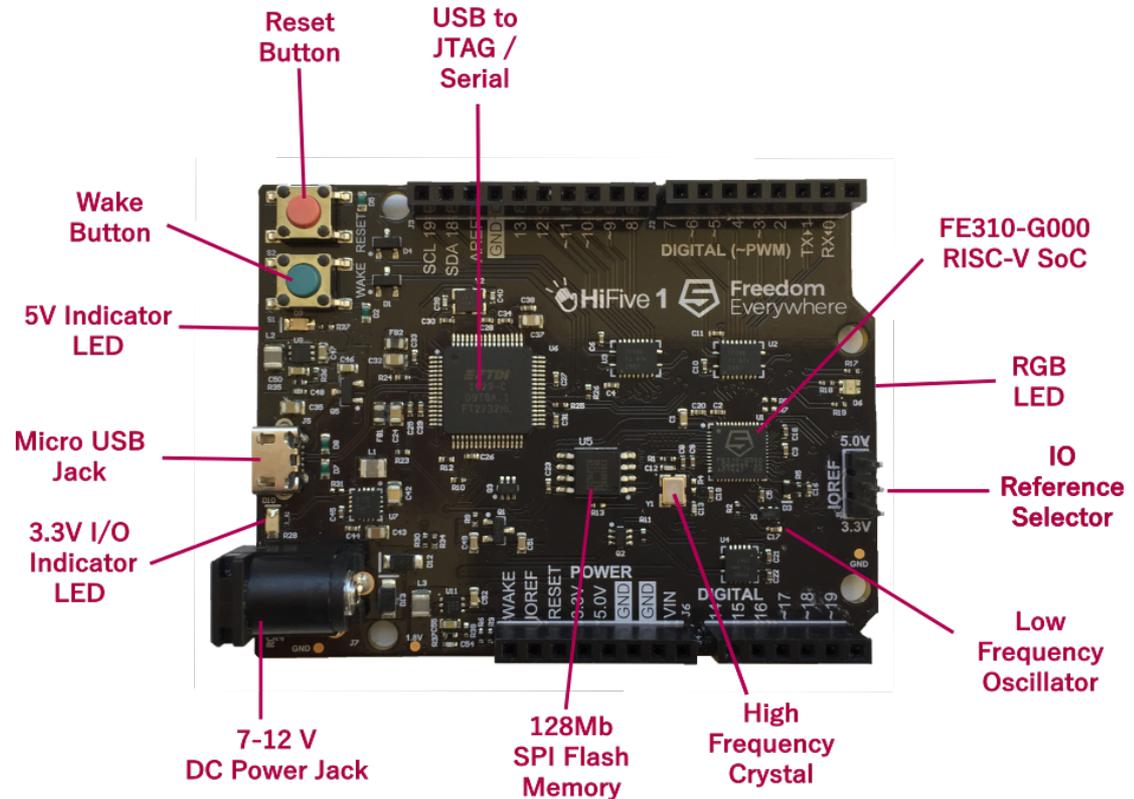


RISC-V Tools and OS Status

- Linux kernel mainline 4.13
- GCC 7.1
- Binutils
- OpenOCD
- Glibc
- LLVM

RISC-V – HiFive1 MCU from SiFive

- SiFive Freedom E310
- SiFive E31 RISC-V Core
- 32-bit RV32IMAC
- Speed: 320+ MHz
- 1.61 DMIPs/MHz, 2.73 Coremark/MHz
- 16 KB Instruction Cache 16 KB Data Scratchpad
- Hardware Multiply/Divide, Debug Module, Flexible Clock Generation with on-chip oscillators and PLLs

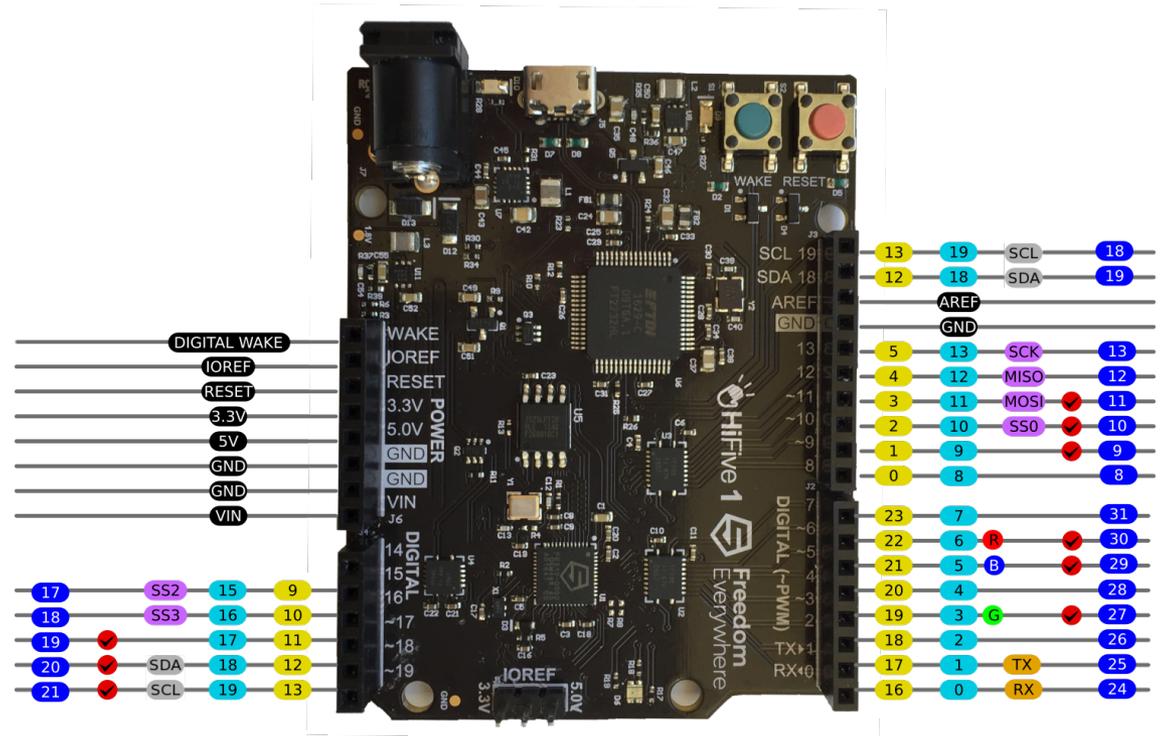


Source: <https://www.sifive.com/products/hifive1/>

RISC-V – HiFive1 from SiFive Pinout

HiFive1 Pinout

- Arduino pinout
- OpenOCD
- RISC-V GNU Toolchain



GPIO PIN POWER SERIAL SPI PWM INTERRUPT SW I2C



2014 by Bouni
2016 By SiFive, Inc
Photo by SiFive, Inc

<https://github.com/sifive/freedom-e-sdk>