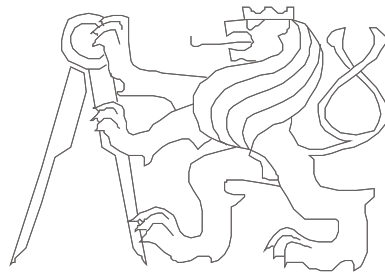


Pokročilé architektury počítačů

Příklady RISC architektur ARM, AArch64 a RISC-V

Pavel Píša, Michal Štepanovský



České vysoké učení technické, Fakulta elektrotechnická

Architektura ARM - registry

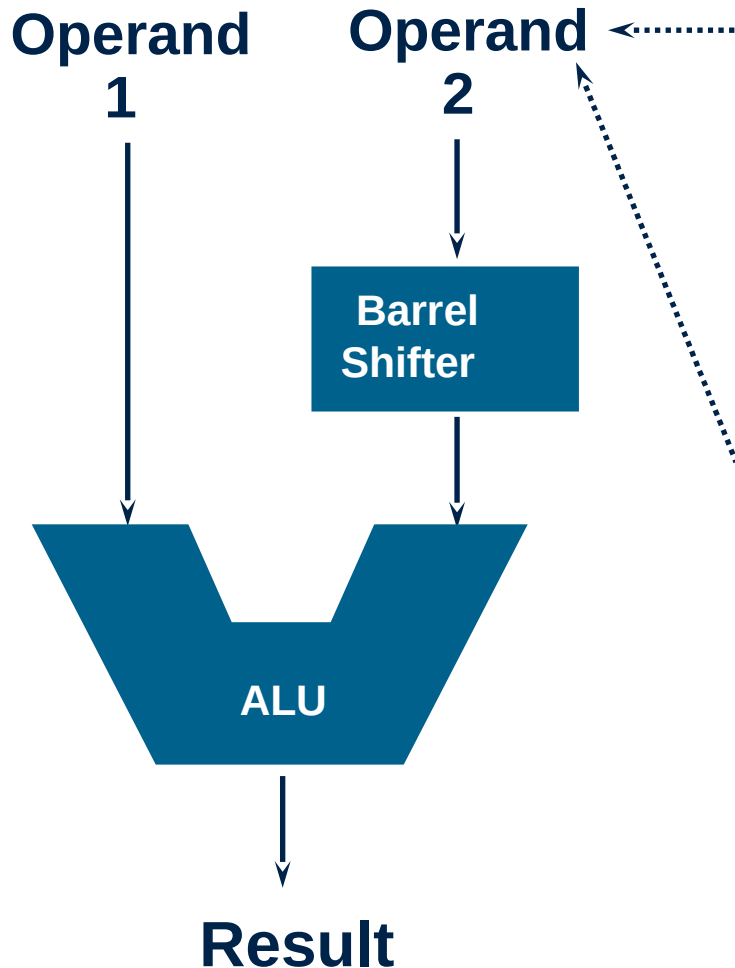
Current Visible Registers

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)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
	spsr	spsr	spsr	spsr

Architektura ARM – ALU a operandy



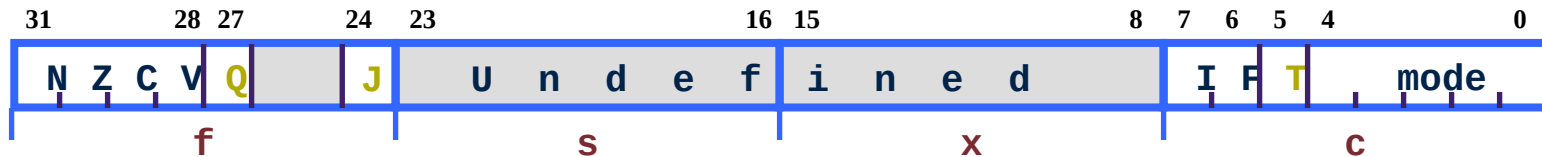
Register, optionally with shift operation

- 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

Architektura ARM – stavové slovo



■ 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

Architektura ARM – operační režimy

- 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

- K volání používá LR, ruší banky registrů, pro výjimky ELR
- PC vyčleněn zvlášť (není již běžný registr)
- 31 64-bitových registrů R0 až R30 (R30 = X30 \cong LR)
 - V kódu Wn ($W0$) pro 32-bit přístup, Xn ($X0$) pro 64-bit
 - Reg. code 31 stejně jako MIPS 0, v kódu WZR/XZR
 - V některých instrukcích má reg. code 31 význam WSP, SP
- Přímý operand 12-bit s volitelným LS 12 pro aritmetické operace, opakující se bitové masky pro logické
- 32-bit operace ignorují bity 32–63 na vstupu a nulují na výstupu

AArch64 – Větvení a podmíněné operace

- Zrušená možnost podmínění každé instrukce i Thumb IT
- Zůstává příznakový registr, přidané CBNZ, CBZ, TBNZ, TBZ
- Pouze několik podmíněných instrukcí
 - součet a rozdíl s přenosem, select (move C?A:B)
 - nastavení 0 a 1 (nebo -1) podle podmínky
 - podmíněná instrukce porovnání
- 32-bit a 64-bit násobení a dělení (3 registry), násobení se sčítáním $64 \times 64 + 64 \rightarrow 64$ (čtyři registry), bity 64 až 127 z výsledku násobení

AArch64 – Přístup do paměti

- Adresa 48+1 bit, znaménkové rozšíření na 64-bitů
- Přímě zakódovaný offset voliteně násoben šířkou přístupu
- Když je použitý registr jako index, může být násobený šířkou přístupu a také lze omezit na 32-bitů
- PC relative $\pm 4\text{GB}$ lze zakódovat do 2 instrukcí
- Pouze načtení dvou nezávislých registrů LDP a STP (zrušeno LDM, STM), přidané LDNP, STNP
- Podporuje nezarovnané přístupy
- LDX/STX(RBHP) pro exkluzivní přístup 1,2,4,8 a 16 bytů

AArch64 – Adresní režimy

- 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

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.

- PC-relative (literal) load

label

RISC-V – znovu zjednodušit/optimalizovat RISC

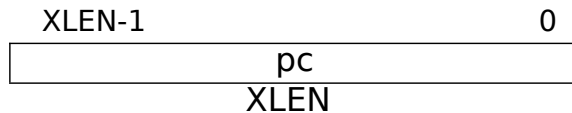
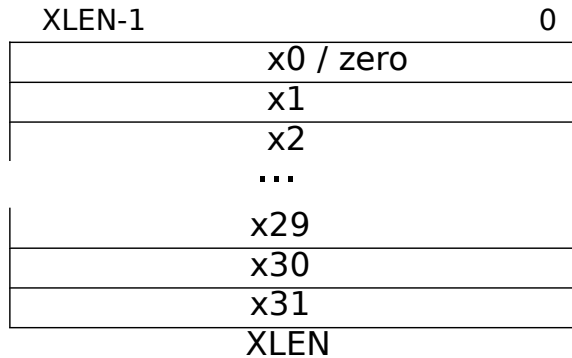
- Prof. Patterson, Berkeley RISC 1984 → počátek RISC éry, vyvinul se ve **SPARC** (Hennessy **MIPS**, Stanford University)
- Komericializací o rozvojem se architektury staly opět příliš složité, zároveň současné licence a patenty znemožňují původním tvůrcům vlastní reálnou implementaci ve formě křemíku pro výuku a výzkum
- MIPS je základem většiny základních kurzů a často je i implementace podobného procesoru součástí pokročilejších kurzů (B4M35PAP)
- Krste Asanovic a další studenti Dr. Pattersona proto začali vyvíjet vlastní novou architekturu (počátek 2010)
- BSD Licence, proto, aby byla přístupná i v budoucnu
- Podpora GCC, binutils., Linux, QEMU, atd.
- Proti SPARC jednodušší, více podobná MIPS ale optimalizovaná na úrovni zátěže hradel (fanout) a délky kritických cest v budoucích návrzích
- Existuje již několik otevřených implementací **Rocket** (SiFive, BOOM), projekt **lowRISC** má za cíl výzkum v oblasti bezpečnosti, v ČR Codasip
- Již více než 10 implementací SoC na křemíku

RISC-V – specifikace architektury

- Specifikace ISA dostupná na <http://riscv.org/>
 - The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0
Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic
 - Ne jen popis architektury, ale i důvodů proč bylo dané řešení vybrané a jaké problémy/cenu obnáší alternativní řešení
- klasický návrh 32 celočíselných registrů, jeden nula (zero), operandy `regsrc1`, `regsrc2`, `regdest`, zajímavost striktní dodržení i pro `SaveWord`, důsledkem nespojitost přímých/immediate operandů, PC mimo registry, PC-relativní adresování
- Varianty s 32, 64 a 128-bitovými registry a adresací
- Vysoká hustota kódu (plánovaná 16-bit varianta kódování instrukcí)
- V kódování systematicky vyhrazen prostor pro plovoucí řádovou čárku (single, double, quad) a multimediální SIMD instrukce atd.

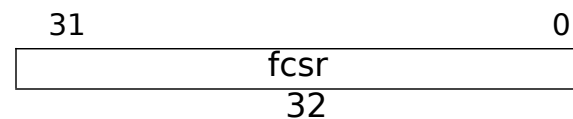
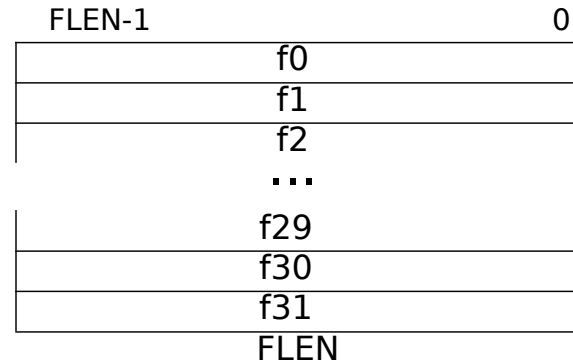
RISC-V – registry

Integer registers



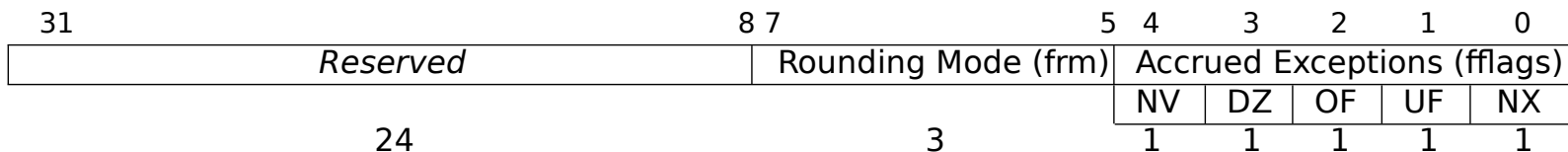
Variant	XLEN
RV32	32
RV64	64
RV128	128

Floating point registers



Variant	FLEN
F	32
D	64
Q	128

Floating-point control and status register



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

RISC-V – kódování délky instrukcí

xxxxxxxxxxxxxxxxaa 16-bit (aa ≠ 11)

xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxbbb11 32-bit (bbb ≠ 111)

· · ·xxxx xxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxx011111 48-bit

· · ·xxxx xxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxx01111111 64-bit

· · ·xxxx xxxxxxxxxxxxxxxxxxxxxx xnnnxxxxx11111111 (80+16*nnn)-bit, nnn ≠ 111

· · ·xxxx xxxxxxxxxxxxxxxxxxxxxx x111xxxxx11111111 Reserved for ≥192-bits

Address:

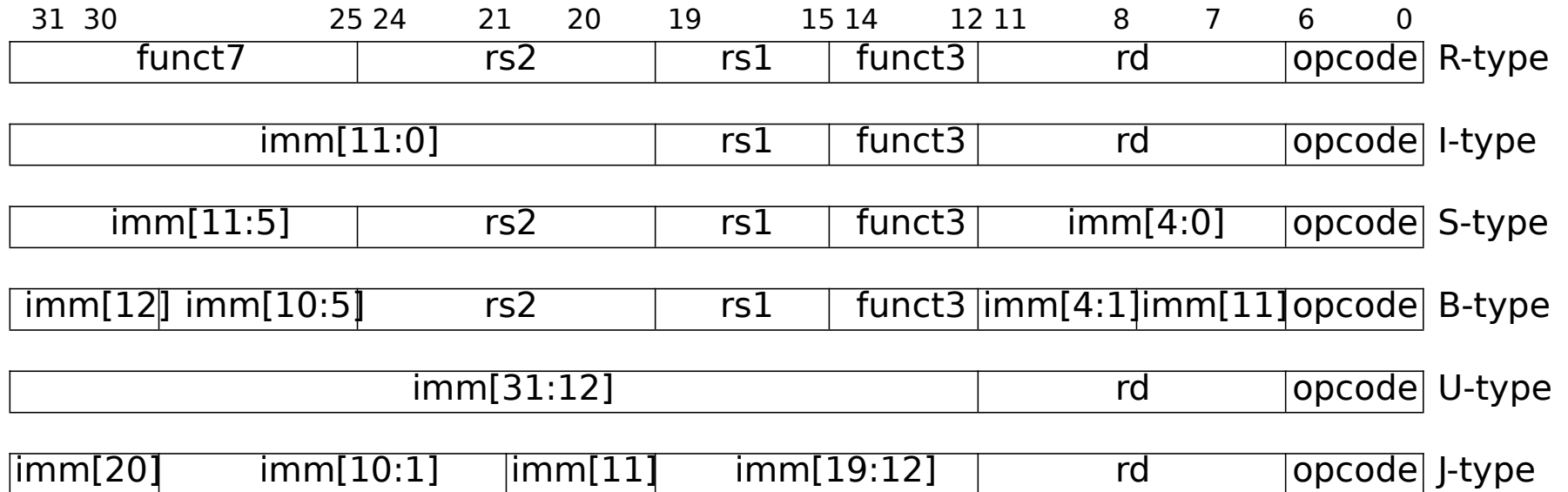
base+4

base+2

base

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

RISC-V – kódování 32-bit instrukcí

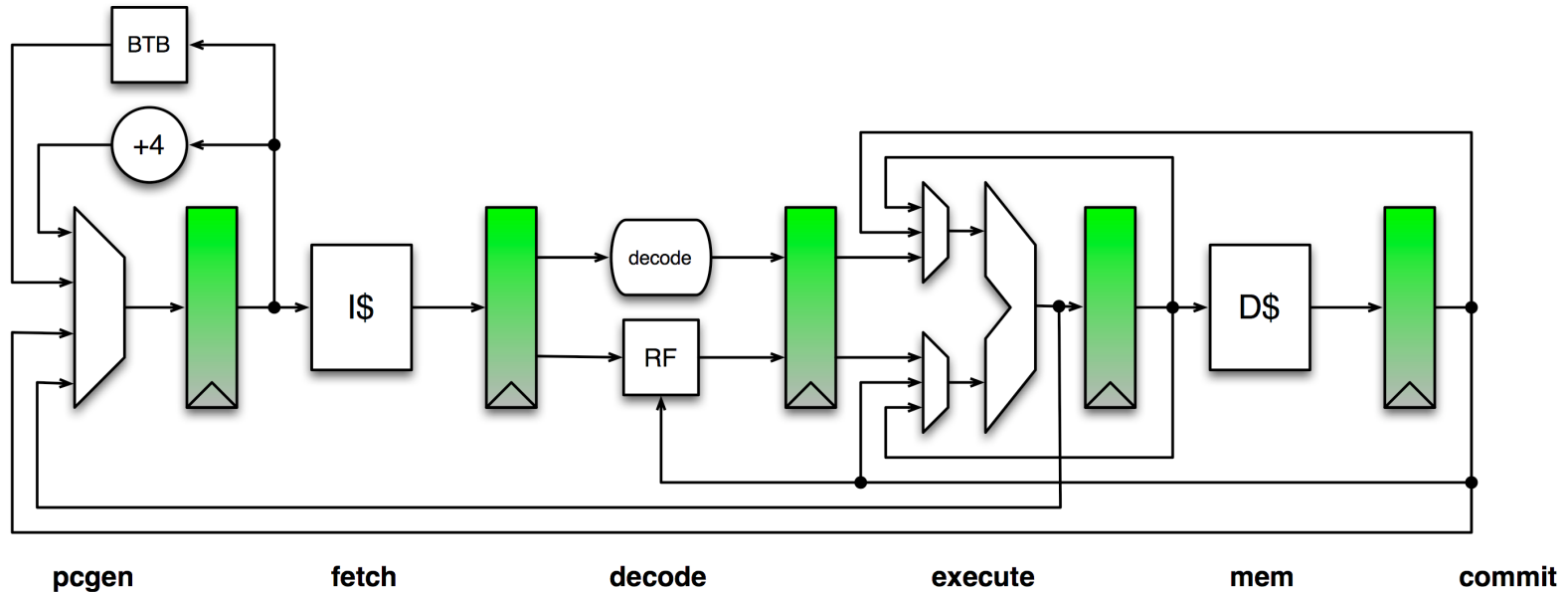


Zdroj: <https://riscv.org/specifications/>

RISC-V – volací konvence

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>

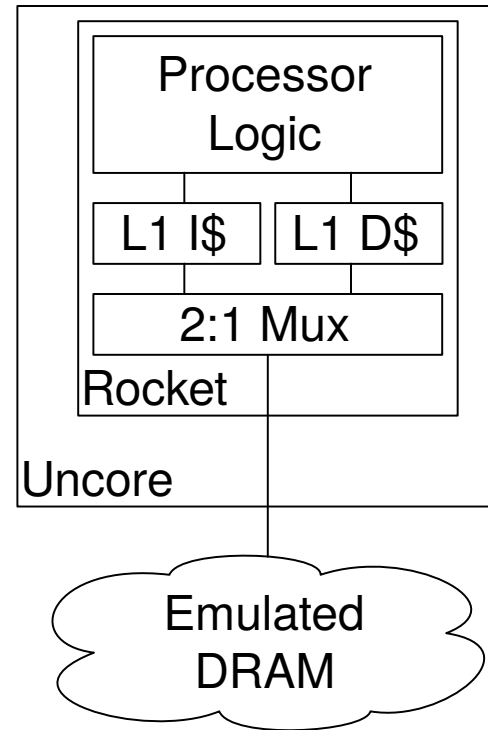
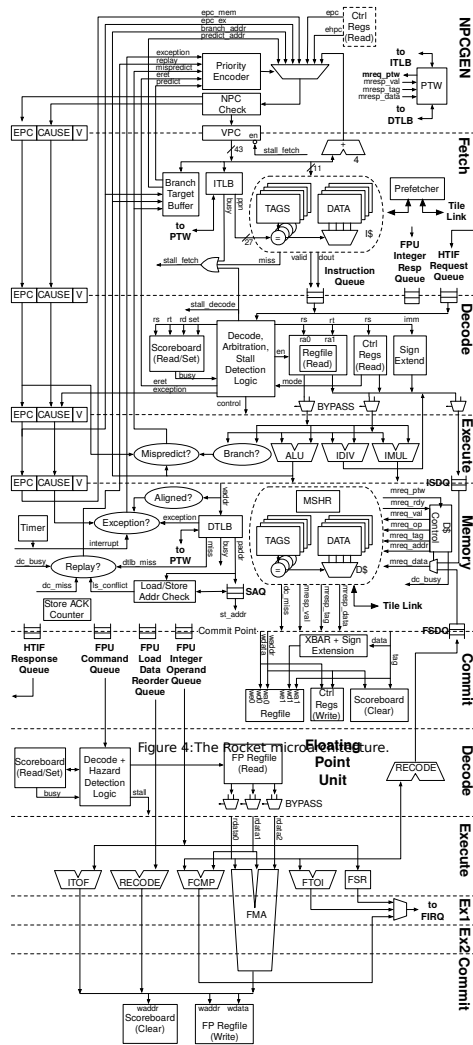
Implemented in chisel

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

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

branches boom, boom-devel, boom2 ...

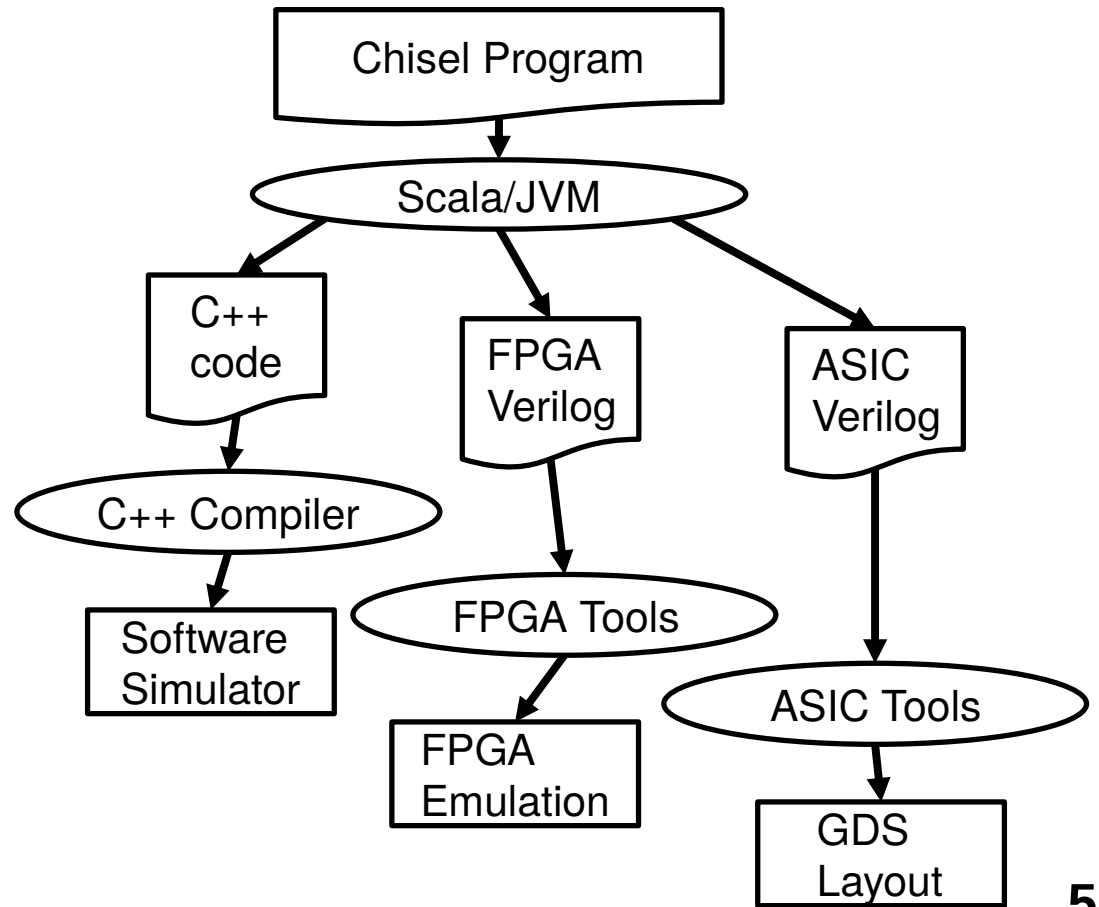
CS250 Rocket Pipeline and Memory Hierarchy



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

Why Chisel?

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



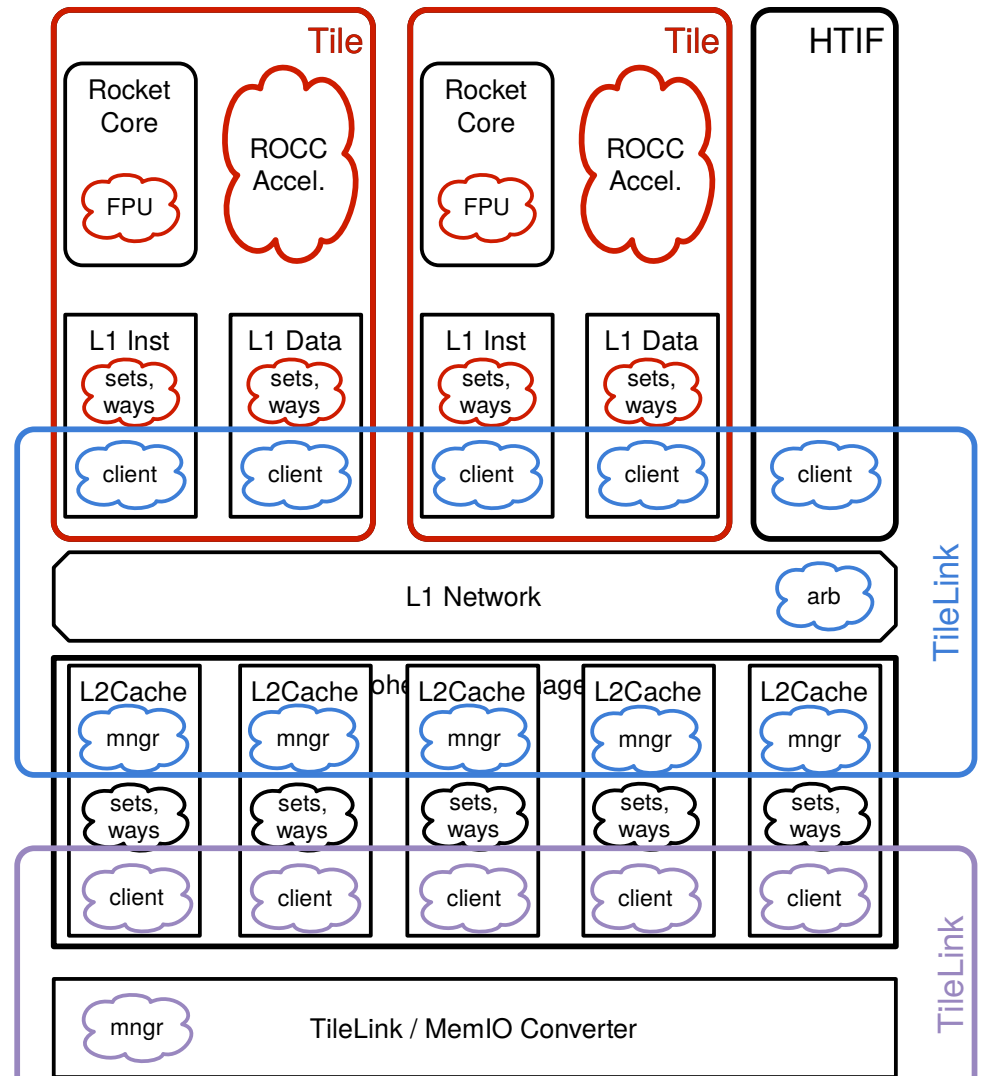
CHISEL

- 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/>

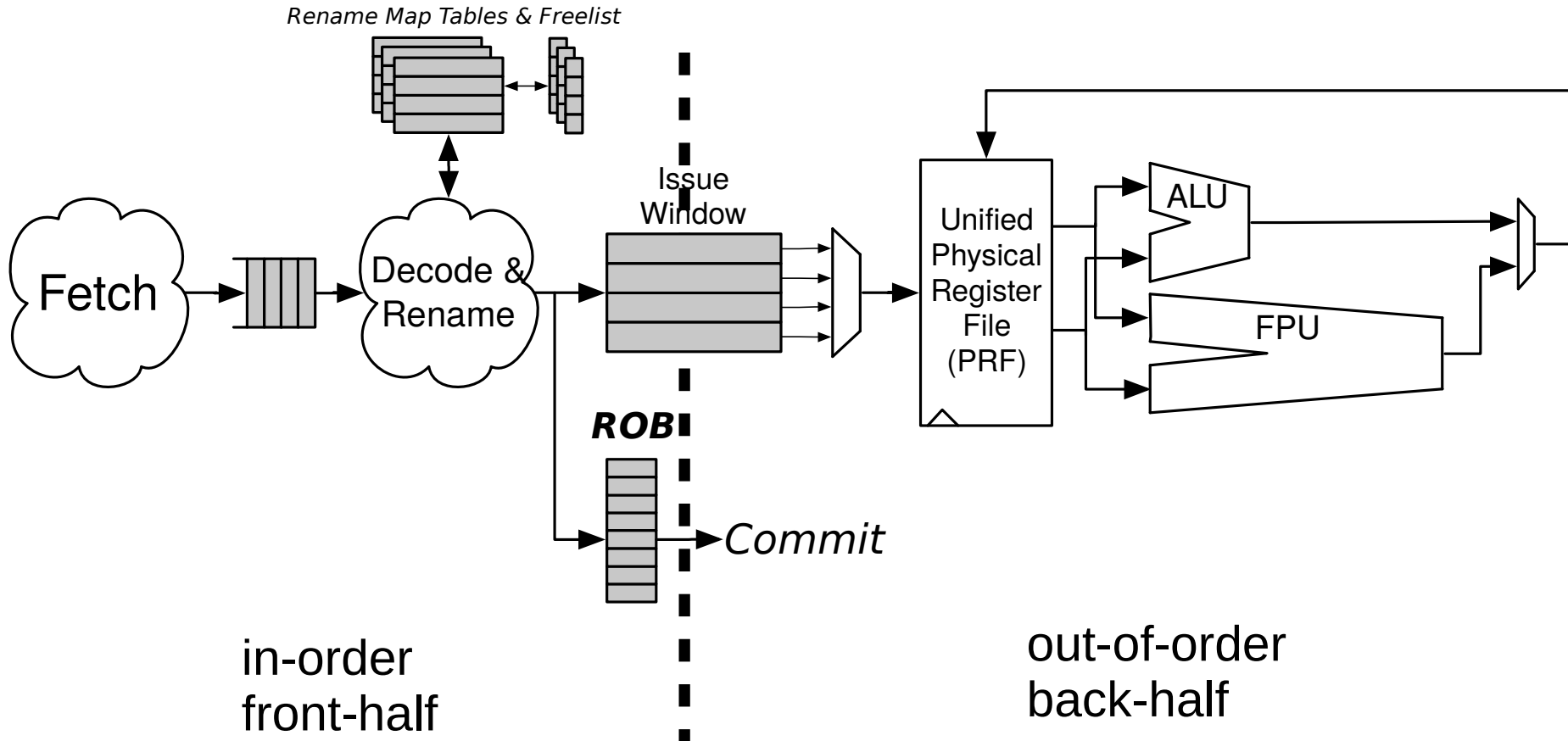
“Rocket Chip” SoC Generator

- 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

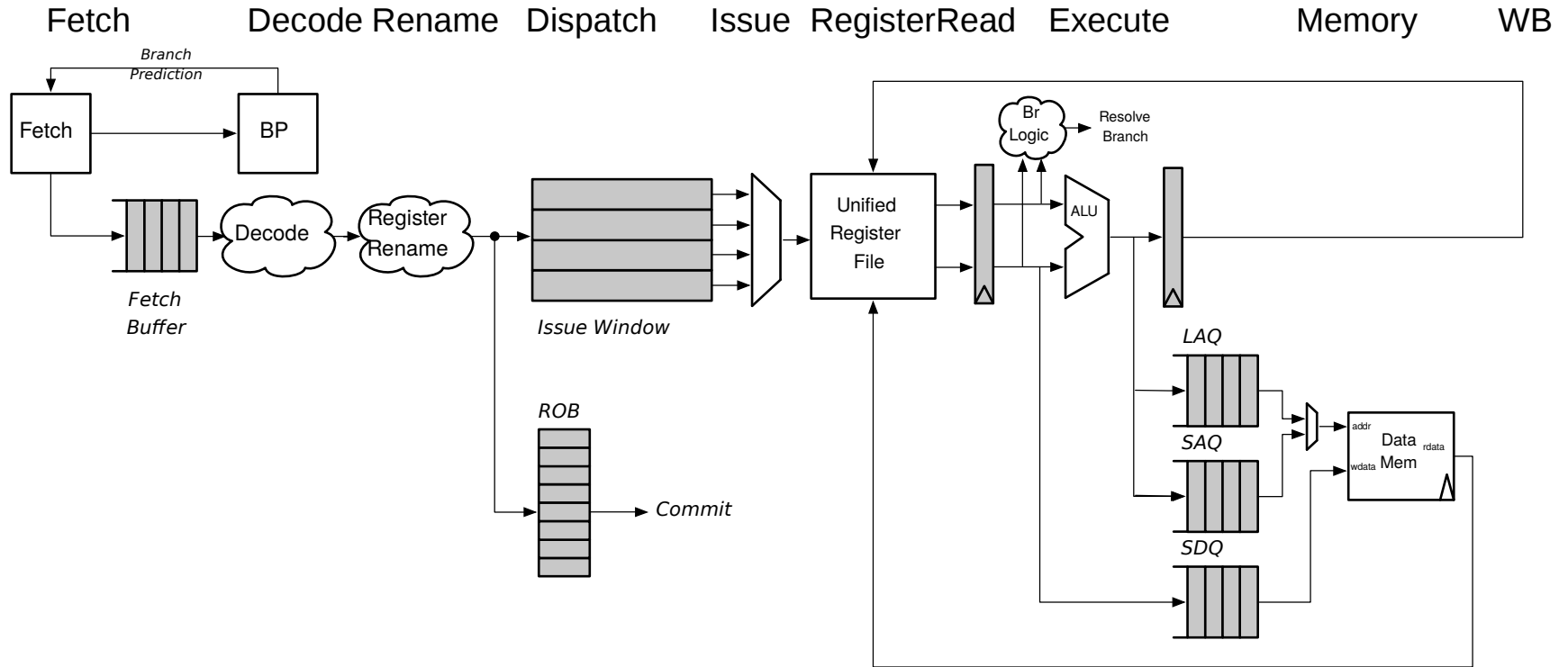


Main developer: Christopher Celio

9k source lines + 11k from Rocket

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

BOOM Stages

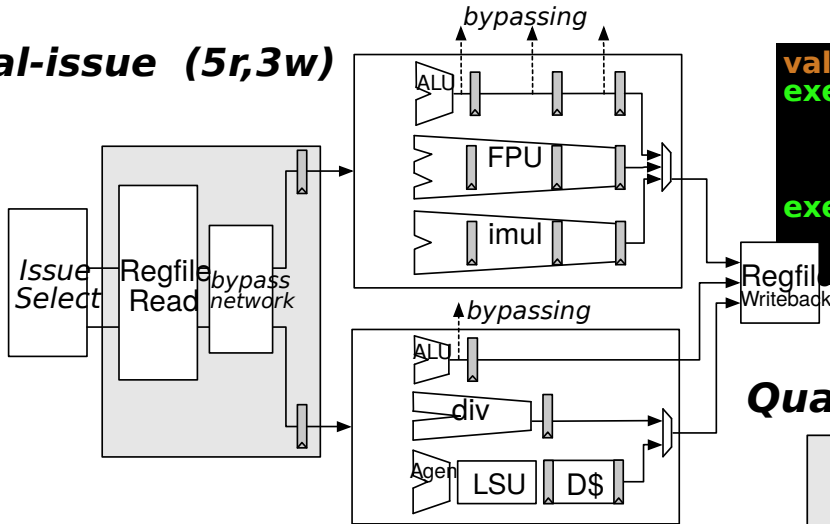


Source: <https://github.com/ccelio/riscv-boom-doc>

BOOM Parametrized Superscalar

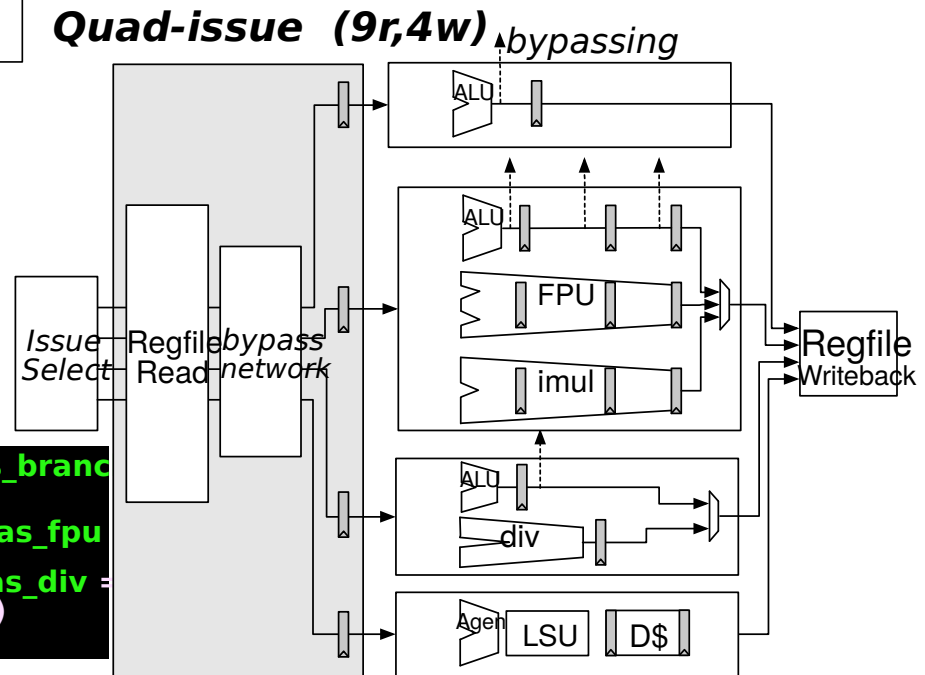


dual-issue (5r,3w)



```
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)

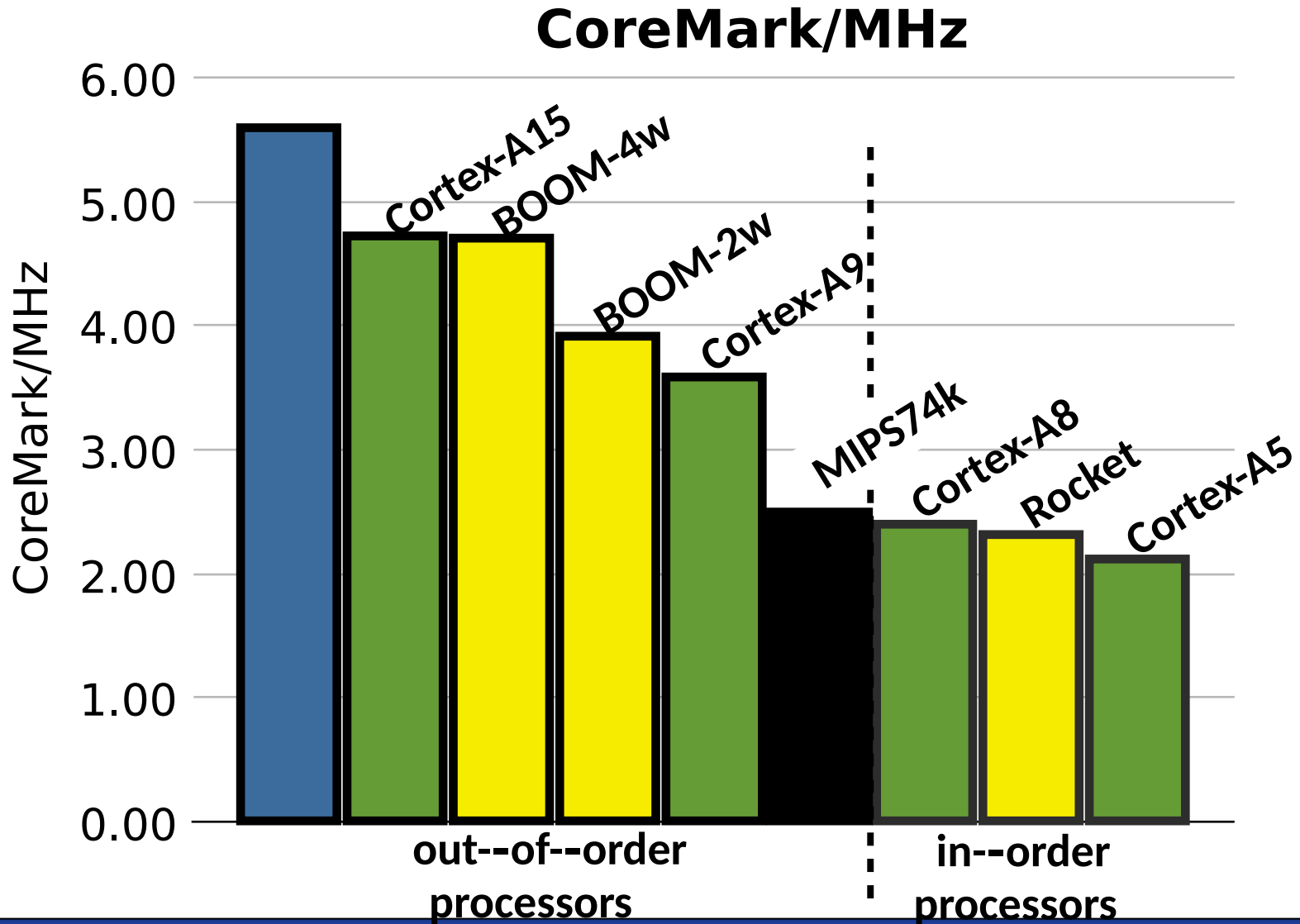


OR

```
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

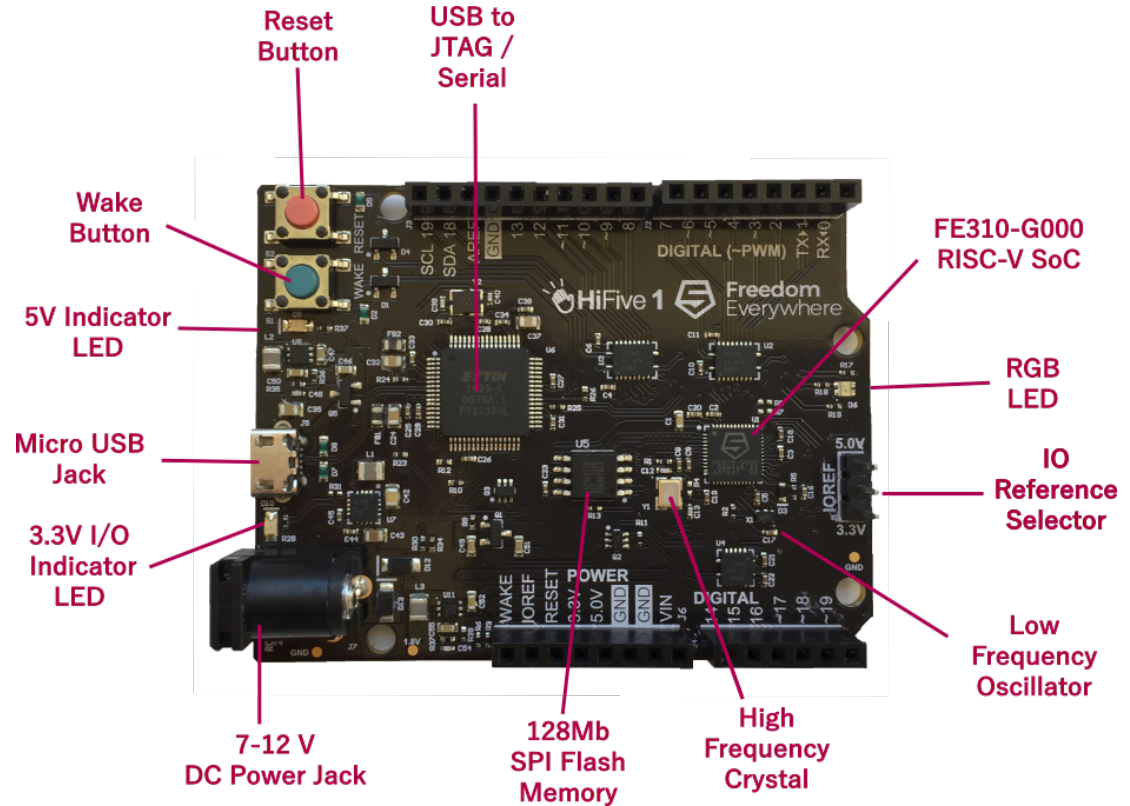


RISC-V Tools and OS Status

- 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

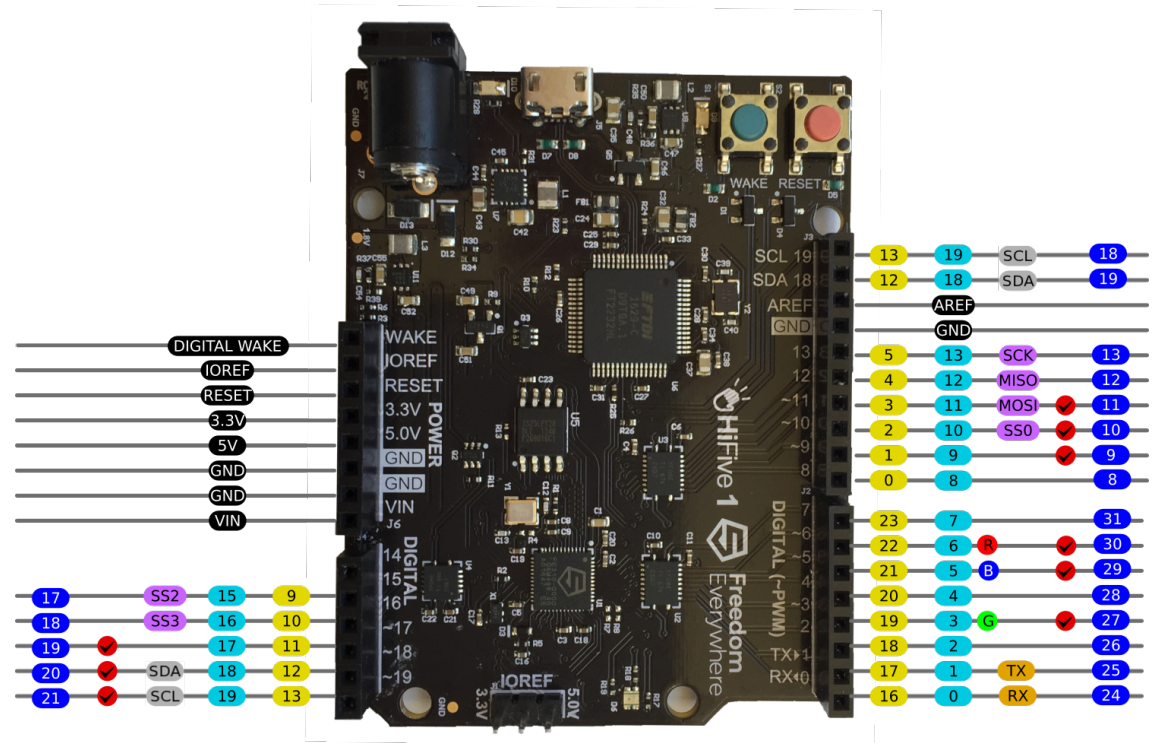


Source: <https://www.sifive.com/products/hifive1/>

RISC-V – HiFive1 from SiFive Pinout

- Arduino pinout
- OpenOCD
- RISC-V GNU Toolchain

HiFive1 Pinout



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>