Developments in LLVM-based toolchains and tooling for RISC-V

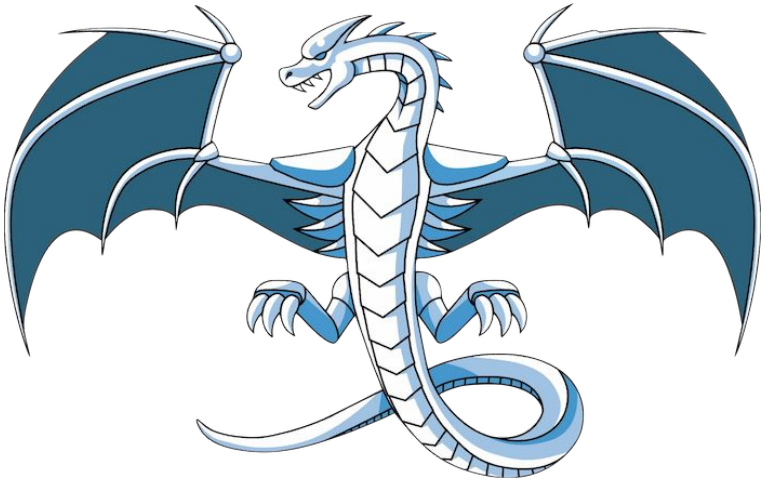
Alex Bradbury asb@igalia.com

RISC-V Summit Europe, 2023-06-08

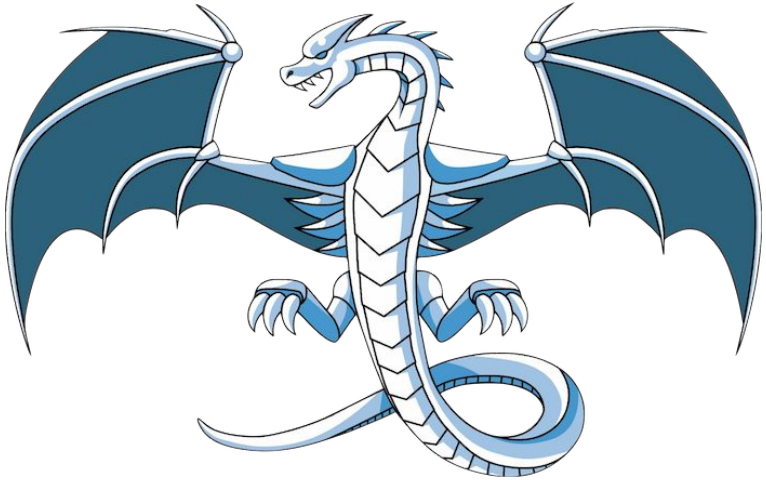
igalia

What is LLVM?

```
int add(int a, int b) {
  return a+b;
}
```

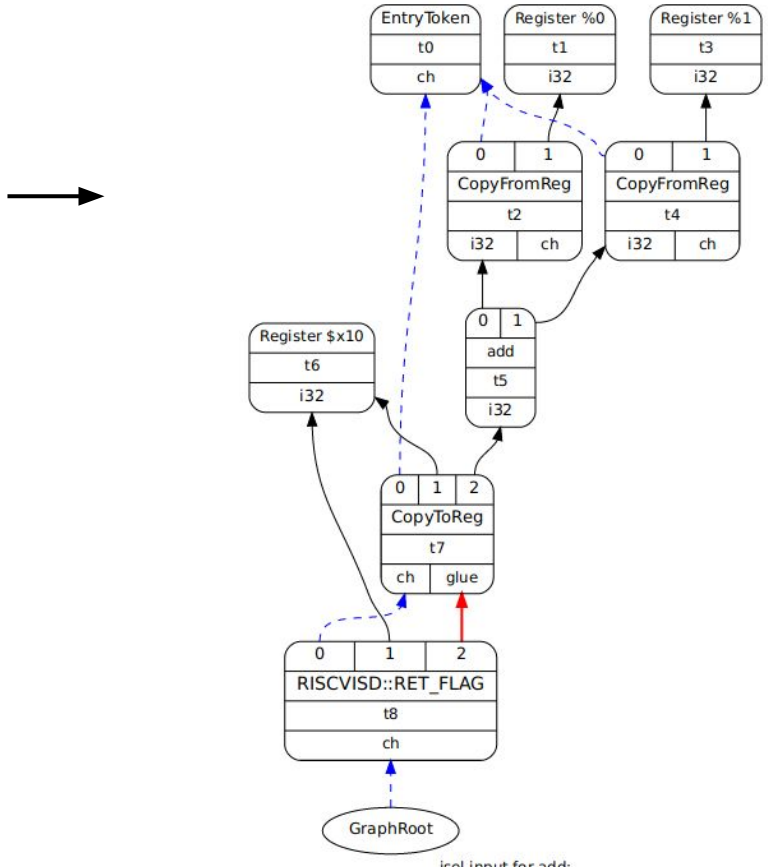


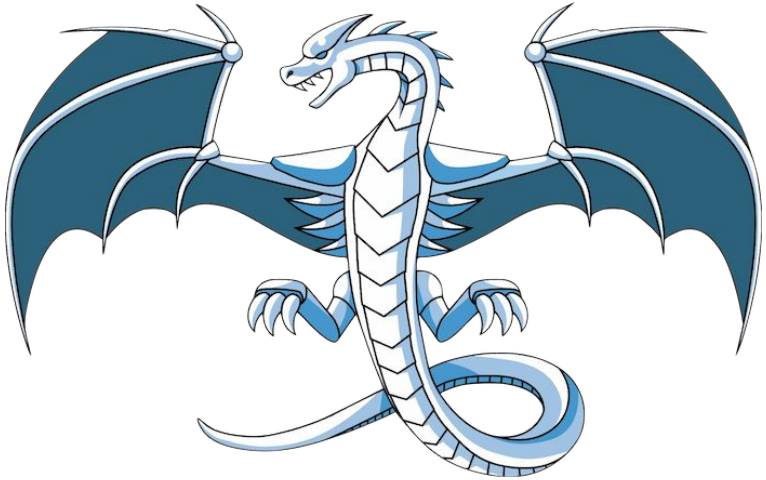
What is LLVM?



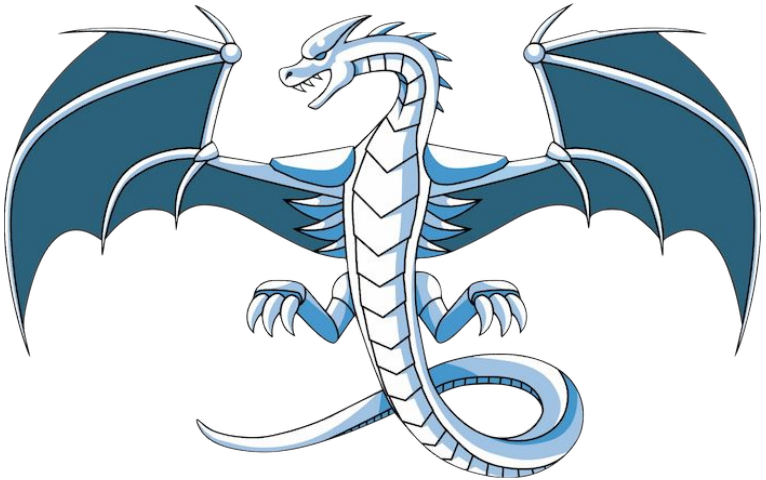
```
define i32 @add(i32 %a, i32 %b) {
  %1 = add i32 %a, %b
  ret i32 %1
}
```

What is LLVM?


EntryToken
t0
ch
Register %0
t1
i32
Register %1
t3
i32
0
1
CopyFromReg
t2
i32
ch
0
1
CopyFromReg
t4
i32
ch
Register $x10
t6
i32
0
1
add
t5
i32
0
1
2
CopyToReg
t7
ch
glue
0
1
2
RISCVISD::RET_FLAG
t8
ch
GraphRoot



What is LLVM?



```
add.o:    file format ELF32-riscv

Disassembly of section .text:
0000000000000000 add:
       0:    33 05 b5 00     add    a0, a0, a1
       4:    67 80 00 00     ret
```

What is LLVM?

- A collection of modular compiler and toolchain technologies
- Modern C++ implementation
- Library-based design
- Permissively licensed
- C/C++ toolchain (Clang) and equivalents to various binutils tools
- Primary backend for e.g. Rust
- Used by many downstream vendor toolchains

What is LLVM: Beyond the compiler you know

- MLIR
- Flang
- libopenmp
- libcxx
- lld
- lldb
- libc
- BOLT
- ...

RISC-V LLVM current status

- A ‘default’ (rather than experimental) backend since LLVM/Clang 9.0 (Sep 2019), first patches merged October 2017.
- Current extension support status: https://llvm.org/docs/RISCVUsage.html
- LLD (with linker relaxation) now also stable.

RISC-V LLVM current status: vs RVA22U64 profile

	Assembler	Codegen
rv64imafdc	✔	✔
zicsr, zicntr, zihpm	✔	N/A
zicclsm	✘	✔(*)
zihintpause	✔	N/A
zba, zbb, zbs	✔	✔
zicbom, zicbop, zicboz	✔	N/A
zfhmin, zfh	✔	✔
zkt	✔	N/A
V	✔	✔
zkn, zks	✔	✔

RISC-V LLVM current status: vs draft RVA23U64 profile

	Assembler	Codegen
zicond (experimental)	✔	✔(*)
zcb	✔	✔
zfa (experimental)	✔	✔
zbc	✔	✔
zvfh (experimental)	✔	✔(*)
zfbfmin (experimental)	✔	WIP
zvfbfmin, zvfbfwma (experimental)	✔	WIP
zvkng, zvksg, zvbb, zvbc (experimental)	✔	✘

RISC-V LLVM: A success story for cross-community upstream collaboration

(Partial) RISC-V LLVM credits

(Contributions in forms of code, reviews, advice, etc)

Sameer Abu Asal, Alexey Bataev, Alexey Baturo, Alex Bradbury, Qihan Cai, Chandler Carruth, Leonard Chan, Ahmed Charles, Chih-Mao Chen, Piyou Chen, Shiva Chen, Kito Cheng, Vitaly Cheptsov, Nelson Chu, David Chisnall, Liao Chunyu, Jessica Clarke, Simon Cook, Fraser Cormack, David Craven, Nick Desaulniers, Conor Dooley, Sam Elliott, Hal Finkel, Eli Friedman, Mikhail Gadelha, Ondrej Glasnak, Eric Gouriou, Mandeep Singh Grang, Jianjian Guan, Jonas Hahnfeld, Ben Horgan, Mitchell Horne, Petr Hosek, ShihPo Hung, Roger Ferrer Ibanez, Ed Jones, Andrew Kelley, David Kipping, Paul Kirth, James Y Knight, Aditya Kumar, Yeting Kuo, Luke Lau, Jim Lin, Michael Maitland, David Majnemer, Luís Marques, Ed Maste, John McCall, Dylan McKay, Azharuddin Mohammed, Job Noorman, Tim Northover, Krzysztof Parzyszek, Ana Pazos, Wang Pengcheng, Jordy Portman, Nitin John Raj, Philip Reames, Lewis Revill, John Russo, Colin Schmidt, Ed Schouten, Andrews Schwab, Jun Sha, Ben Shi, Anton Sidorenko, Pavel Šnobl, Fangrui Song, Shao-Ce Sun, Sami Tolvanen, Philipp Tomsich, Manolis Tsamis, Rui Ueyama,, Hsiangkai Wang, Ulrich Weigand, Mario Werner, Jim Wilson, Brandon Wu, Xinlong Wu, Eugene Zalenko, Florian Zeitz, Leslie Zhai, Zhu Zijia, ...

and certainly more I missed (sorry!)

Lots of contributors over time, but a small core set of most active contributors - more contributions very welcome!

RISC-V LLVM stats

- About 4600 commits(*)
- About 56KLoC in llvm/lib/Target/RISCV
  - Many more lines in tests of course!

(*): `git rev-list --count HEAD -- llvm/lib/Target/RISCV llvm/test/CodeGen/RISCV/ llvm/test/MC/RISCV/ lld/ELF/Arch/RISCV.cpp clang/test/CodeGen/RISCV/`

How we collaborate in (RISC-V) LLVM

- Time-based releases
  - Prioritisation?
- RFCs
- Mailing list / discourse discussion
- ‘Code owners’ and pre-commit review
- Biweekly sync-up / coordination calls

How we collaborate in RISC-V LLVM: Related repos

riscv-non-isa/riscv-elf-psabi-doc

A RISC-V ELF psABI Document

Makefile ★ 490 136

riscv-non-isa/riscv-asm-manual

RISC-V Assembly Programmer's Manual

★ 1.2k 216

riscv-non-isa/riscv-toolchain-conventions

Documenting the expected behaviour and supported command-switches for GNU and LLVM based RISC-V toolchains

★ 98 21

riscv/riscv-isa-manual

RISC-V Instruction Set Manual

TeX ★ 2.7k 474

riscv/riscv-c-api-doc

Documentation of the RISC-V C API

★ 44 18

riscv-non-isa/rvv-intrinsic-doc

C ★ 190 69

Not-yet-ratified and vendor specific extensions

- Enable upstream collaboration on not-yet-ratified standards
  - Agreed policy on merging support behind 'experimental' flags (e.g. `-menable-experimental-extensions`) with explicit spec version
  - Usual code review standards apply
  - No backwards compatibility or support expectation for anything other than final ratified spec.
- Allow vendor extensions to be supported upstream, reducing need for fragmentation for vendor-specific toolchains.
  - e.g. XVentanaCondOps, Xsfvcp, XTHeadVDot (and many others)
  - Considerations for inclusion: complexity/ invasiveness, support story, user base, …

Compilation for RISC-V: Custom passes

- RISCVSExtWRemoval
- RISCVCodeGenPrepare
- RISCVExpandAtomicPseudoInsts
- RISCVInsertSETVLI
- RISCVRedundantCopyElimination
- RISCVMakeCompressible
- RISCVRVVInitUndef

What’s new (ish): Vector support

- Auto vectorisation with the loop vectorizer
  - Enabled upstream
  - Parallel downstream work with BSC (and others) on tail folding using LLVM vector predication intrinsics (setting VL rather than using masked loads and stores).
  - More tuning to be done
  - Has support for scalable vectors (and vector register grouping)
    - Can generate scalable strided loads and stores
    - Patches for scalable interleaved and deinterleaved ("segmented") loads and stores very close to landing

What’s new (ish): Vector support

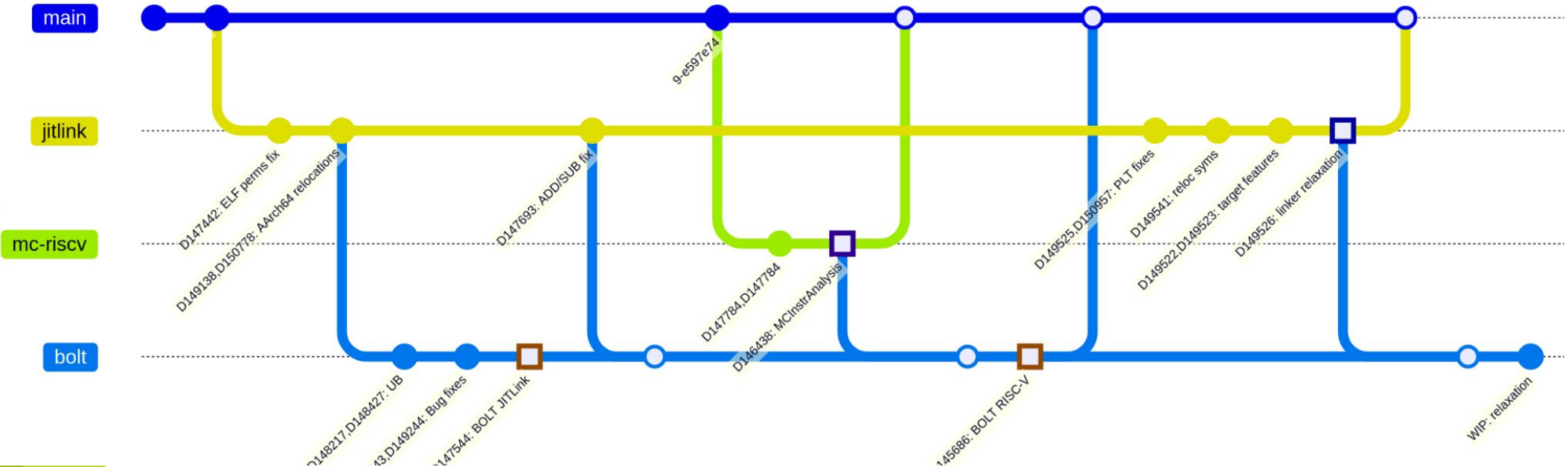
- Auto vectorisation with the superword-level parallelism (SLP) vectorizer
  - Not yet enabled upstream by default, but getting very close - working to ensure the cost model disables it when it's not beneficial.
  - Recent tuning e.g. using scalar instructions for copies/stores of small fixed size vectors.
- Intrinsics
  - v0.11.1 supported, eagerly awaiting v1.0 finalisation.

What's new (ish): BOLT

- BOLT is a post-link optimiser designed to speed up large applications.
  - => those suffering from high iTLB / I$ misses
- Takes information from a sampling profiler, disassembles functions and reconstructs the CFG, performing (primarily) code layout optimisations
- RISC-V port largely finished and almost merged.

What's new (ish): BOLT

Work needed to get BOLT upstream:

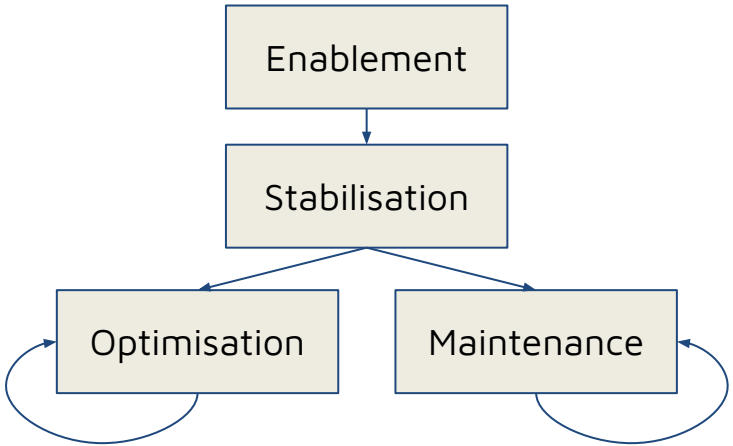


What's new(ish): Other

- LLVM libc
  - Same level of completeness as x86-64
- CI improvements
- llvm-mca
- Various newly supported ISA extensions (merged or WIP). e.g. z[f|d]inx, code size reduction extensions, vector crypto, zacas.

The future - RISC-V compilers work going forwards

- More ISA extensions, enablement of additional LLVM tooling and features.
- More targeted optimisations due to real hardware, different microarchitectures, investment in specific workloads.
- Very different kind of work to early enablement efforts
- How to take on more of a leadership role within toolchain-related projects?



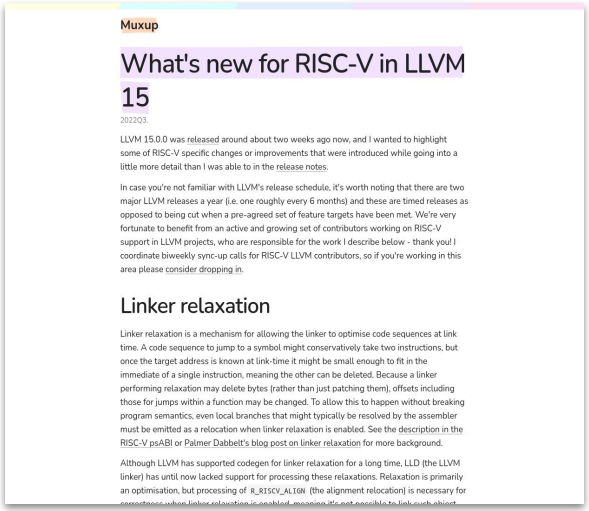
The future: features and development directions

Disclaimer: Not a declared roadmap, but an interpretation of areas people likely want to invest in. Unordered.

- Ongoing autovectorisation improvements
- LTO fixes
- CI and performance tracking
- Performance modeling
- llvm-mca
- MLIR RISC-V vector dialect exploration
- μarch-specific tuning + scheduling models
- Enable SLP autovectorisation
- Improved constant materialisation
- Security hardening features
- Formal verification
- Fuzzing
- Fuzzing
- LLDB
- TLSDESC
- Atomics ABI changes
- RV32E codegen
- GlobalISel
- libc
- LLVM-built Linux userspace investments
- Vector crypto
- SIMD
- Easy custom instruction definition flow
- ...

Get involved and keep track

- https://llvm.org/docs/RISCVUsage.html
- Biweekly contributor sync-ups (see “RISC-V on discourse.llvm.org)
- LLVM Weekly https://llvmweekly.org/
- Announcements posts for 6-monthly LLVM releases (https://muxup.com)



Muxup

What's new for RISC-V in LLVM 15

2022Q3.

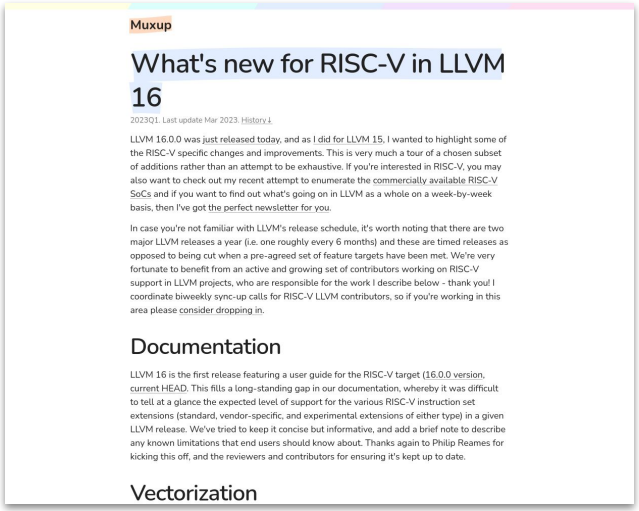
LLVM 15.0.0 was released around about two weeks ago now, and I wanted to highlight some of RISC-V specific changes or improvements that were introduced while going into a little more detail than I was able to in the release notes.

In case you're not familiar with LLVM's release schedule, it's worth noting that there are two major LLVM releases a year (i.e. one roughly every 6 months) and these are timed releases as opposed to being cut when a pre-agreed set of feature targets have been met. We're very fortunate to benefit from an active and growing set of contributors working on RISC-V support in LLVM projects, who are responsible for the work I describe below - thank you! I coordinate biweekly sync-up calls for RISC-V LLVM contributors, so if you're working in this area please consider dropping in.

Linker relaxation

Linker relaxation is a mechanism for allowing the linker to optimise code sequences at link time. A code sequence to jump to a symbol might conservatively take two instructions, but once the target address is known at link-time it might be small enough to fit in the immediate of a single instruction, meaning the other can be deleted. Because a linker performing relaxation may delete bytes (rather than just patching them), offsets including those for jumps within a function may be changed. To allow this to happen without breaking program semantics, even local branches that might typically be resolved by the assembler must be emitted as a relocation when linker relaxation is enabled. See the description in the RISC-V psABI or Palmer Dabbelt's blog post on linker relaxation for more background.

Although LLVM has supported codegen for linker relaxation for a long time, LLD (the LLVM linker) has until now lacked support for processing these relaxations. Relaxation is primarily an optimisation, but processing of `R_RISCV_ALIGN` (the alignment relocation) is necessary for



Muxup

What's new for RISC-V in LLVM 16

2023Q1. Last update Mar 2023. History ↓

LLVM 16.0.0 was just released today, and as I did for LLVM 15, I wanted to highlight some of the RISC-V specific changes and improvements. This is very much a tour of a chosen subset of additions rather than an attempt to be exhaustive. If you're interested in RISC-V, you may also want to check out my recent attempt to enumerate the commercially available RISC-V SoCs and if you want to find out what's going on in LLVM as a whole on a week-by-week basis, then I've got the perfect newsletter for you.

In case you're not familiar with LLVM's release schedule, it's worth noting that there are two major LLVM releases a year (i.e. one roughly every 6 months) and these are timed releases as opposed to being cut when a pre-agreed set of feature targets have been met. We're very fortunate to benefit from an active and growing set of contributors working on RISC-V support in LLVM projects, who are responsible for the work I describe below - thank you! I coordinate biweekly sync-up calls for RISC-V LLVM contributors, so if you're working in this area please consider dropping in.

Documentation

LLVM 16 is the first release featuring a user guide for the RISC-V target (16.0.0 version, current HEAD. This fills a long-standing gap in our documentation, whereby it was difficult to tell at a glance the expected level of support for the various RISC-V instruction set extensions (standard, vendor-specific, and experimental extensions of either type) in a given LLVM release. We've tried to keep it concise but informative, and add a brief note to describe any known limitations that end users should know about. Thanks again to Philip Reames for kicking this off, and the reviewers and contributors for ensuring it's kept up to date.

Vectorization

End

- Thanks again to all of the (many) contributors so far.
- Closing summary
  - RISC-V LLVM as a model of successful upstream collaboration.
  - Recent milestones and developments.
  - A vision for the future.
- Contact: asb@igalia.com