Keine Tutorien, regelmäßiges Online-Treffen alle 2-3 Wochen?

Emulierung RISC-V 64-bit auf x86-64 via Binary Translation

- RISC-V virtueller Speicher, gleicher Speicherbereich
- Emulator an anderer Adresse, sodass keine Kolission
- RISC-V Binary in Speicher laden
- Register init, Speicher (Stack) allozieren
- → Binary Translation
- Block Code decodieren bis Branch, x86-64 Binärcode generieren
- Semantisch äquivalent, läuft auf emulierten Registern
- Einsprung
- Nächste Codeadresse in RISC-V-Programm niederschreiben
- Wieder Code ziehen
- Komplexität Ready for Benchmarking, evtl. echte Programme
- Multithreading muss nicht (problematisch, nicht fürs Erste)

Optimierungspotential

- Overhead der Register im Speicher
- Register-Mapping, dynamisch welche gebraucht werden
- Übersetzung nicht nur Basic-Blöcke sondern Super-Blöcke
- Kontrollflussgenerierung performant NP-vollständig?
- LLVM Backend für Codegenerierung

C-Checkliste

- Makefiles anschauen!
- CMake
- Meson
- oder Make klassisch

Testsuite Binary Programme

- eigene ELF-Binaries in RISC-V zum Testen
- Glibc Standardlibrary kompilieren zum Testen
- gzip/bzip zum Testen
- Image Processing (ohne Multithreading!)
- Benchmarking

Memory allocation

- Malloc in C vs mmap allocate pages
- 2 Stdlibs in einem Speicher nicht möglich
- PROT_EXEC in mmap → pages sind executable
- 1 Virtueller Adressraum: RISC-V und unser Emulator im selben Adressraum

Startpunkt

- Alexis-Syscalls
- C-Programmierung
- RISC-V-Spezifikation
- x86-64-Architektur für die Übersetzung von 64-bit Code!
  → Materialien vom regulären ERA-Praktikum
- intel.com/sdm Differences x86-64 vs. -32
- System-V ABI 3.2, 3.3
- ABI RISC-V Documentation
- ELF-Format Contents, what is needed to load?
- Reading ELF files with MMAP in C?
- Decode Block and then generate code for x86 (von Hand oder lib?)