A year of RISC-V adventures: embracing chaos in your software journey

How I started from zero and ended up porting a JIT compilation library and assembling files by hand

Ekaitz Zárraga

Who I am

- Telecommunication engineer (EEE equivalent)
- Freelance engineer/programmer at ElenQ.Tech[1]
- Guix user and contributor

Recently

- Interested in small computing
- Lisp, specially Scheme, more specifically small Scheme implementations.

[1] https://elenq.tech

The beginnings

- Started a very small scheme compiler for small machines, and thought about RISC-V. I decided to target RISC-V assembly following “Let’s Build a Compiler” by Jack Crenshaw[2]
- I studied RISC-V with: “The RISC-V Reader: An Open Architecture Atlas”[3]
- I studied several scheme implementations and read some papers
- Nothing happened out of this, I gave up when I read about continuations but I learned some stuff.

---

[2] https://compilers.iecc.com/crenshaw/

[3] http://riscvbook.com/

Chaos starts

- One day a friend (the same guy who introduced me to Guix) tells me the Guix mailing list is looking for help to make a RISC-V port. I decide to raise my hand.
- Now I'm involved in RISC-V porting effort somehow.
- One day Andy Wingo, the Guile maintainer, mentions it would be interesting to port Lightening, Guile's JIT code generation library to RISC-V.

Now I’m working on the Lightening port somehow

Lightening is a machine code generation library. It’s a fork of GNU Lightning, made for Guile, that aims to be simple.

The experience

- My C programming skills are rusty
- 0 documentation: only the one inherited from GNU Lightning
- Dead code
- Andy Wingo helped me take the good direction and made everything easier

Lightening: Things learned

- How to assemble instructions by hand *
- Code is data. I'm a lisp guy, I already knew that! *
- Some cool GDB debugging tricks I already forgot
- Machine code generation is not that complex
- Relocations and immediates are painful[4]

[4] Read more: https://ekaitz.elenq.tech/machine-code-generation.html

Lightening: Assemble by hand

```
addi a0, zero, 56
```

- Opcode `addi`: `0010011`
- Destination register `a0`: `01010`
- `funct3`: `000`
- Source register `zero`: `00000`
- The immediate `56`: `000000111000`

`000000111000 | 00000 | 000 | 01010 | 0010011`

All together: `00000011100000000000010100010011`
(In hex: `0x3800513`)

Lightening: Code is data

```c
#include<stdint.h>
#include<stdio.h>

typedef int f0(void);

int main(int argc, char* argv[]){
    uint32_t instructions[2];

    instructions[0] = 0x03800513; // addi a0, zero, 56
    instructions[1] = 0x00008067; // jalr zero, ra, 0

    // Reinterpret the array address as a function
    f0 *load_56 = (f0*) instructions;

    int a = load_56();
    printf("%d\n", a);
}
```

Moving to the bootstrap system

There was a chance to work on Guix's RISC-V support via NlNet and the guys involved in the porting effort told me I should send a proposal.

I had to learn about the full-source bootstrap project for this so I ended up taking part on it.

The proposal was rejected, but the learning was already done.
And what now?

Stage0

Stage0 is a full source bootstrap system.

Stage0's steps:

1. Hex0: Hex encoded raw ELF file with comments
2. Hex1: Hex0 + one character labels and some extras
3. Hex2: Hex1 + proper labels and reasonable basics
4. M0 (macro system): A simple macro system
5. M2-Planet: A C subset that uses M0 as output
6. C compilers (i.e. GNU Mes's `mescc`)

Stage0

The experience

Hex0

Hex0 is a simple assembler written in Hex0. It just converts a Hexadecimal encoded ELF file to binary.

It's heavily commented so anyone can decode the instructions and make sure that it works as expected. It looks like this:

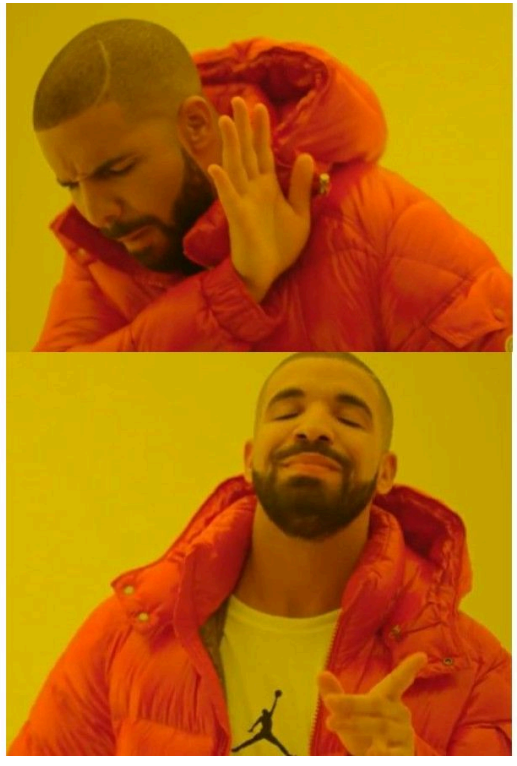
```
# :_start ; (0x0600078)

; Open input file and store FD in s2
93 08 80 03    # RD_A7 !56 ADDI       ; sys_openat
13 05 C0 F9    # RD_A0 !-100 ADDI     ; AT_FDCWD
93 05 06 00    # RD_A1 RS1_A2 MV      ; input file
13 06 00 00    # RD_A2 MV             ; read only
73 00 00 00    # ECALL
13 09 05 00    # RD_S2 RS1_A0 MV      ; Save fd
```

Hex0

I wanted to make the first version of the POSIX based Hex0 for RV64. I just needed to:

1. Write it in assembly first
2. Assemble everything by hand...



Hex0: Abandoned projects that add entropy: pysc-v

While I was working on Lightening I started a RISC-V assembler in Python.

I abandoned the project[5]

But the backend was reusable so I could generate all the instructions in the Hex0 with the format I wanted!

[5] http://git.elenq.tech/pysc-v/

Hex0: disassembly tricks

I still need to calculate addresses by hand though.

But wait! I learned how to disassemble when I worked on Lightening so I could obtain each instructions address easily, without all the counting.

Hex0

The experience

- It was rewarding to see that I could write assembly.
- I reused all the GDB tricks from Lightening to check that everything was working correctly.
- The documentation is weird and there are too many subprojects that make the project hard to understand for newcomers. It's hard to understand the reasons behind some decisions and everything looks fragile.
- But `#bootstrappable` at `libera.chat` is a good place to ask questions and learn from others.

Hex0 extra: embrace boredom

Sunday morning. I'm bored.

I reviewed the status of the project and saw RV32 is not ready yet, even if it is similar to RV64.

I start a discussion on IRC and in less than half an hour and with the help of the Wikipedia we replace the ELF headers and make the Hex0 support for RV32.

Status of the projects

- **Lightening**: is passing all the test but needs testing. The code[6] is available for anyone to read and improve.
- **Stage0**: included my small RISC-V contribution to Hex0[7] and other contributors expanded my work to every stage.

[6] https://gitlab.com/wingo/lightening/-/merge_requests/14/commits
[7] https://github.com/oriansj/bootstrap-seeds/pull/2

My status

I tried again with NlNet, with a proposal for some RISC-V porting efforts on GNU Mes and other bootstrap related projects.

They look interested on a full-source-bootstrap for RISC-V!

I might end up working on a compiler

Conclusions

- All this is just random work, done without a plan or purpose
- Embrace the chaos of life, stay curious and you may reach interesting places
- If you can, learn from people: it’s faster and better
- You don’t need to be a genius: most of the things are easy once you know the context.

Just try, and let it happen