New issue

Jump to bottom

How to encode and decode instructions #806

Open **ethindp** opened this issue on Dec 31, 2021 · 0 comments

ethindp commented on Dec 31, 2021

I'm writing an assembler for RISC-V (mainly as a curiosity but it might become more in future) and am struggling to understand how decoding and encoding works. Examining the opcodes parser (which I had fun translating to Rust) yields that, for an instruction `i`, there exists an associated mask `m` and an associated match `M`. The algorithm for decoding (say) an `ADD` instruction is therefore as simple as:

1. Scan the instruction stream. Let `i` be any instruction.
2. Let `m` be the mask for `ADD` (`0xFE00707F`).
3. Let `M` be the match value for ADD (`0x33`).
4. For each instruction:
    i. Calculate `(i & m)` and determine if `m = M`.
    ii. If `m = M`, extract bits `11:7` for `rd`, bits `19:15` for `rs1`, and bits `24:20` for `rs2`, and return the decoded instruction.
    iii. Otherwise, continue.

I know I used a bit of mathematical notation here, and I hope no one minds. If the above algorithm is indeed the correct way of detecting and disassembling instructions (minus symbol and label resolution), how does one go about encoding them? I'm using the parse-opcodes program I ported to Rust from Python to get all the instructions, dump them into a Rust enum, and then to write an function that determines what instruction its processing at the moment (I assume 128-bit integers for each value, and though that's overkill, the RISC-V ISA manual does hint at a possible extension sometime in the (far?) future for a variable-length instruction width).

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

1 participant