One of the goals I have forAirScript is to make the language fully composable and enable writing modular “gadgets”. But as I’ve been thinking of how to accomplish this, I realized that AirScript is not the right level for where code composition should happen. There are just too many high-level constructs (and more are on the way) which makes code composition non-trivial.

Enter AirAssembly, which is a low-level language for encoding Algebraic Intermediate Representation (AIR) of computations. The goal of the language is to provide a minimum number of constructs required to fully express AIR for an arbitrary computation, and to make code composition much much simpler.

Unlike AirScript, AirAssembly is intended to be a compilation target for higher-level languages (of which AirScript will be one, but maybe other languages will emerge over time). I’m also hoping that AirAssembly could become a common language understood by various STARK libraries. So, the model could be something like this:

[

image

1389×573 13.6 KB

](https://ethresear.ch/uploads/default/original/2X/a/a5eedc914d812508dab9d72b4bdea813fb36bc17.png)

I’ve just published the first draft ofAirAssembly specifications, and would love to get feedback/comments from people working with or interested in STARKs.

Also, here is the obligatory MiMC example written in AirAssembly:

(module (field prime 340282366920938463463374607393113505793) (const 3) (static (cycle 42 43 170 2209 16426 78087 279978 823517)) (transition (span 1) (result vector 1) (add (exp (load.trace 0) (load.const 0)) (load.static 0))) (evaluation (span 2) (result vector 1) (sub (load.trace 1) (add (exp (load.trace 0) (load.const 0)) (load.static 0)))) (export mimc128 (steps 256)) (export mimc256 (steps 1024)))