Mini-Documentation: Executable Format & Compilation Process

Section 1: Executable Format

Global Tables

- global_to_local: maps global_id → (module_id, local_id)
- module_metadata: module_id → metadata (first_extern_function, file offset, table lengths)

Per-Module Tables

- function_ends: function_id → end-offset (rel. to functions block)
- extern_to_global: extern_id → global_id
- functions block: concatenated bytecode, offsets relative to block start

Function Call Mechanism

```
if magic < first_extern_function:
    module_id = current_module_id
    function_id = magic
else:
    ext_id = magic - first_extern_function
    glob_id = extern_to_global[ext_id]
    module_id, function_id = global_to_local[glob_id]

# Use (module_id, function_id) to jump in target module's functions block.
```

Section 2: Compilation Process

1. Parsing

Input: source files → Output: AST

2. Symbol Resolution

Enrich AST with exports/imports, ensure declarations

3. Airport (Dependency Resolver)

Modules wait by dependencies; tree-shaking discovers & compiles only needed modules

4. Type Check & Auto Cast

Check types, insert implicit casts (e.g. common type of a+b may be neither operand type)

5. High-Level Assembly IR

Generate JS-object-based IR for each function

6. Target Assembly Generation

Translate IR into 12-bit or 6-bit instructions

7. Executable Assembler

Build tables, functions block, final executable format

Philosophy Nuggets

- Airport step: each module "waits for itself" to avoid special casing.
- Auto-cast: result type of a+b may be neither operand type (promote each component).