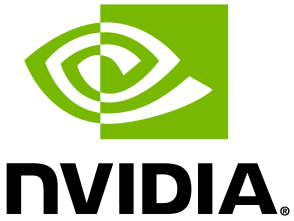

Assessment for JR2005761 Graphics and AI Tools Engineering Intern

Write a WebAssembly interpreter (or native compiler) that can execute the test modules in `tests/wat/`. Review the test files to determine which instructions and features need to be supported.

Treat this as a work assignment you would receive at NVIDIA. Write well-tested, production-level code. Document your design decisions, what you've implemented, and any challenges or limitations you encountered. Include tests demonstrating correctness and handle edge cases appropriately.

Try to get as far as possible. The interpreter doesn't have to fully work or pass all tests. Demonstrate your approach, show what you've implemented, and document where you encountered challenges or limitations.

The interpreter must be written in C++ and built using CMake. You can develop on any platform (Linux, macOS, Windows), but the solution will be tested on Windows, so ensure it builds and runs there.

You may use any tools, references, documentation, or resources at your disposal, including AI coding assistants. However, **all code must be your own original work**. You cannot simply copy implementation code from existing interpreters. The goal is to evaluate your problem-solving skills and coding abilities.

Module Structure

Your interpreter must handle function definitions with parameter and return types, local and global variables (both mutable and immutable), memory with initial size and growth, function tables for `call_indirect`, data segments for memory initialization, element segments for table initialization, exports, and imports including basic WASI functions.

Tools

wat2wasm

`wat2wasm` converts WebAssembly text format to binary format. It's part of the WebAssembly Binary Toolkit (wabt).

On macOS, install it with `brew install wabt`. On Linux, download from https://github.com/WebAssembly/wabt/releases or build from source. On Windows, download pre-built binaries from the releases page.

The tool suite includes `wat2wasm` (text to binary) and `wasm2wat` (binary to text).

Test Files

Tests are organized by priority and feature set:

- `01_test.wat` - Basic MVP i32 operations
- `02_test_prio1.wat` - Functions, floats, conversions
- `03_test_prio2.wat` - i64, data segments, call_indirect
- `04_test_prio3.wat` - Additional MVP features
- `05_test_complex.wat` - Complex control flow
- `06_test_fc.wat` - Non-trapping float-to-int conversions
- `07_test_bulk_memory.wat` - Bulk memory operations
- `08_test_post_mvp.wat` - Post-MVP features (optional)
- `09_print_hello.wat` - WASI output example

The priority files `02`, `03`, and `04` contain the core MVP features needed for most use cases.

Extra Credit

For extra credit, try to find real wasm/wasi projects online and get them to work with your interpreter. Examples include Lua, Coremark, or Python. If you do, document which projects you attempted, what worked, and what didn't.