Design, Implementation and Comparison of Software Emulation Techniques

by Oniga Andrei-Mihai
under the supervision of
Lect. Dr. Mihai Andrei

Motivation & Scope

Why was this done? Who for?
Why Brainfuck and CHIP-8?

Project Goals

Implementation
(interpreter / JIT compiler),

Optimizations,

and Program Analysis

Brainfuck Machine Model

Relatively infinite memory tape

with a pointer

Brainfuck Language Overview

+	Increments the value at the current position that the machine points to.
-	Decrements the value at the current position that the machine points to.
<	Moves the pointer one cell to the left.
>	Moves the pointer one cell to the right.
[	Jumps after the corresponding closed bracket when the value at the current cell is 0.
]	Jumps after the corresponding open bracket when the value at the current cell is not 0.
.	Outputs the value at the current cell.
,	Read a value to be placed at the current cell.

Intermediate Representation (IR)

```
typedef enum bf_operation
{
    // Basic instructions, no optimizations
    BF_INSTRUCTION_INC = 0,
    BF_INSTRUCTION_DEC,
    BF_INSTRUCTION_NEXT,
    BF_INSTRUCTION_PREV,
    BF_INSTRUCTION_JUMP_START,
    BF_INSTRUCTION_JUMP_BACK,
    BF_INSTRUCTION_INPUT,
    BF_INSTRUCTION_OUTPUT,
    BF_INSTRUCTION_END,

    // Optimized instructions
    BF_INSTRUCTION_ADD,
    BF_INSTRUCTION_MOVE,
    BF_INSTRUCTION_JUMP,

    // Composite instructions
    BF_INSTRUCTION_CLR,
    BF_INSTRUCTION_ADDCLR,
    BF_INSTRUCTION_MOVNZ
} bf_operation_t;
```

Translating Brainfuck into Efficient Structures

Optimizations Applied

```c
typedef enum bf_optimizations {
    BF_OPTIMIZATIONS_NONE = 0,
    BF_OPTIMIZATIONS_INSTRUCTION_FOLDING,
    BF_OPTIMIZATIONS_JUMP_CACHING,
    BF_OPTIMIZATIONS_LOOP_REPLACEMENT,
    BF_OPTIMIZATIONS_ALL
} bf_optimizations_t;
```

Jump caching, Instruction Folding and Pattern Matching

Interpreter Implementation

```c
typedef struct bf_interpreter
{
    bool running;
    uint16_t pc;
    uint16_t index;
    dynarray_t program;
    bf_state_t* state;
} bf_interpreter_t;
```

Basic Dispatch Loop and The Fetch Decode Execute cycle

Static JIT Compilation

```c
typedef struct bf_jit_lightning
{
    bool running;
    bf_state_t* state;
    jit_state_t* jit_state;
    bf_jit_function_t code;
} bf_jit_lightning_t;
```

Choosing a library,

why GNU Lightning,

implementation Details

Optimization Insights

When Simplicity Beats Aggressiveness: replacing too many loops becomes redundant as the generated JIT code is identical

Performance Results (Brainfuck)

Benchmark Comparisons

Brainfuck Testing Strategy

Ensuring Correctness with Unit Tests

CHIP-8 Architecture

64 KB Memory,

16 Registers,

35 Instructions, 2 Timers,

64 x 32 XOR Display

and 16-key Input

Interpreter Implementation

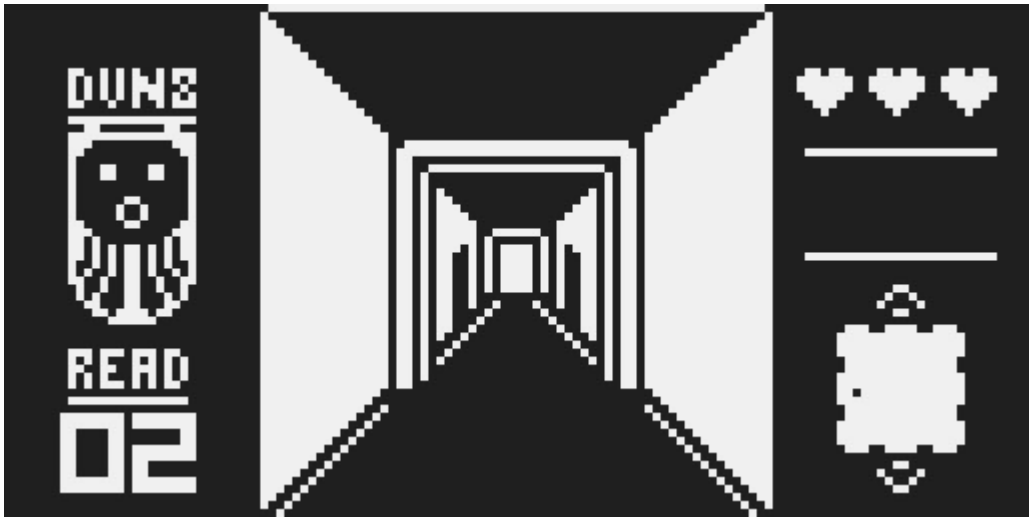
Modular Design and Execution Strategy

```c
typedef struct bf_state
{
    bf_input_f in;
    bf_output_f out;
    uint8_t* memory;
    void* aux_arg;
} bf_state_t;
```

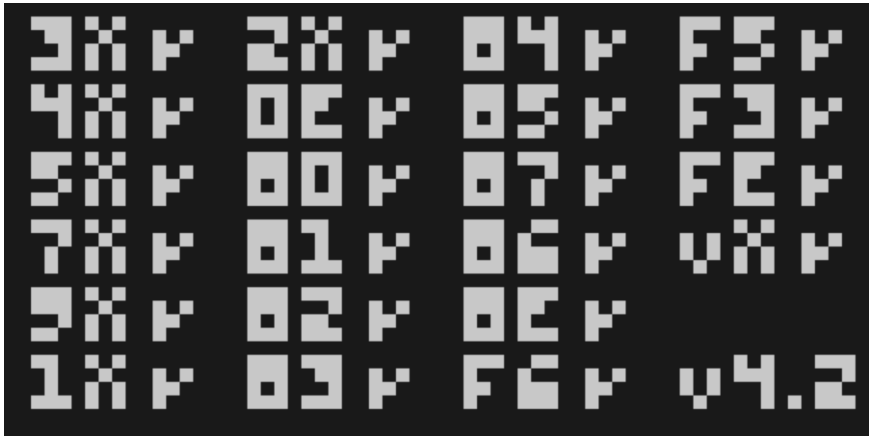
```c
typedef struct cbf_context {
    bf_state_t state;
    union {
    #ifdef JIT_LIGHTNING
        bf_jit_lightning_t jit_lightning;
    #endif
        bf_interpreter_t interpreter;
    } cpu;
    bf_run_mode_t cpu_run_mode;
    uint8_t memory[0x10000];
    dynarray_t output;
} cbf_context_t;
```

Quirks & Extensions

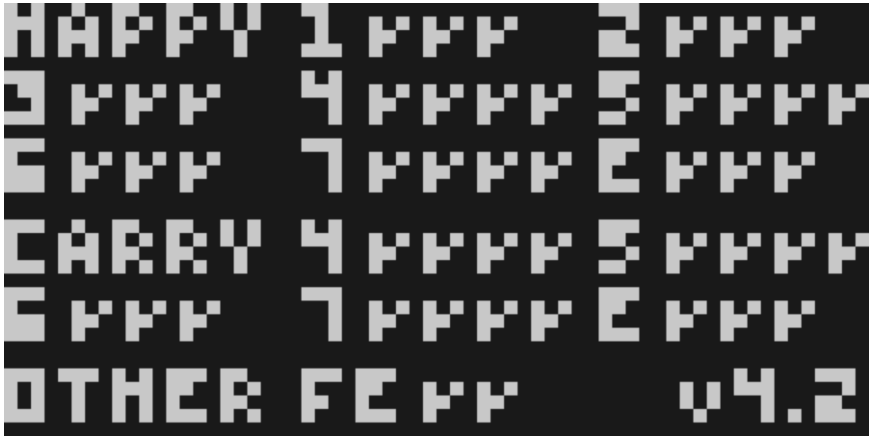
Modern SCHIP-8 Support and Differences



CHIP-8 Testing – Timendus' Test Suite



Instruction corectness check



Flag corectness check

CHIP-8 Testing – Timendus' Test Suite

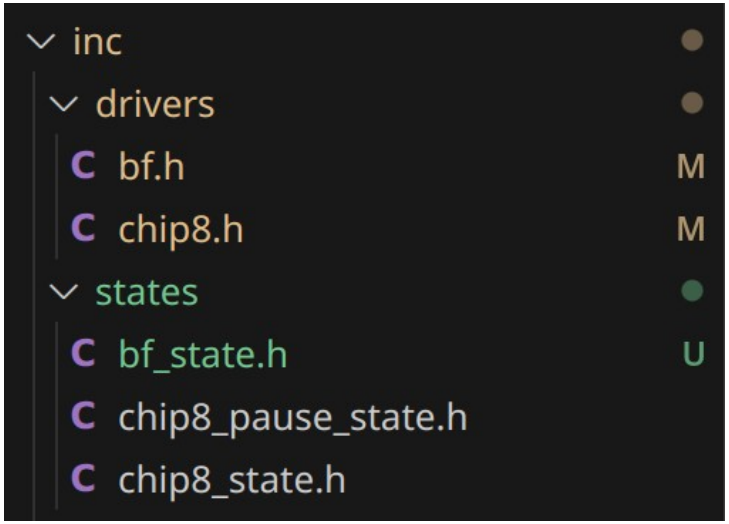
Quirk Test in SCHIP Mode

Quirk Test in Normal Mode

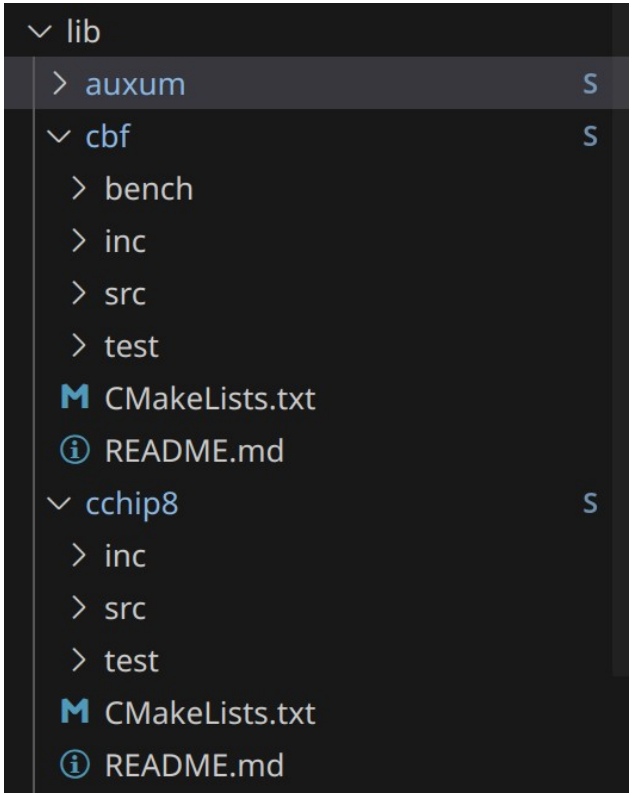
The Final Application – Edra

Glueing everything toghether in C17

Design and Architecture

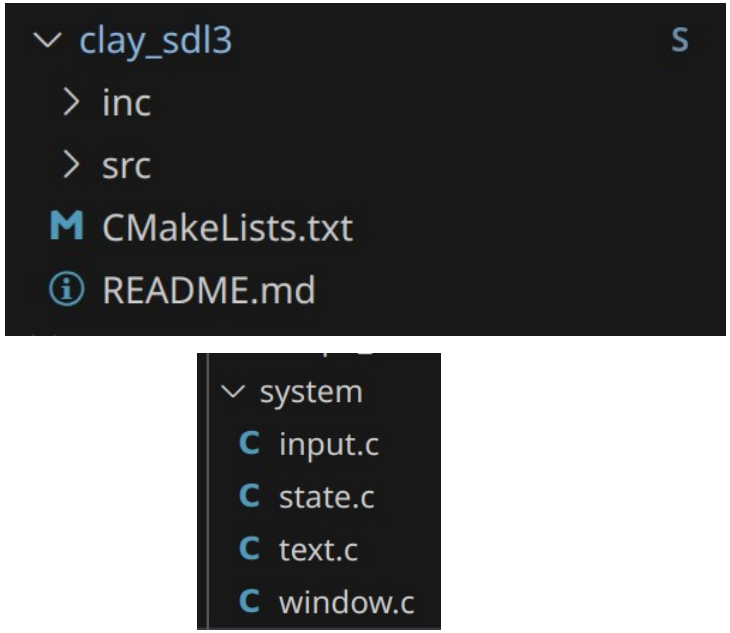


Modularity and separation of concerns.



User Interface in SDL3 using CLay

Handling Graphics, Input, and Display



Cross-Platform Porting

Windows:

- No JIT as GNU Lightning works only on POSIX compliant systems.

Linux (+Android on Termux + Termux X11 display server):
- Works out of the box with all modules properly implemented.

PSVita:
- Gamepad input only and also no JIT as system is not POSIX compliant.

Build System & Usage

Makefile Structure and Platform Targets

How to compile:

- specify SDL3_DIR / SDL3_TTF_DIR in the build cache of CMake if needed.
- run make b{platform}{d/r} for an automated build.
    - platforms:
        - w - Windows
        - u - Unix
        - v - PSVita
    - d / r -> debug / release.
- make r for an automated run.

Future Work

UX Improvements, More Platforms,
More Architectures

Final Thoughts

Contributions to entry level

Emulation and to Education

Showcase and Demonstration

Running on Desktop and PS Vita
Live!

The End

Thank You