!!! info inline end ""

!!! danger ""

    Programming Language: C
    Version Control: Git
    Build System: GNU Make
    AST VISUALIZER: Graphviz
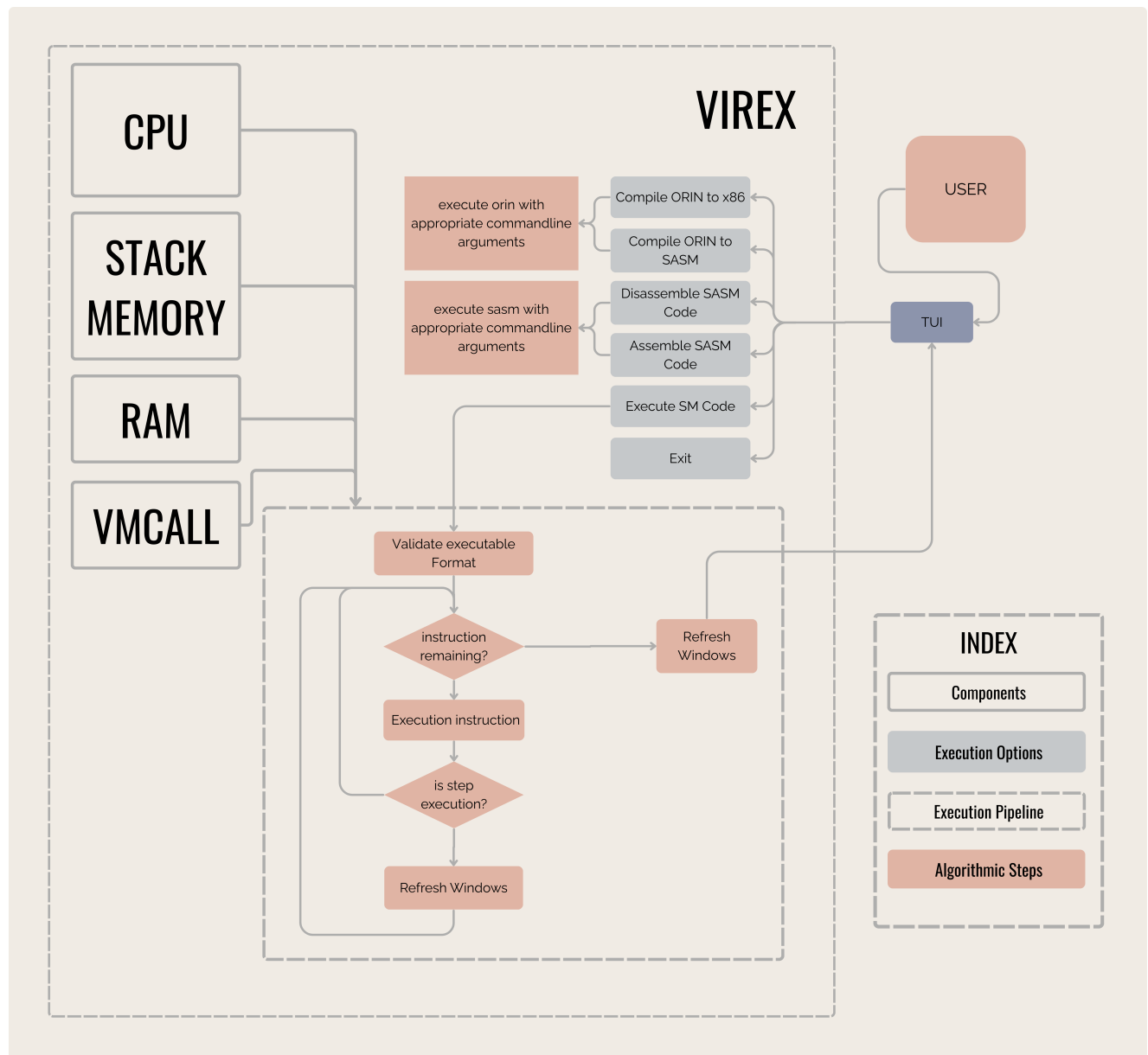
Tech Stack
!!! warning ""

Other than the above mentioned tools, we have also used libraries like
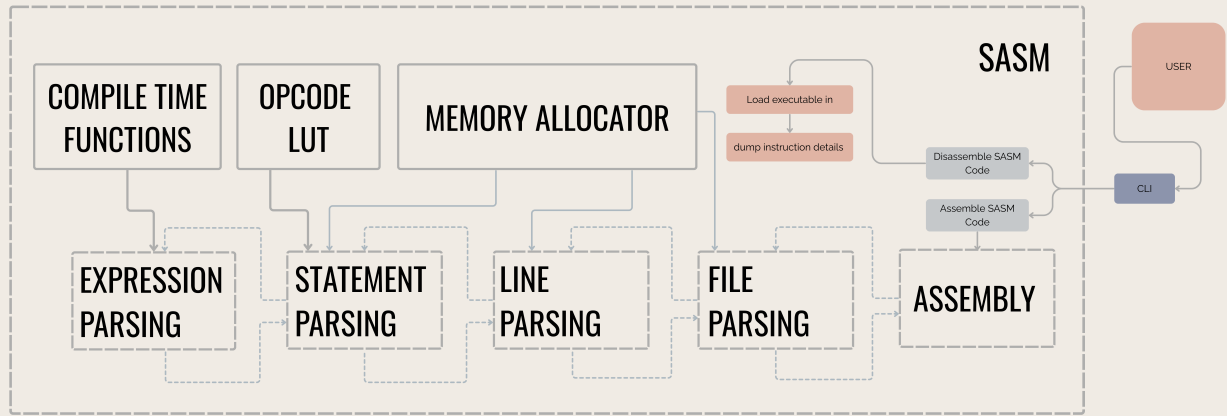NCurses to build the UI.

```
VIREX
├── build/
├── docs/                              # Reference documentation
├── examples/                          # Sample programs
├── extras/
│   └── orin_grammar.txt               # Grammer for ORIN
(PROTOTYPE)
├── html/                              # Latest Build of the
documentation website
├── include/                           # Public headers for VM,
SASM, OCC
├── install.sh                         # Install script for linux
├── LICENSE                            # GPL-3
├── Makefile                           # RULES: virex, sasm, occ,
html, clean
├── mkdocs.yml
├── README.md
├── requirements.txt                   # requirements for mkdocs
├── src/                               # Core implementation (VM,
assembler, compiler)
├── tests/                             # Simple Test programs
written in SASM
└── tools/
    ├── fonts/
    │   └── EnvyCodeR/                  # font used when building
TUI
    └── themes/
        ├── kitty/                      # kitty terminal conf used
when building TUI
        └── vs_code/
            └── sasm_syntax_highlight/    # VS Code syntax highlighter
```

Architecture

VIREX

SASM

SASM

COMPILE TIME FUNCTIONS | OPCODE LUT | MEMORY ALLOCATOR

Load executable in

dump instruction details

USER

Disassemble SASM Code

Assemble SASM Code

CLI

EXPRESSION PARSING | STATEMENT PARSING | LINE PARSING | FILE PARSING | ASSEMBLY

Individual Components of SASM

ASSEMBLY

create and push a "global" scope

"translate" the input file

pop scope

resolve all unresolved operands

resolve entry point address

create executable

(a) Assembly Pipeline

FILE PARSING

Create a Lexer class object

load input file into lexer

convert these lines into blocks of statements

Translate the Code block

generate an AST for current file

(b) File Parsing

LINE PARSING

Line remaining?

Discard comments

Identify line type

Parse statement from line

Add Statement to Code Block

move to next line

(c) Line Parsing

EXPRESSION PARSING

Load the string into tokenizer

tokens remaining?

Fetch next token

convert text into valid expressions

(e) Expression Parsing

STATEMENT PARSING

is directive? | is Label? | is Instruction?

process the instruction body into expressions

NOTE: the structure for an instruction line consists of "instruction name" and "body"

entry | bind | include | Scope

convert the label name from string to expression format

process the expression from the passed directive body

process the binding name and value expressions from the passed directive body

process the expression from the passed directive body

convert lines into CodeBlock until end directive is found

NOTE: The structure of a directive statement consists of a head ptr to a linked list of statements, and as such, the entire block will be treated as a single statement of type 'local scope'

(d) Statement Parsing