VIREX : Virtual Execution Console

VIREX (**VIR**tual **EX**ecuter) is a platform-independent virtual machine designed around a flexible intermediate language called **SASM** (Simulated Assembly). It’s inspired by the **Java Virtual Machine (JVM)**, but unlike JVM bytecode, SASM is **open, readable, and writable** — you can program directly in it.

🚀 What is SASM?

Just like Java compiles to bytecode for the JVM, any language can be compiled into SASM for VIREX. The difference is:

- SASM is **assembly-like**, human-readable, and editable.
- SASM is **open**, letting anyone build tools and languages around it.

You can even create your own programming language that compiles into SASM and runs anywhere VIREX runs — making your language instantly portable.

🧠 Why SASM?

- Learn how **assembly-level code** works through a clean and simplified syntax.
- Build a **compiler** without worrying about machine-level code generation.
- Make your own language **platform-independent** by targeting SASM.

🛠️ Current Features

- ✅ **VS Code syntax highlighter** for SASM
- 🌲 **AST visualizer** for seeing how your SASM code is parsed and compiled
- 🔧 A new programming language called **ORIN** is currently under development. It is being designed to compile directly to SASM.

> If you're interested in compilers, language design, or virtual machines — **contributions are very welcome**!

📦 Project Structure

```
/docs/       # Reference documentation
/examples/   # Sample programs
/include/    # Public headers for VM, SASM, OCC
/src/        # Core implementation (VM, assembler, compiler)
/tests/      # Simple Test programs written in SASM
/tools/themes/vs_code/ # VS Code syntax highlighter
/install.sh # Install script for linux
```

🧪 Getting Started (LINUX)

1. **Clone this repo:**

```
git clone https://github.com/your-username/virex.git
cd virex/
```

2. **Build the project (requires `sudo`):**

```
./install.sh
```

3. **Run an example program:**

```
cd ./examples/SASM/
virex
```

> If the **TUI doesn't render properly**, try adjusting your **terminal font size**.

> If that doesn't help, you can tweak layout values in **src/VM/vm_tui.c::CreateWindows()**.
> The constants used are defined as **percentages** of the screen dimensions.

> P.S. **kitty terminal** config, and font used, are available in `/tools`

4. **Inside VIREX, do the following:**

- Select **"Run SASM/ORIN command with custom flags"**
- Enter the following command:

```
-i helloWorld.sasm -I ./ -o tmp.sm
```

> ⌨️ use **Arrow keys** for navigation in menu.

- Select **"SASM build and exec"** by pressing **'a'**
- Enter the output filename (`tmp.sm`)

5. **Activate the syntax highlighter in VS Code**

- Open VS Code
- Press `Ctrl + Shift + P`
- Type: `Preferences: Color Theme`
- Select: `Palenight+sasm`

> 🎨 Open any `.sasm` file in vs code to see the syntax highlighter at work!

💡 Want to Contribute?

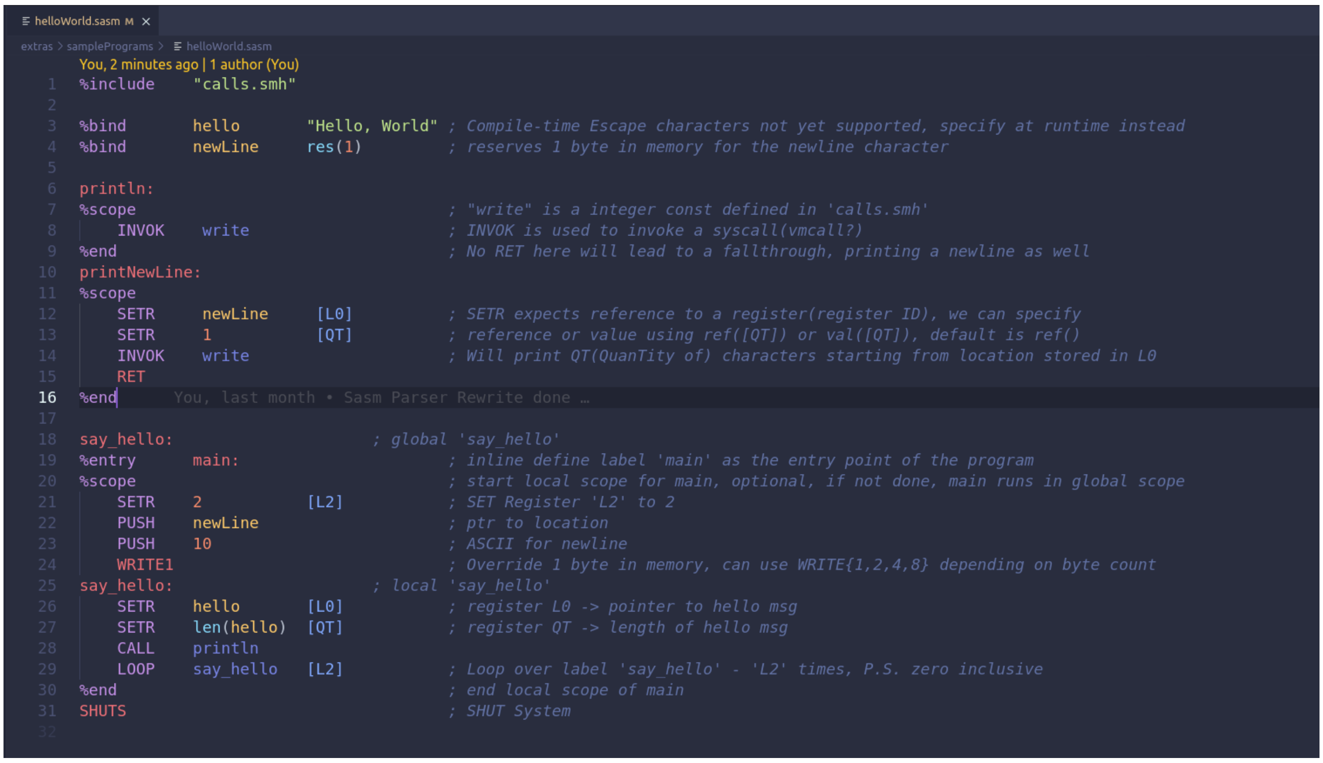
We're actively building:

1. The **ORIN programming language**
2. Improved **SASM tooling** (UI, debuggers, optimizers, etc.)
3. Expanded **documentation** and **tutorials**

📌 **For contribution guidelines and a roadmap, see CONTRIBUTING.md (coming soon).**

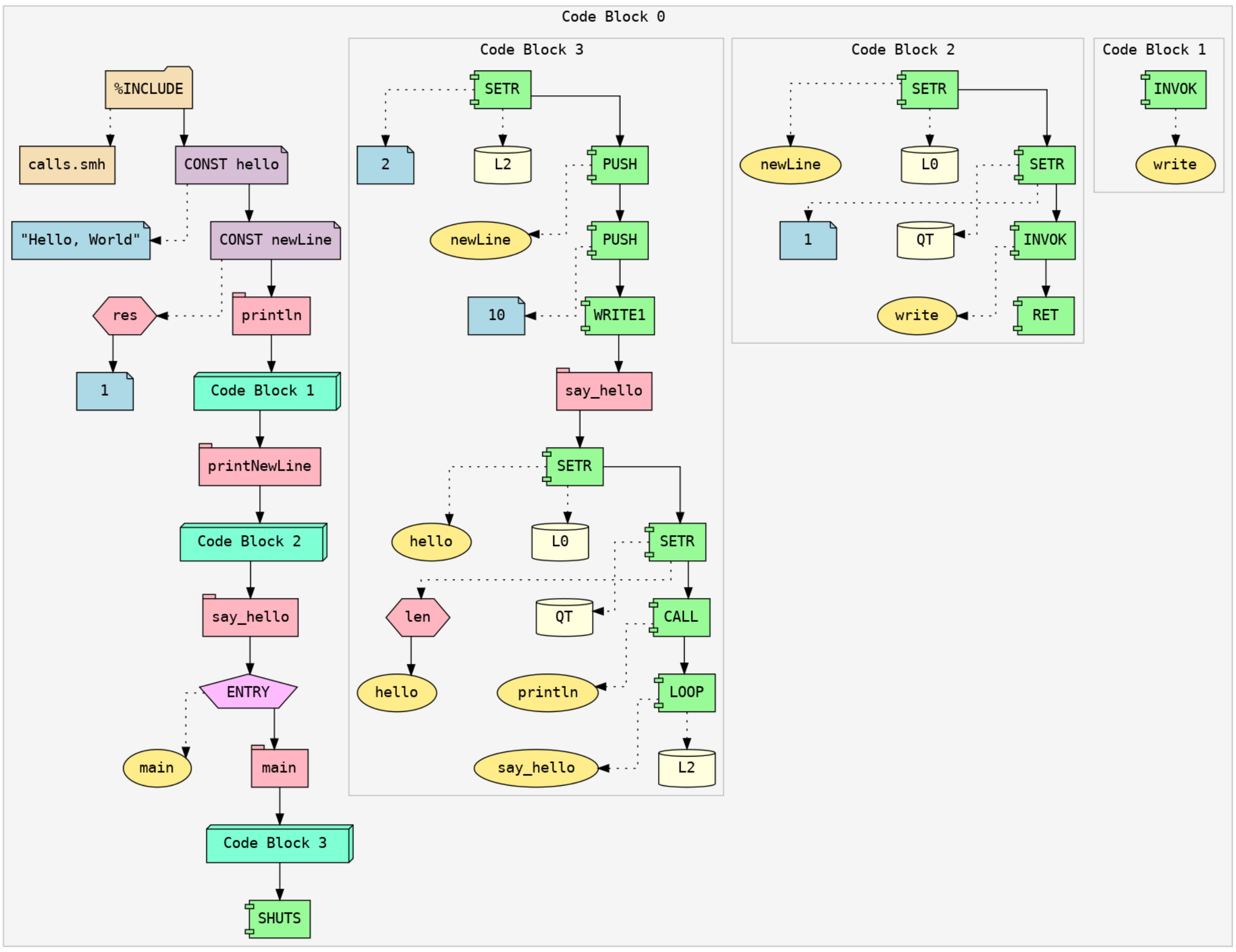
Examples

Syntax Highlighting:


helloWorld.sasm M ×
extras > samplePrograms > helloWorld.sasm
You, 2 minutes ago | 1 author (You)
You, last month • Sasm Parser Rewrite done …

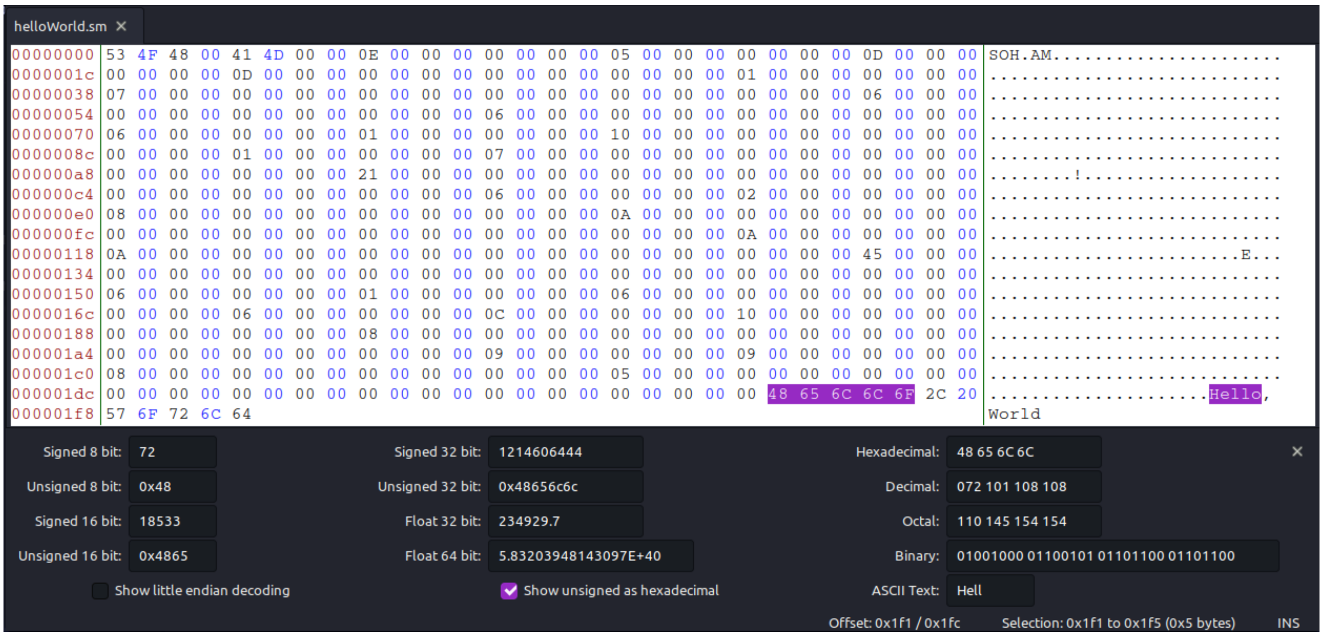
```
%include    "calls.smh"

%bind       hello       "Hello, World" ; Compile-time Escape characters not yet supported, specify at runtime instead
%bind       newLine     res(1)         ; reserves 1 byte in memory for the newline character

println:
%scope                                 ; "write" is a integer const defined in 'calls.smh'
    INVOK    write                     ; INVOK is used to invoke a syscall(vmcall?)
%end                                   ; No RET here will lead to a fallthrough, printing a newline as well
printNewLine:
%scope
    SETR     newLine     [L0]          ; SETR expects reference to a register(register ID), we can specify
    SETR     1           [QT]          ; reference or value using ref([QT]) or val([QT]), default is ref()
    INVOK    write                     ; Will print QT(QuanTity of) characters starting from location stored in L0
    RET
%end

say_hello:                     ; global 'say_hello'
%entry      main:                      ; inline define label 'main' as the entry point of the program
%scope                                 ; start local scope for main, optional, if not done, main runs in global scope
    SETR    2           [L2]           ; SET Register 'L2' to 2
    PUSH    newLine                    ; ptr to location
    PUSH    10                         ; ASCII for newline
    WRITE1                             ; Override 1 byte in memory, can use WRITE{1,2,4,8} depending on byte count
say_hello:                     ; local 'say_hello'
    SETR    hello       [L0]           ; register L0 -> pointer to hello msg
    SETR    len(hello)  [QT]           ; register QT -> length of hello msg
    CALL    println
    LOOP    say_hello   [L2]           ; Loop over label 'say_hello' - 'L2' times, P.S. zero inclusive
%end                                   ; end local scope of main
SHUTS                                  ; SHUT System
```

AST:



Note: Each Code Block in the visualized AST represents a Scope, Block 0 being global scope.

Binary Executable:

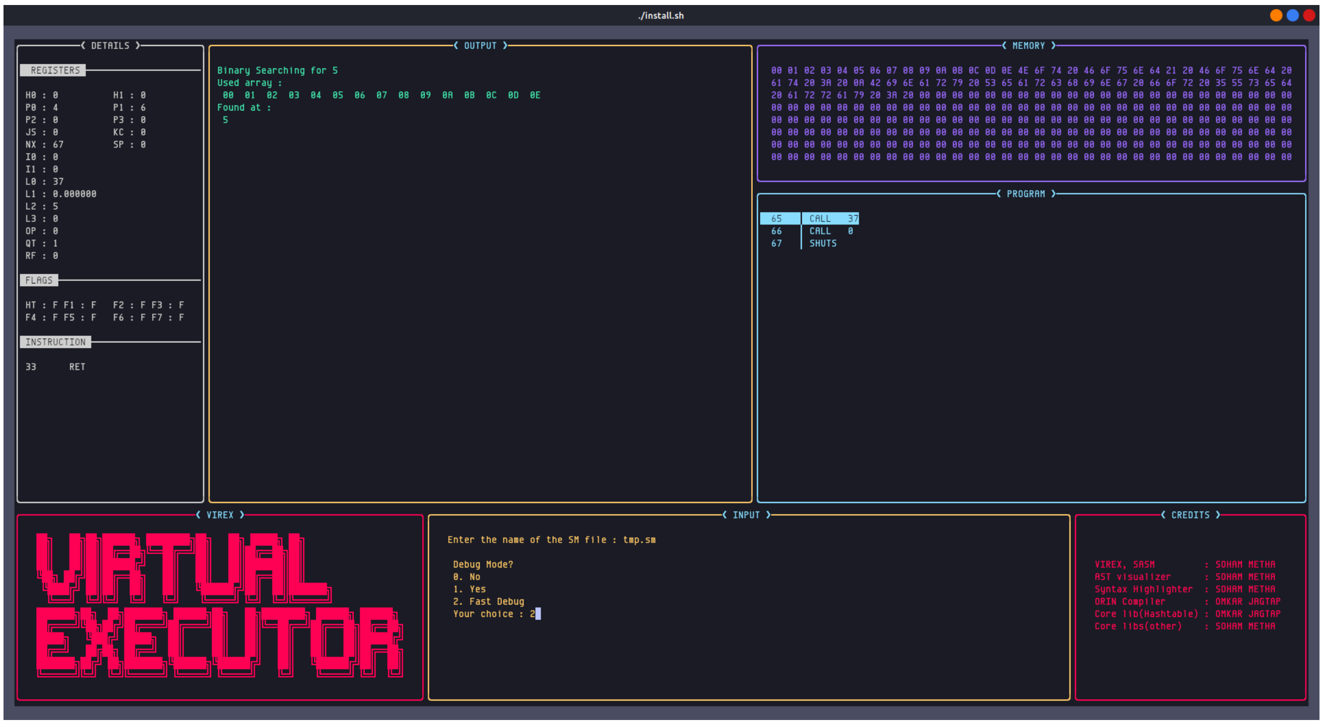


helloWorld.sm

```
00000000 53 4F 48 00 41 4D 00 00 0E 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00 0D 00 00 00 SOH.AM......................
0000001c 00 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 ............................
00000038 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 ............................
00000054 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ............................
00000070 06 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 ............................
0000008c 00 00 00 00 01 00 00 00 00 00 00 00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ............................
000000a8 00 00 00 00 00 00 00 00 21 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ........!...................
000000c4 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 ............................
000000e0 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0A 00 00 00 00 00 00 00 00 00 00 00 ............................
000000fc 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0A 00 00 00 00 00 00 00 ............................
00000118 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00 00 00 ........................E...
00000134 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ............................
00000150 06 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 00 00 00 00 ............................
0000016c 00 00 00 00 06 00 00 00 00 00 00 00 0C 00 00 00 00 00 00 00 10 00 00 00 00 00 00 00 ............................
00000188 00 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ............................
000001a4 00 00 00 00 00 00 00 00 00 00 00 00 09 00 00 00 00 00 00 00 09 00 00 00 00 00 00 00 ............................
000001c0 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 ............................
000001dc 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 48 65 6C 6C 6F 2C 20 .....................Hello,
000001f8 57 6F 72 6C 64                                                                      World
```

```
Signed 8 bit: 72            Signed 32 bit: 1214606444              Hexadecimal: 48 65 6C 6C
Unsigned 8 bit: 0x48        Unsigned 32 bit: 0x48656c6c            Decimal: 072 101 108 108
Signed 16 bit: 18533        Float 32 bit: 234929.7                 Octal: 110 145 154 154
Unsigned 16 bit: 0x4865     Float 64 bit: 5.83203948143097E+40     Binary: 01001000 01100101 01101100 01101100
[ ] Show little endian decoding   [✓] Show unsigned as hexadecimal  ASCII Text: Hell
                                  Offset: 0x1f1 / 0x1fc    Selection: 0x1f1 to 0x1f5 (0x5 bytes)    INS
```

GUI:



```
./install.sh

DETAILS
REGISTERS

H0 : 0          H1 : 0
P0 : 4          P1 : 6
P2 : 0          P3 : 0
JS : 0          KC : 0
NX : 67         SP : 0
I0 : 0
I1 : 0
L0 : 37
L1 : 0.000000
L2 : 5
L3 : 0
OP : 0
QT : 1
RF : 0

FLAGS

HT : F F1 : F   F2 : F F3 : F
F4 : F F5 : F   F6 : F F7 : F

INSTRUCTION

33      RET
```

```
OUTPUT

Binary Searching for 5
Used array :
 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E
Found at :
 5
```

```
MEMORY

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 4E 6F 74 20 46 6F 75 6E 64 21 20 46 6F 75 6E 64 20
61 74 20 3A 20 0A 42 69 6E 61 72 79 20 53 65 61 72 63 68 69 6E 67 20 66 6F 72 20 35 55 73 65 64
20 61 72 72 61 79 20 3A 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

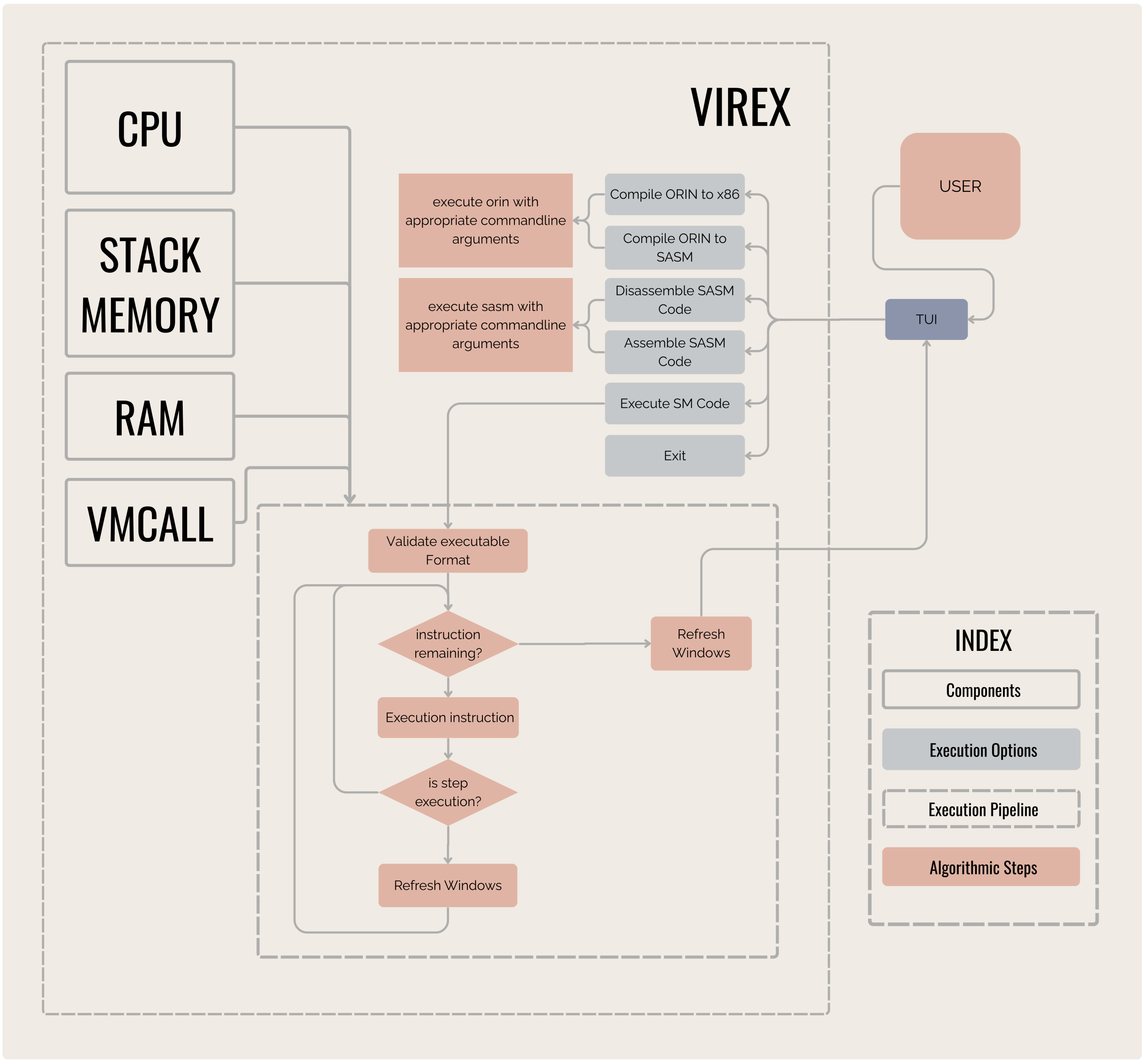
```
PROGRAM

65   | CALL   37
66   | CALL   0
67   | SHUTS
```

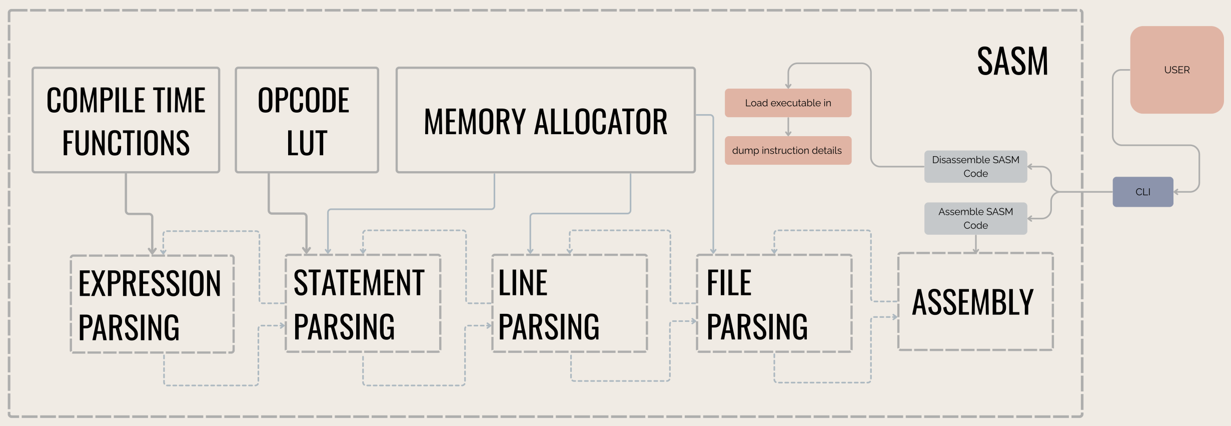
```
VIREX

VIRTUAL
EXECUTOR
```

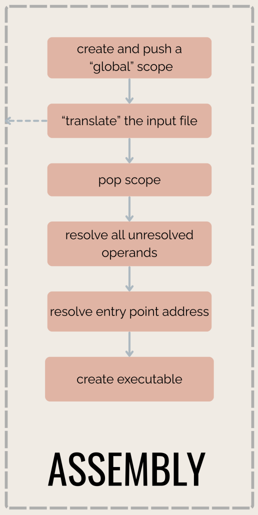
```
INPUT

Enter the name of the SM file : tmp.sm

 Debug Mode?
 0. No
 1. Yes
 2. Fast Debug
 Your choice : 2
```

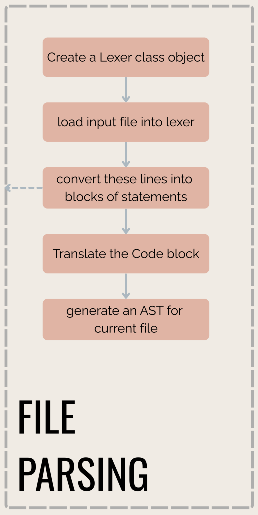
```
CREDITS

VIREX, SASM          : SOHAM METHA
AST visualizer       : SOHAM METHA
Syntax Highlighter   : SOHAM METHA
ORIN Compiler        : OMKAR JAGTAP
Core lib(Hashtable)  : OMKAR JAGTAP
Core libs(other)     : SOHAM METHA
```

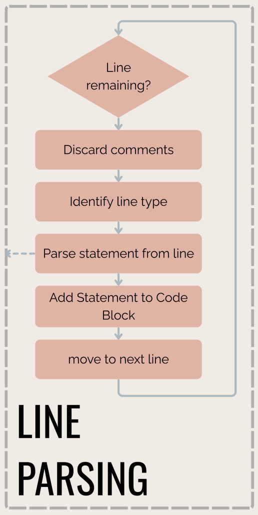
System Design and Architecture


VIREX
CPU
STACK MEMORY
RAM
VMCALL
USER
TUI
execute orin with appropriate commandline arguments
execute sasm with appropriate commandline arguments
Compile ORIN to x86
Compile ORIN to SASM
Disassemble SASM Code
Assemble SASM Code
Execute SM Code
Exit
Validate executable Format
instruction remaining?
Refresh Windows
Execution instruction
is step execution?
Refresh Windows
INDEX
Components
Execution Options
Execution Pipeline
Algorithmic Steps

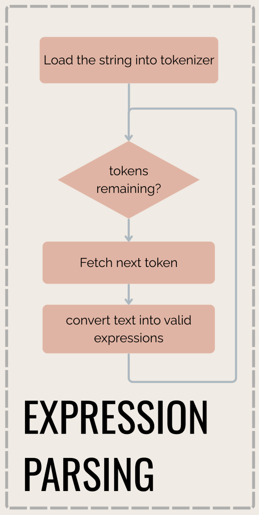


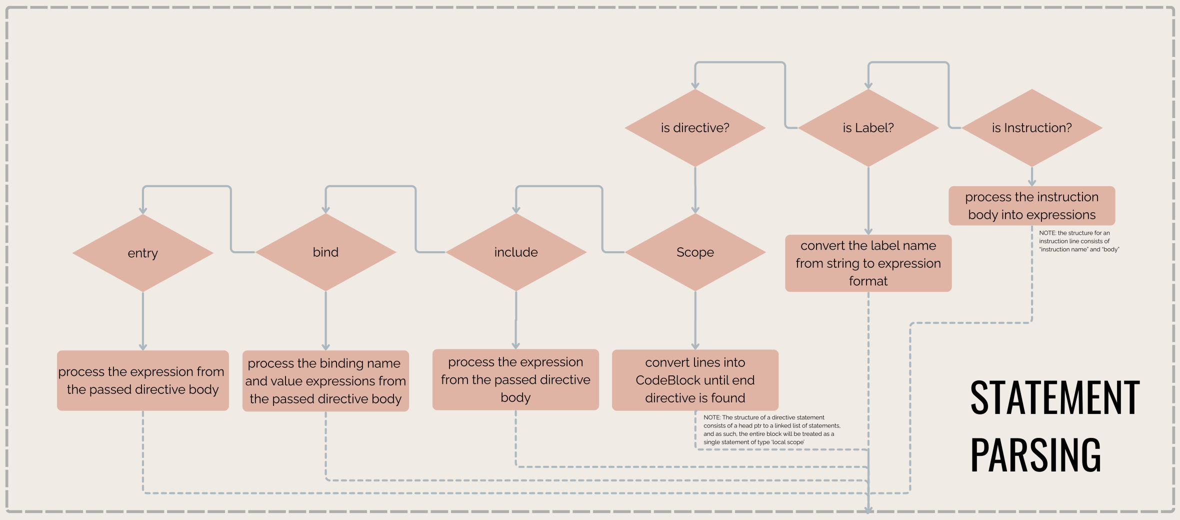

create and push a "global" scope
"translate" the input file
pop scope
resolve all unresolved operands
resolve entry point address
create executable
ASSEMBLY
(a) Assembly Pipeline


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
FILE PARSING
(b) File Parsing



(c) Line Parsing


Load the string into tokenizer
tokens remaining?
Fetch next token
convert text into valid expressions
EXPRESSION PARSING
(e) Expression 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'
STATEMENT PARSING
(d) Statement Parsing

Tech Stack

- **Programming Language:** C

- **Version Control:** Git
- **Build System:** GNU Make
- **AST VISUALIZER:** Graphviz

Maintainers

Tool	Maintainer
VIREX, SASM	Soham Metha
AST visualizer	Soham Metha
Syntax Highlighter	Soham Metha
ORIN Compiler	Omkar Jagtap
Core lib(Hashtable)	Omkar Jagtap
Core libs(other)	Soham Metha

References

- Tsoding
- Dr Birch
- Low Byte Productions
- Cobb Coding
