

VIREX : Virtual Execution Console

VIREX (VIRtual EXecuter) 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/Soham-Metha/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**

!!! info inline end ""

 For contribution guidelines and a roadmap, see [CONTRIBUTING.md]() (coming soon).

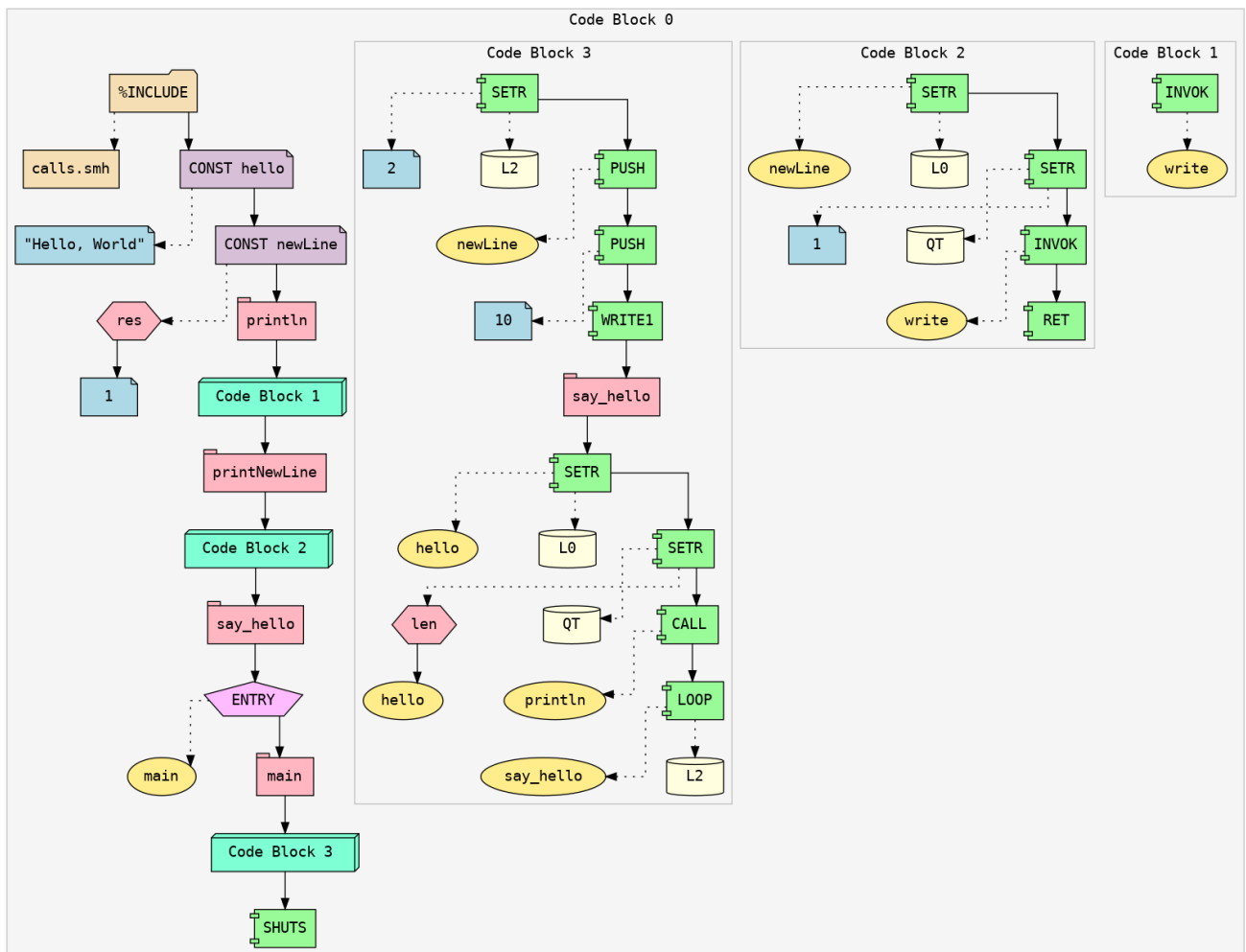
Examples

Syntax Highlighting:

```
helloWorld.sasm M X
extras > samplePrograms > helloWorld.sasm
You, 2 minutes ago | 1 author (You)
1 %include "calls.smh"
2
3 %bind hello "Hello, World" ; Compile-time Escape characters not yet supported, specify at runtime instead
4 %bind newLine res(1) ; reserves 1 byte in memory for the newline character
5
6 println:
7 %scope ; "write" is a integer const defined in 'calls.smh'
8 | INVOKE write ; INVOK is used to invoke a syscall(vsyscall?)
9 %end ; No RET here will lead to a fallthrough, printing a newline as well
10 printNewLine:
11 %scope
12 | SETR newLine [L0] ; SETR expects reference to a register(register ID), we can specify
13 | SETR 1 [QT] ; reference or value using ref([QT]) or val([QT]), default is ref()
14 | INVOKE write ; Will print QT(Quantity of) characters starting from location stored in L0
15 | RET
16 %end You, last month • Sasm Parser Rewrite done ...
17
18 say_hello: ; global 'say_hello'
19 %entry main: ; inline define label 'main' as the entry point of the program
20 %scope ; start local scope for main, optional, if not done, main runs in global scope
21 | SETR 2 [L2] ; SET Register 'L2' to 2
22 | PUSH newLine ; ptr to location
23 | PUSH 10 ; ASCII for newline
24 | WRITE1 ; Override 1 byte in memory, can use WRITE{1,2,4,8} depending on byte count
25 say_hello: ; local 'say_hello'
26 | SETR hello [L0] ; register L0 -> pointer to hello msg
27 | SETR len(hello) [QT] ; register QT -> length of hello msg
28 | CALL println
29 | LOOP say_hello [L2] ; Loop over label 'say_hello' - 'L2' times, P.S. zero inclusive
30 %end ; end local scope of main
31 SHUTS ; SHUT System
32
```

{ width="400" }

AST:



!!! info "Local/Global Scopes"

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 05 00 00 00 00 00 0D 00 00 00

SOH.AM.....

00000001

00 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00

.....

00000002

07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00

.....

00000003

00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 00 00 00 00

.....

00000004

06 00 00 00 00 00 00 00 01 00 00 00 00 00 00 10 00 00 00 00 00 00 00

.....

00000005

00 00 00 00 01 00 00 00 00 00 00 07 00 00 00 00 00 00 00 00 00 00 00

.....

00000006

00 00 00 00 00 00 00 21 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

.....!

00000007

00 00 00 00 00 00 00 00 00 00 06 00 00 00 00 00 02 00 00 00 00 00 00

.....

00000008

08 00 00 00 00 00 00 00 00 00 00 00 00 0A 00 00 00 00 00 00 00 00 00

.....

00000009

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0A 00 00 00 00 00 00

.....

0000000A

0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00 00 00

.....E..

0000000B

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

.....

0000000C

06 00 00 00 00 00 00 00 01 00 00 00 00 00 00 06 00 00 00 00 00 00 00

.....

0000000D

00 00 00 00 06 00 00 00 00 00 0C 00 00 00 00 00 10 00 00 00 00 00 00

.....

0000000E

00 00 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00

.....

0000000F

00 00 00 00 00 00 00 00 00 00 09 00 00 00 00 09 00 00 00 00 00 00

.....

00000010

08 00 00 00 00 00 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00 00

.....

00000011

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 48 65 6C 6C 6F 2C 20

.....Hello,

00000012

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

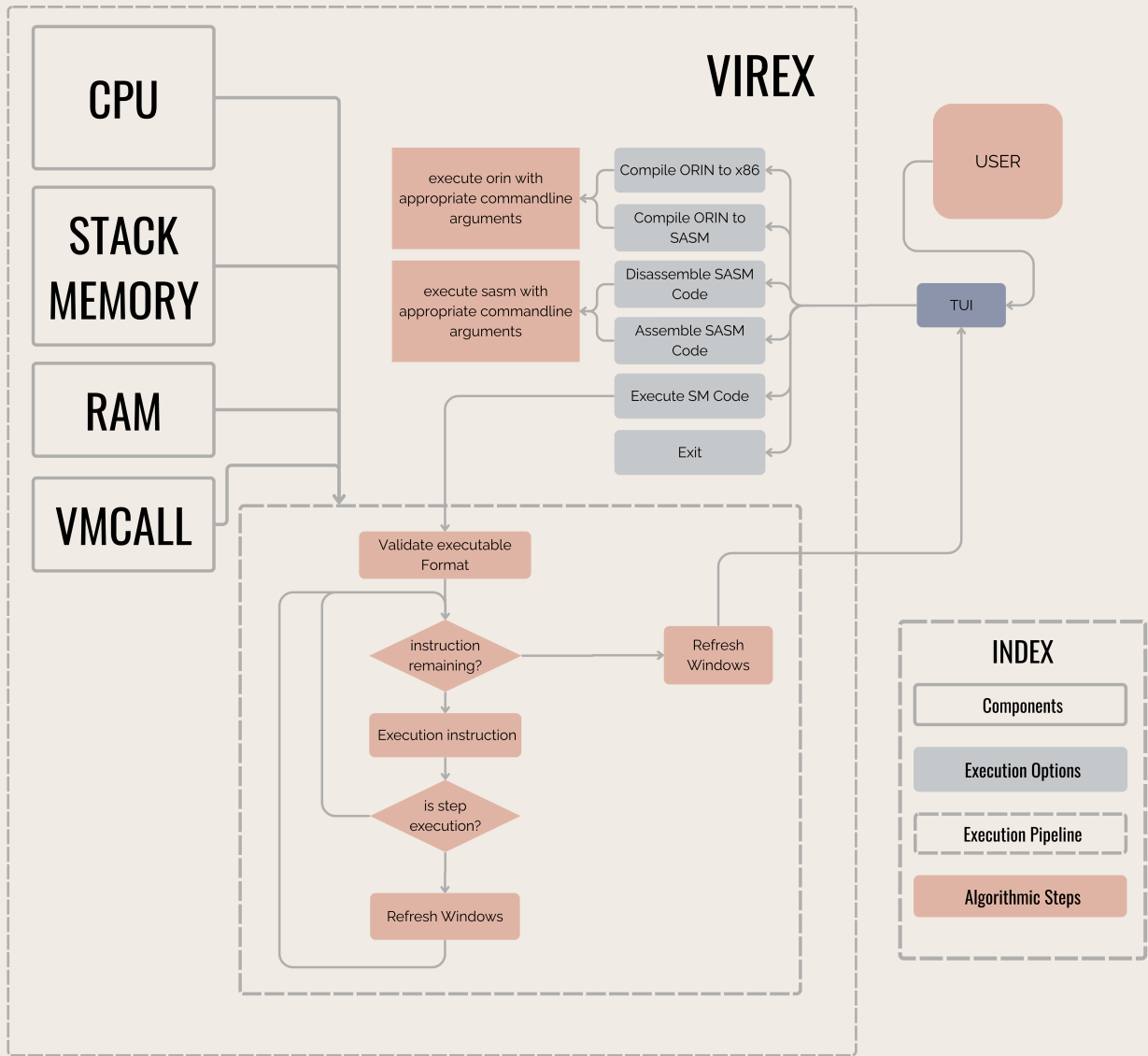
Offset: 0x1f1 / 0x1fc

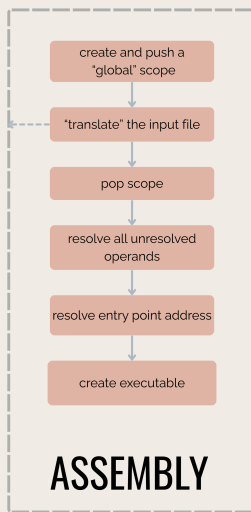
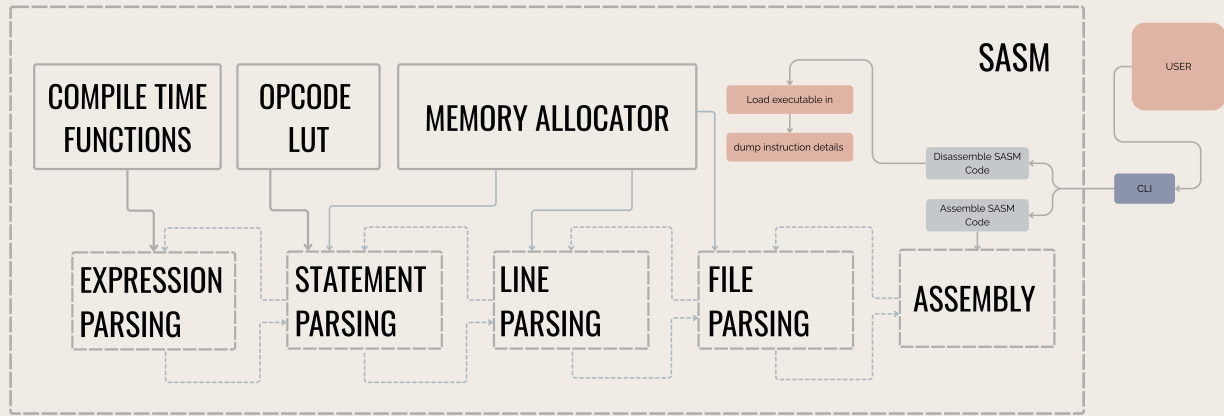
Selection: 0x1f1 to 0x1f5 (0x5 bytes)

INS

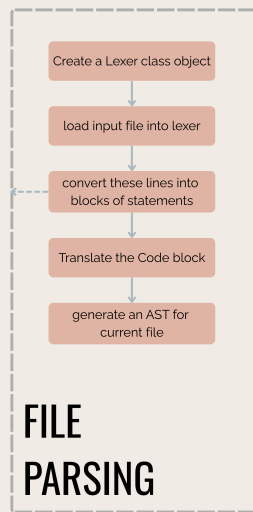
The image shows a Windows desktop with a 'Virtual Executable' application window. The application has a dark theme and is divided into several panels:

- REGISTERS:** Displays CPU registers (HI, PI, P2, P3, JS, NA, TI, T1, L0, L1, L2, L3, OP, QT, RT) and flags (HT, F1-F4, F5-F7).
- OUTPUT:** Shows binary search results for a binary array, including the array content and the found value (5).
- MEMORY:** Displays a memory dump with addresses and hex values.
- PROGRAM:** Shows a list of instructions (CALL, SHUTS) with addresses (65, 66, 67).
- VIREX:** Displays 'VIRTUAL EXECUTOR' in large, stylized red letters.
- INPUT:** Prompts the user to enter the name of the SH file (e.g., tap.sm).
- CREDITS:** Lists the components used in the application, including VIREX, ASM, AST visualizer, Syntax Highlighter, ORIN Compiler, Core lib(ashtable), and Core lib(sother).

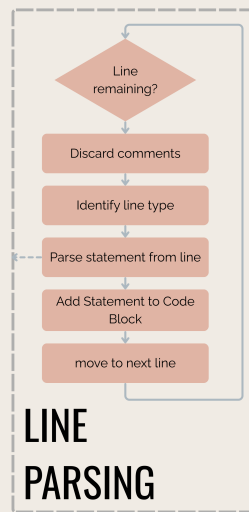




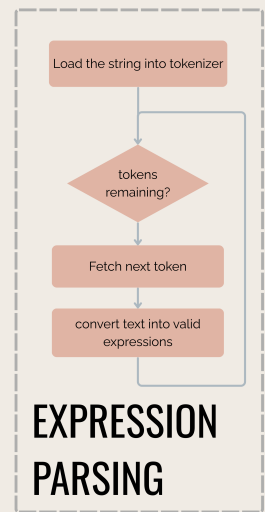
(a) Assembly Pipeline



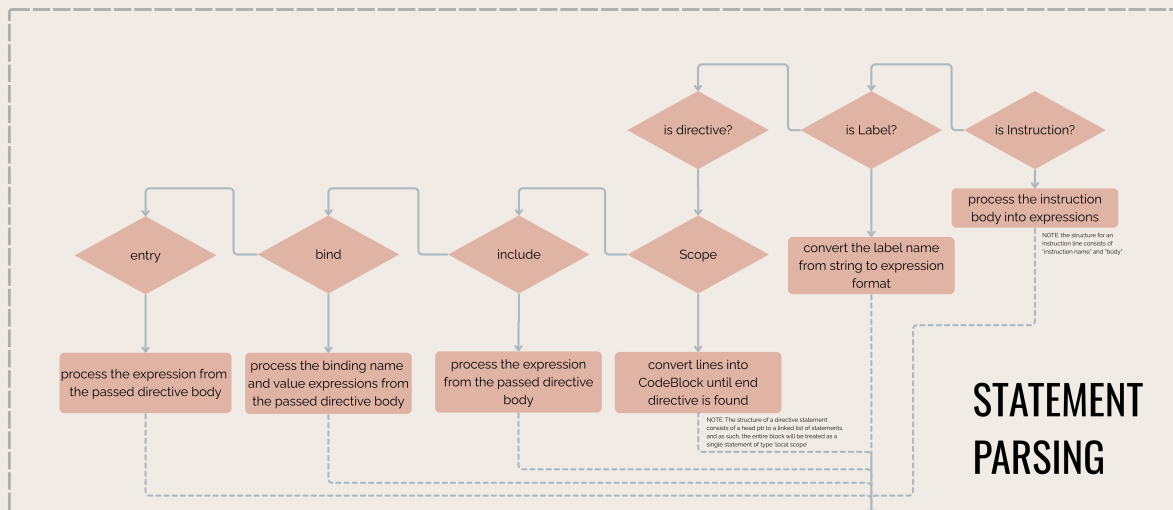
(b) File Parsing



(c) Line Parsing



(e) Expression 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](#)
-