

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.



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
```




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

Binary Executable:

helloWorld.sm X

00000000	53	4F	48	00	41	4D	00	00	0E	00	00	00	00	00	00	00	05	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	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	06	00	00	00
00000054	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
0000008c	00	00	00	00	01	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	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	06	00	00	00	00	02	00	00	00	00	00	00	00	00	00	00
000000e0	08	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	0A	00	00	00	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	45	00	00	00	00	00	00	00
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
00000150	06	00	00	00	00	00	00	00	01	00	00	00	00	00	06	00	00	00	00	00	00	00	00	00	00	00	00
0000016c	00	00	00	00	06	00	00	00	00	00	00	0C	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00
00000188	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	09	00	00	00	00	00	09	00	00	00	00	00	00	00	00	00	00
000001c0	08	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	00
000001dc	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	48	65	6C	6C	6F	2C	20	00	00	00
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

R0 : 0 H1 : 0
R8 : 4 P1 : 6
P2 : 0 P3 : 0
J5 : 0 KC : 0
NX : 67
I0 : 0
I1 : 0
L0 : 37
L1 : 0.000000
L2 : 5
L3 : 0
OP : 0
QT : 1
RF : 0

FLGOS

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 4F 74 20 46 6F 75 6E 64 21 20 46 6F 75 6E 64 20
61 74 20 38 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 77 20 38 20 00
00
00
00
00 00

< PROGRAM >

65 CALL 3
66 CALL 0
67 SHUTS

< VIREX >

VIRTUAL EXECUTOR

< INPUT >

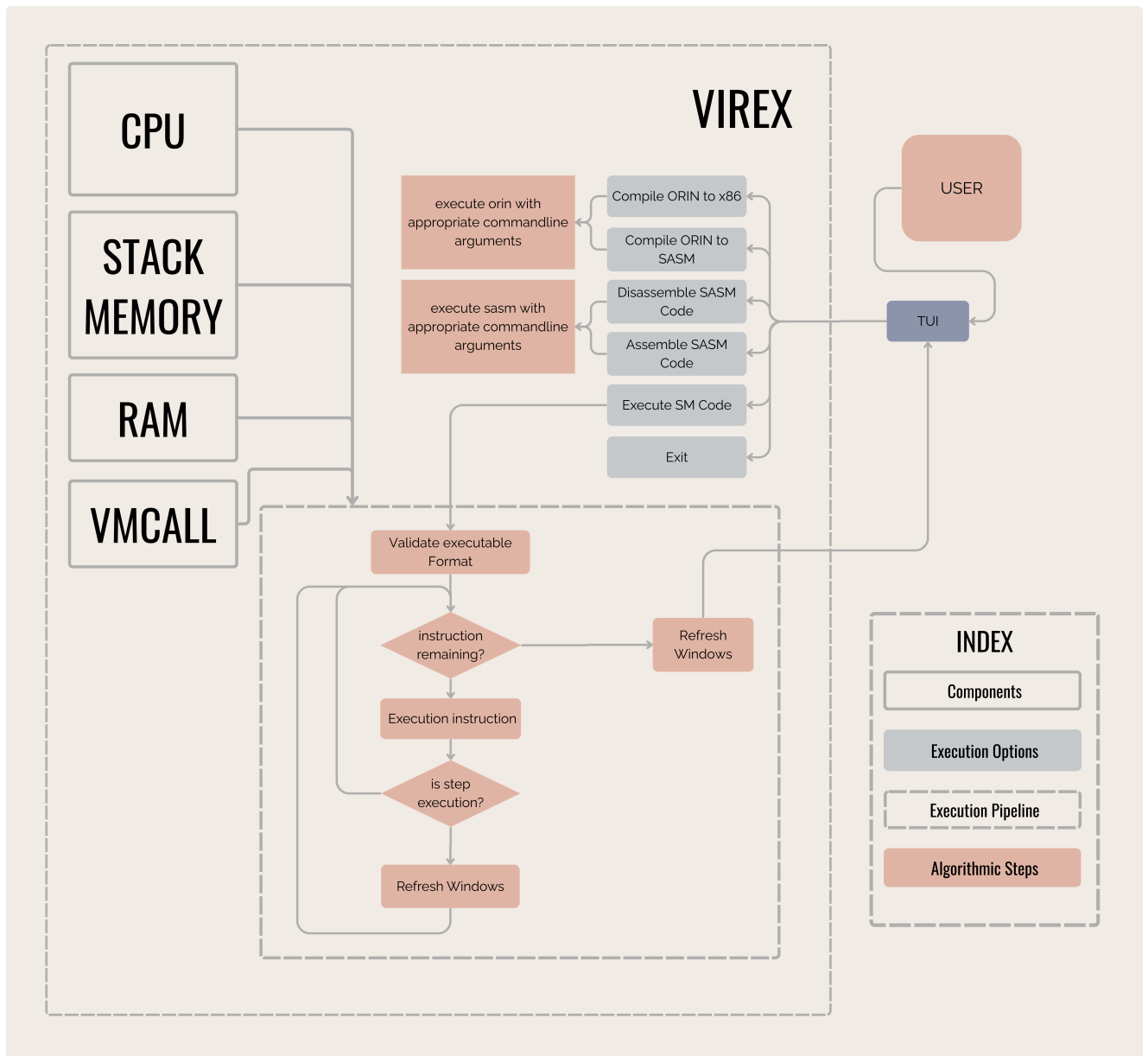
Enter the name of the SH file : tap.sm

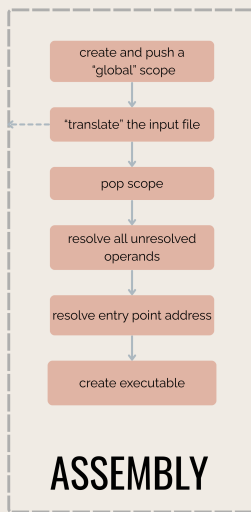
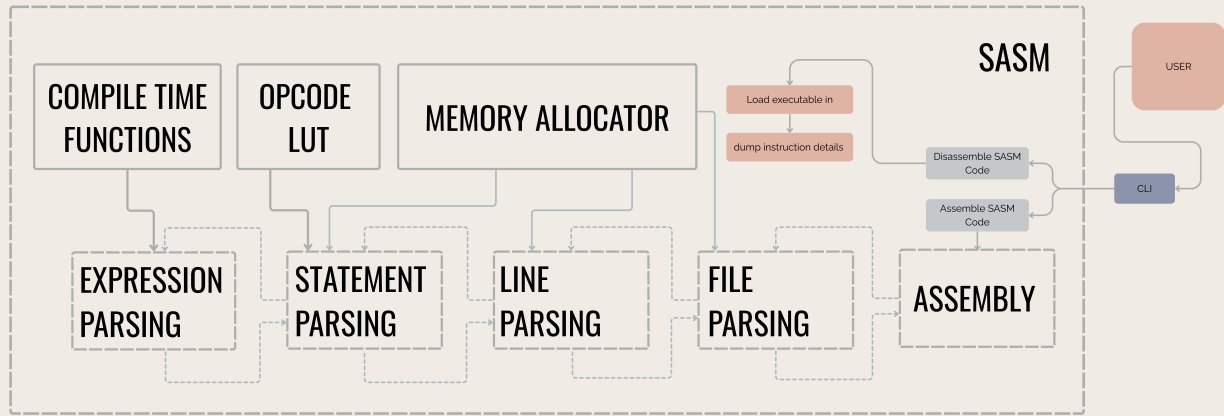
Debug Mode?
0. No
1. Yes
2. Fast Debug
Your choice : 2

< CREDITS >

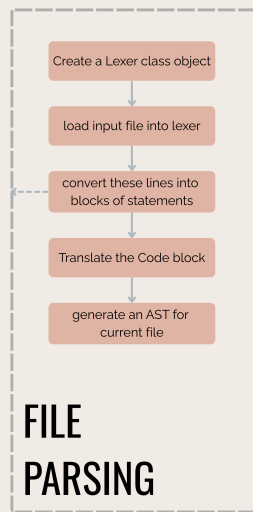
VIREX, SASM : SOHAM METHA
AST Visualizer : SOHAM METHA
Syntax Highlighter : SOHAM METHA
ODIN Compiler : ONKAR JAGTAP
Core lib(NashTable) : ONKAR JAGTAP
Core libs(other) : SOHAM METHA

System Design and Architecture

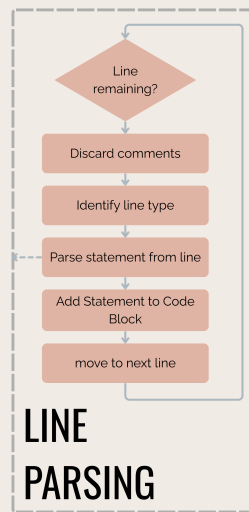




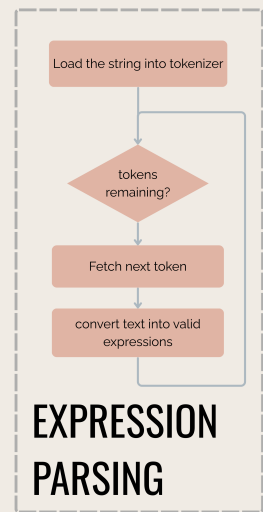
(a) Assembly Pipeline



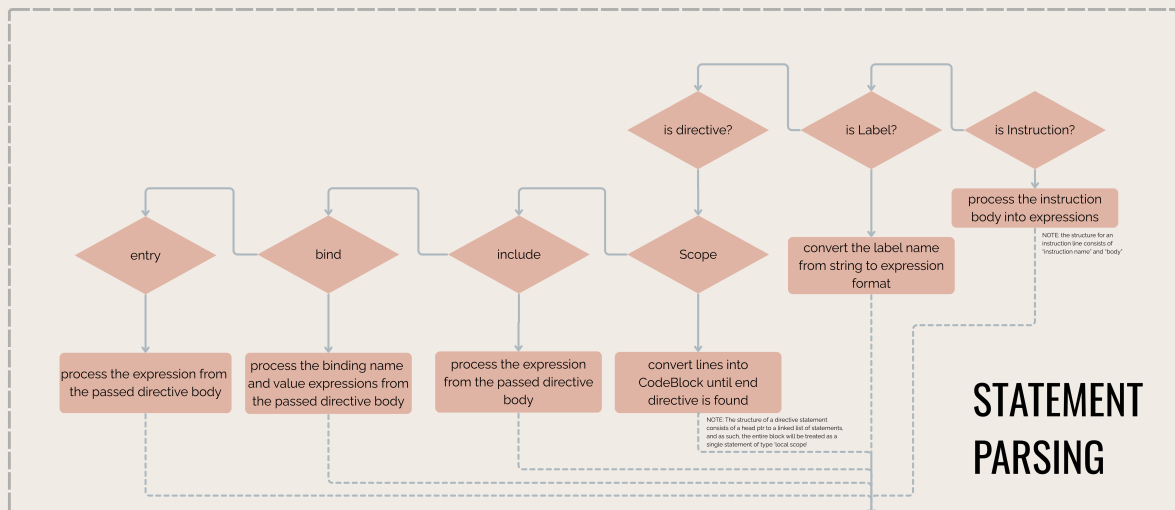
(b) File Parsing



(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](#)
-