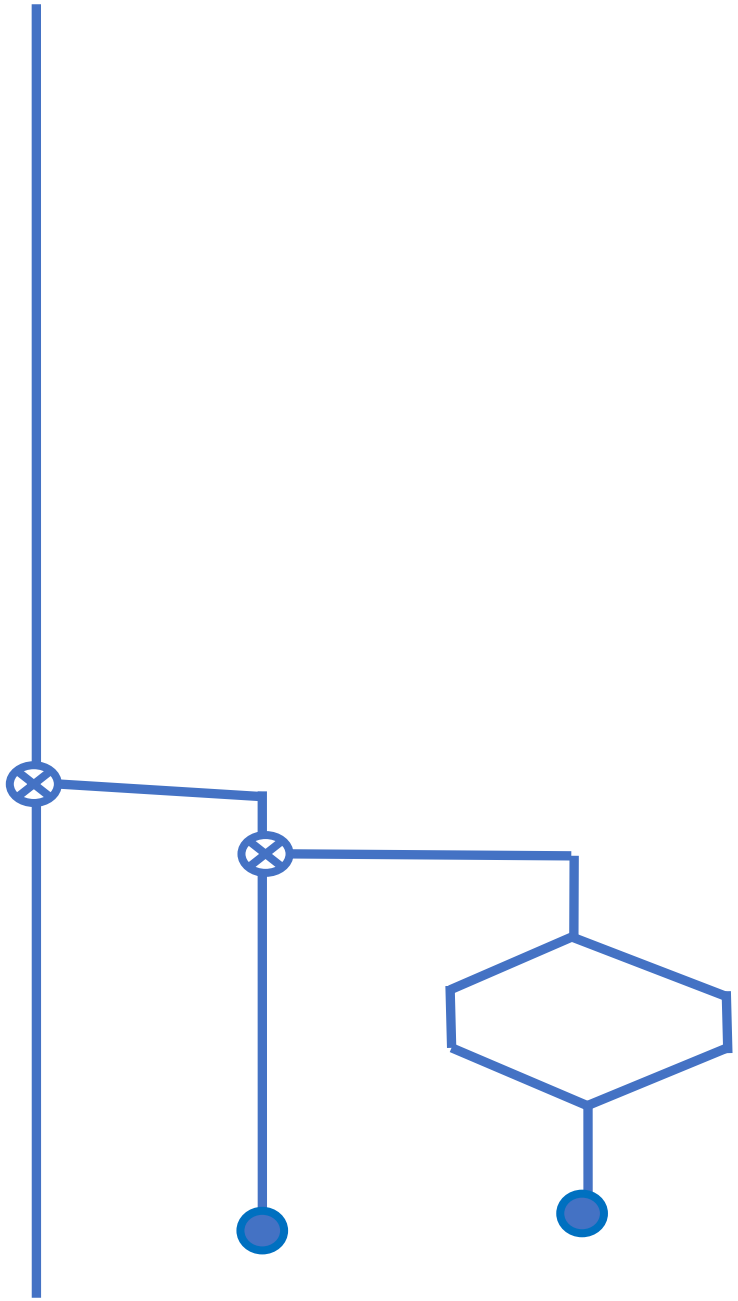

Assembler Op-Codes For A Matrix Virtual Machine

SVFIG

Nov. 15, 2025

Bill Ragsdale

Introduction

Imagine adding matrix capability to Forth in 45 lines of code.

Can be with done with sixteen primatives equivalent to machine opcodes.

Supported by 1,500 lines of OOP source code.

I'll show the clarity and brevity of adding abilities.

What Can 45 Lines Of Code Do?

Define matrix objects.

Initialize by cell size and number of rows and columns.

Supply heap memory allocation.

Enter dummy data for early testing.

List a matrix.

Copy matrix to matrix.

Read and write a cell.

Copy cells matrix to matrix.

Increment a cell.

Test for row and column range errors.

Then We Efficiently Write Application Tools

Populate a matrix in a wide range of formats.

Read from and write to files.

A wide variety of copy and transformation words.

Report formats to match applications.

High level matrix math.

The Virtual Machine

Two defining words (similar to CODE)

Six registers and a flag bit.

Sixteen opcodes.

That's it. In 45 lines of code.

Defining Words

:CLASS Creates a matrix definer:

like CREATE DOES>

:Matrix{ Creates a matrix, a container,

uninitialized like ALLOT or VARIABLE.

The Registers

#rows

#cols

bytes/cell

HeapLocation

size

row-span

protection (1 bit)

Matrix Management Opcodes

Setup:	Set bytes/cell R C and allocate heap.
Params:	Read bytes/cell R C.
MlocSize:	Report heap address and size.
Release:	Release heap allocation.
GetRspan:	Number of bytes in a row.
AssignProtection:	Turn RC range test on & off.

Cell Based Opcodes

RCaddr:	R C memory address of a cell
RCget:	R C reads a float from a cell.
RCput:	R C writes a float to cell.
RC+put:	R C increments cell by float.
Test:	R C verify values within matrix range.

Matrix Support Opcodes

Name:	Counted string of matrix name.
Clear:	Clear matrix to zeros.
Fill:	Fill matrix by row/cell number.
List:	Display a matrix.

A Peek At An Op-code's Source Code

```
:M List:
 Name: self    count type cr
 #rows 0 do
     #cols 0 do
        j i RCget: self  10 2 f.r
     loop cr loop  ;M
```

Define And Initialize A Matrix

```
:Matrix{ x{      b/float 4 3 Setup: x{
 Params: x{    rot . swap . .  \ see 8  4  3
   List: x{    \ and see
     x{
     0.00      0.00      0.00
     0.00      0.00      0.00
     0.00      0.00      0.00
     0.00      0.00      0.00
```

Write And Read A Cell

```
56.78e0  0 0 RCput: x{
         0 0 RCget: x{   f.
And see:     56.7800  ok
 X{ 56.78      0.00      0.00
      0.00     0.00      0.00
      0.00     0.00      0.00
      0.00     0.00      0.00
```

Increment A Cell

```
1000e0 0 0 RC+put: x{
       0 0 RCget: x{  f.
And see:   1056.78
X{   1056.78     0.00     0.00
        0.00     0.00     0.00
        0.00     0.00     0.00
        0.00     0.00     0.00
```

Fill A Matrix For Testing

```
Fill: x{  List: x{

 X{

      0.00      1.00      2.00

     10.00     11.00     12.00

     20.00     21.00     22.00

     30.00     31.00     32.00
```

Clear And Re-dimension A Matrix

```
b/float 8 2 Setup: x{  Fill: x{
Params: x{  rot . swap . .  \ see: 8 8 2
 X{  0.00   1.00
    10.00  11.00
    20.00  21.00
    30.00  31.00
    40.00  41.00
    50.00  51.00
```

Clone And Copy A Matrix

```
:Matrix{ y{   Params: x{  Setup: y{
 MlocSize: x{ drop   MlocSize: y{ cmove
x{ y{ }Dual
 X{                Y{
   0.00   1.00  |    0.00   1.00
  10.00  11.00  |   10.00  11.00
  20.00  21.00  |   20.00  21.00
  30.00  31.00  |   30.00  31.00
  40.00  41.00  |   40.00  41.00
  50.00  51.00  |   50.00  51.00
```

Release Allocation And Forget

```
Release: x{  Release: y{  x{ }Forget
```

High Level: Literals

```
\ Load from literal values
3 3 Matrix{  z{
z{ {[ 1 2 3 | 3.14  2.17  0 | -1 -1 -1 ]}
z{ }list
 z{  1.00     2.00     3.00
     3.14     2.17     0.00
    -1.00    -1.00    -1.00
```

High Level: Eye Matrix

```
: }Eye    ( x{ --- )
  dup }RC <> abort" Not square"
  dup }clear
  dup }R 0  do dup i i 1.0e }}! loop drop ;

z{ }list
 z{  1.00     0.00     0.00     0.00
     0.00     1.00     0.00     0.00
     0.00     0.00     1.00     0.00
     0.00     0.00     0.00     1.00
```

Conclusion

These assembly opcodes are a mix of infix and prefix notations due to OOP.

Combining them into high-level Forth words gets us back to post-fix Forth.

Would it be practical to have a matrix processor in silicon? Just asking.

Thanks to Andrew McKewan for including object-oriented programming in Win32Forth.