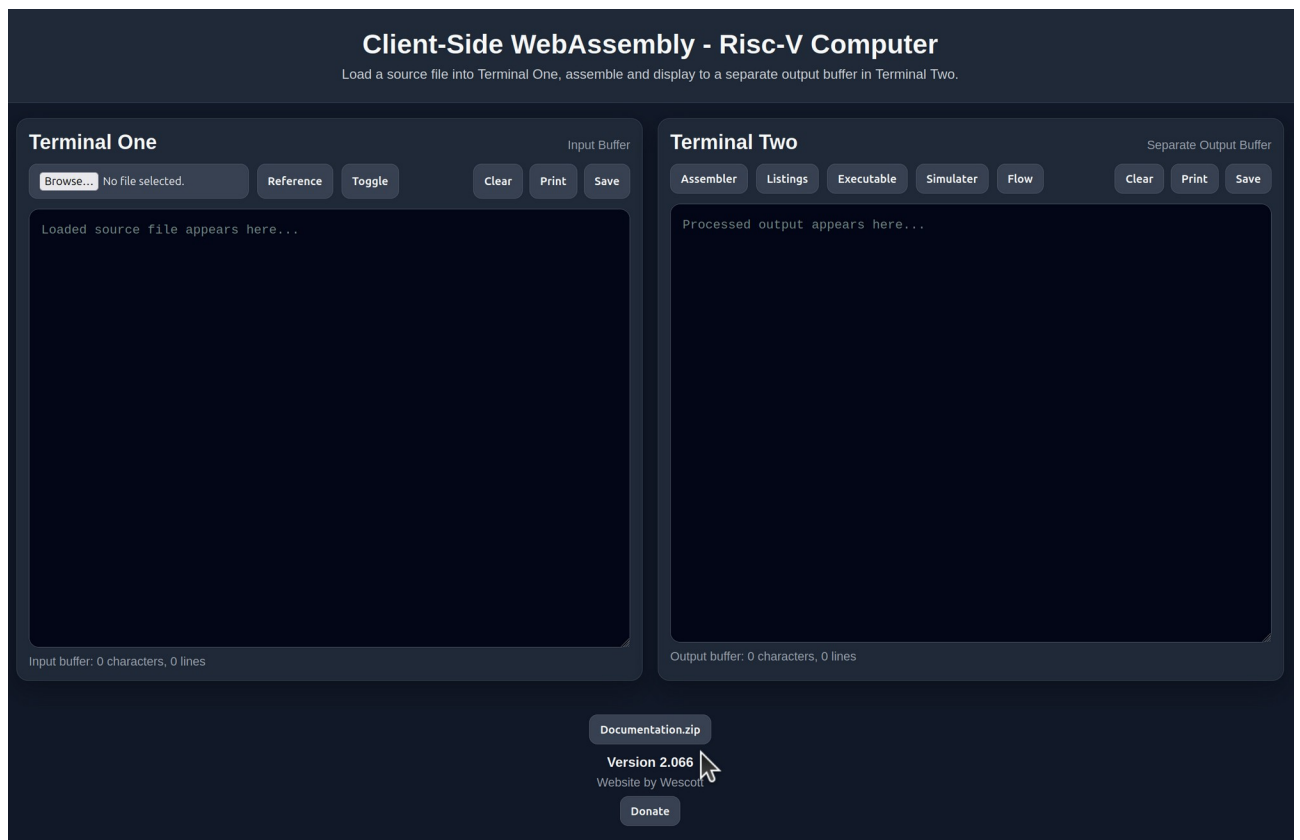


Getting Started;



To download the support package, click on the Documentation.zip button. The file will be downloaded to your download directory.

Files:

Flowchart.txt

GettingStarted.pdf

Demo Programs:

/Demo Programs/auipc_test.txt

/Demo Programs/BubbleSort.txt

/Demo Programs/CallAndReturn.txt

/Demo Programs/DivideBySubtract.txt

/Demo Programs/Division.txt

/Demo Programs/factorial.txt

/Demo Programs/Fibonacci_1.txt

/Demo Programs/GreatestCommonDivisor.txt

/Demo Programs/halfword.txt

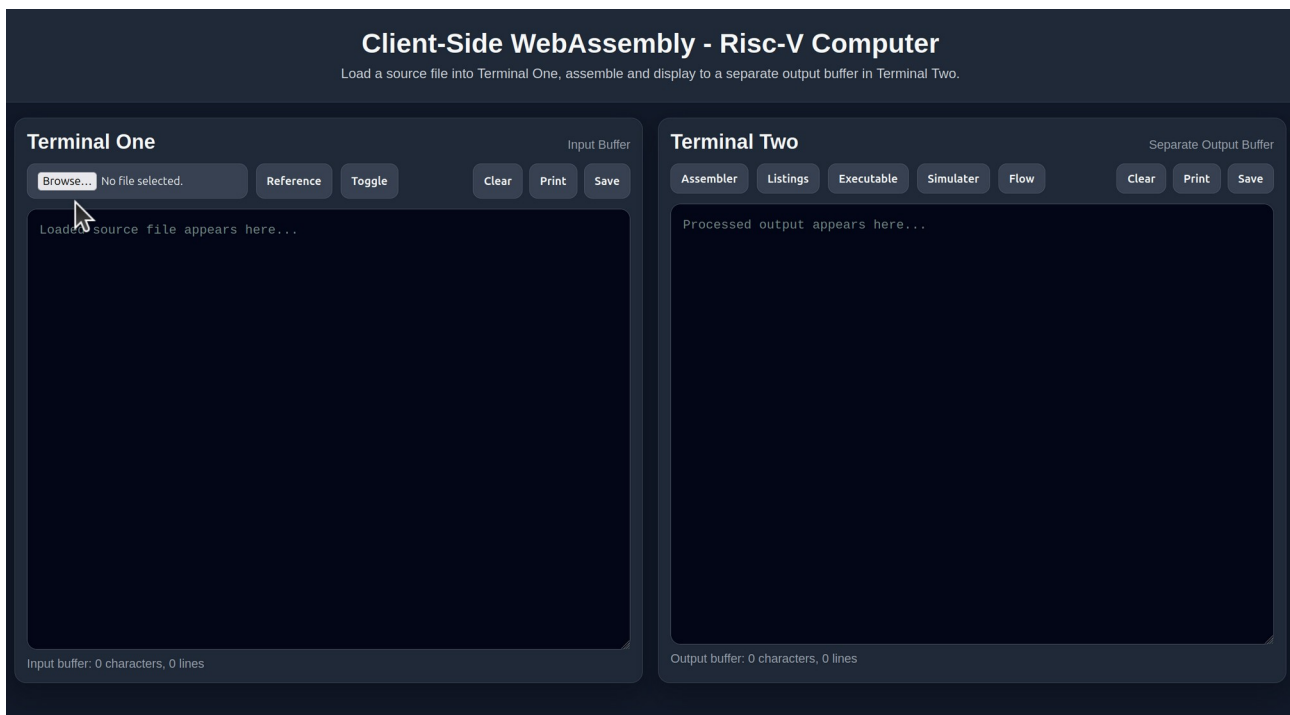
/Demo Programs/jmp_test.txt

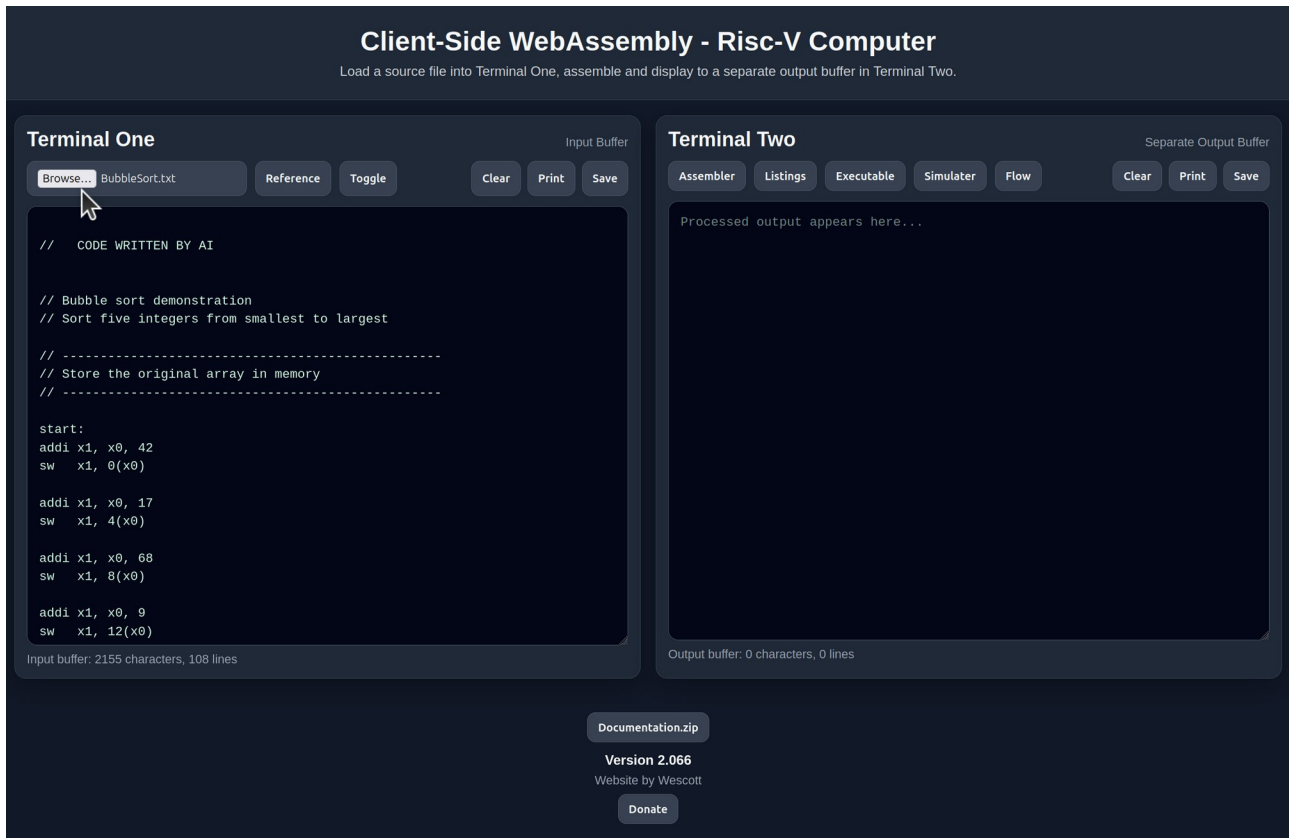
/Demo Programs/Largest.txt

/Demo Programs/lh-test.txt
/Demo Programs/memory messages.txt
/Demo Programs/Primes.txt
/Demo Programs/RecursiveFibonacci.txt
/Demo Programs/sh-tst.txt
/Demo Programs/Simple Calculator.txt
/Demo Programs/SimpleMultiply.txt
/Demo Programs/SumOfSquares.txt
/Demo Programs/sw-lw-test.txt

Strating the process:

Browse to your file and load it





Now your source is loaded

```
// CODE WRITTEN BY AI

// Bubble sort demonstration
// Sort five integers from smallest to largest

// -----
// Store the original array in memory
// -----

start:
addi x1, x0, 42
sw x1, 0(x0)

addi x1, x0, 17
sw x1, 4(x0)

addi x1, x0, 68
sw x1, 8(x0)

addi x1, x0, 9
```

```

sw  x1, 12(x0)

addi x1, x0, 31
sw  x1, 16(x0)

// -----
// Display the original array
// -----

cout << "BUBBLE SORT DEMONSTRATION" << endl;
cout << endl;
cout << "Original values:" << endl;

lw  x1, 0(x0)
cout << x1 << endl;

lw  x1, 4(x0)
cout << x1 << endl;

lw  x1, 8(x0)
cout << x1 << endl;

lw  x1, 12(x0)
cout << x1 << endl;

lw  x1, 16(x0)
cout << x1 << endl;

// -----
// Bubble sort
// -----

// x10 = number of outer passes
// x11 = number of comparisons in current pass
// x12 = current memory address
// x13 = left value
// x14 = right value

addi x10, x0, 4    // Four sorting passes

outer_loop:
add  x11, x10, x0  // Comparisons needed this pass
addi x12, x0, 0    // Begin at memory address 0

inner_loop:
lw  x13, 0(x12)    // Load left value
lw  x14, 4(x12)    // Load right value

bge  x14, x13, no_swap // No swap if right >= left

```

```
sw x14, 0(x12) // Store smaller value on left
sw x13, 4(x12) // Store larger value on right
```

```
no_swap:
addi x12, x12, 4 // Move to next array position
addi x11, x11, -1 // One comparison completed
bne x11, x0, inner_loop
```

```
addi x10, x10, -1 // One pass completed
bne x10, x0, outer_loop
```

```
// -----
// Display the sorted array
// -----
```

```
cout << endl;
cout << "Sorted values:" << endl;
```

```
lw x1, 0(x0)
cout << x1 << endl;
```

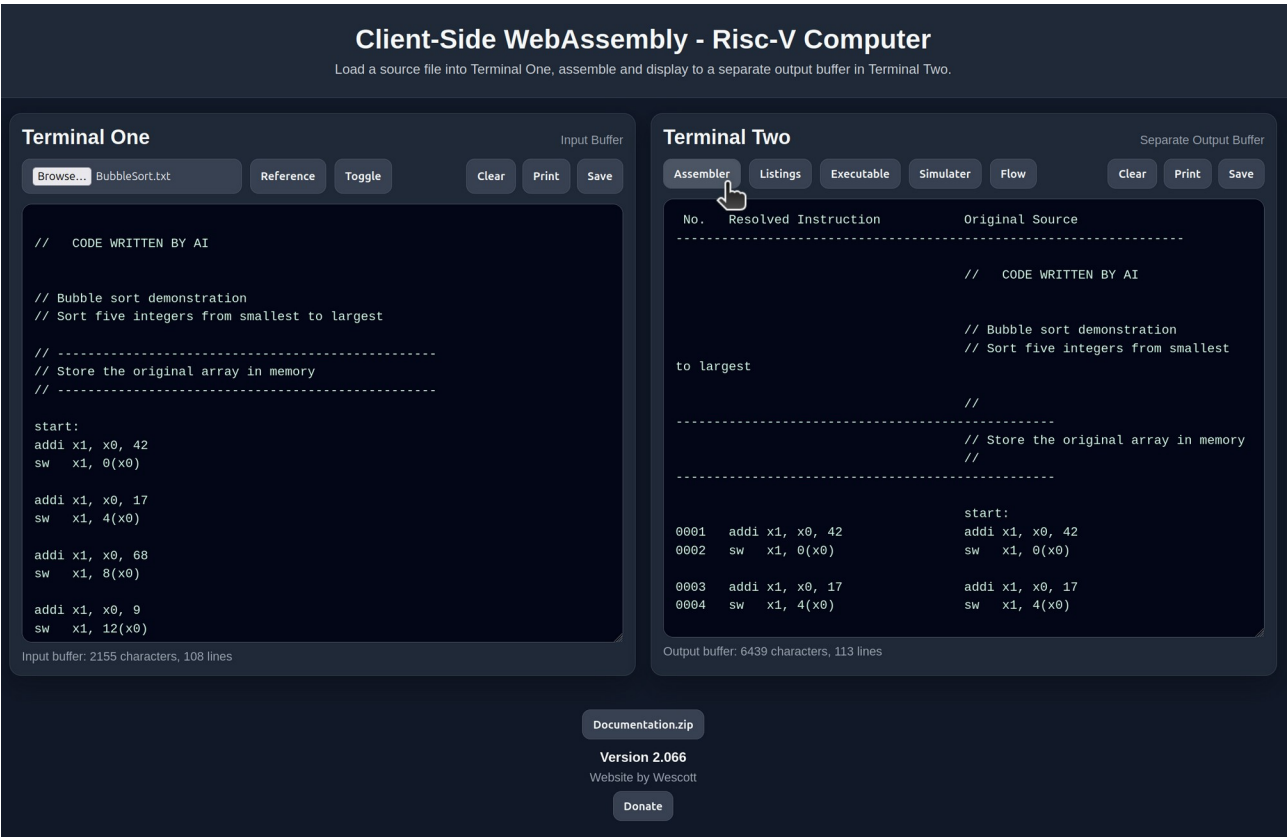
```
lw x1, 4(x0)
cout << x1 << endl;
```

```
lw x1, 8(x0)
cout << x1 << endl;
```

```
lw x1, 12(x0)
cout << x1 << endl;
```

```
lw x1, 16(x0)
cout << x1 << endl;
```

```
cout << endl;
cout << "SORT COMPLETE" << endl;
```



Press Assembler button to obtain final program listing:

No.	Resolved Instruction	Original Source
		// CODE WRITTEN BY AI
		// Bubble sort demonstration
		// Sort five integers from smallest to largest
		// -----
		// Store the original array in memory
		// -----
		start:
0001	addi x1, x0, 42	addi x1, x0, 42
0002	sw x1, 0(x0)	sw x1, 0(x0)
0003	addi x1, x0, 17	addi x1, x0, 17
0004	sw x1, 4(x0)	sw x1, 4(x0)
0005	addi x1, x0, 68	addi x1, x0, 68

0006	sw x1, 8(x0)	sw x1, 8(x0)
0007	addi x1, x0, 9	addi x1, x0, 9
0008	sw x1, 12(x0)	sw x1, 12(x0)
0009	addi x1, x0, 31	addi x1, x0, 31
0010	sw x1, 16(x0)	sw x1, 16(x0)

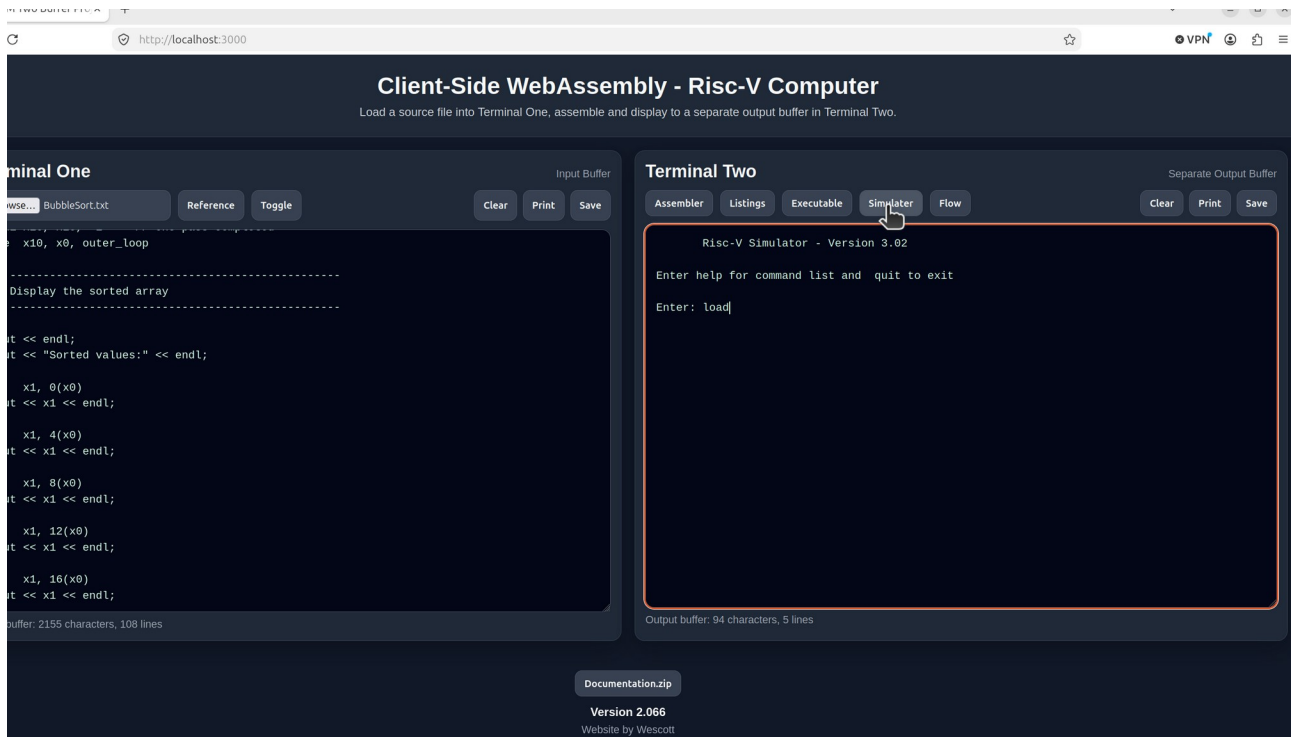
```
// -----
// Display the original array
// -----
```

...

0035	addi x10, x10, -1	addi x10, x10, -1 // One pass completed
0036	bne x10, x0, -44	bne x10, x0, outer_loop

```
// -----
// Display the sorted array
// -----
```

0037	addi x0, x31, 8	cout << endl;
0038	addi x0, x31, 9	cout << "Sorted values:" << endl;
0039	lw x1, 0(x0)	lw x1, 0(x0)
0040	addi x0, x31, 10	cout << x1 << endl;



Select Simulator and enter “load” command

Risc-V Simulator - Version 3.02

Enter help for command list and quit to exit

Enter: load

Load Instructions

```
1 addi x1, x0, 42
2 sw  x1, 0(x0)
3 addi x1, x0, 17
4 sw  x1, 4(x0)
5 addi x1, x0, 68
6 sw  x1, 8(x0)
7 addi x1, x0, 9
8 sw  x1, 12(x0)
9 addi x1, x0, 31
10 sw  x1, 16(x0)
11 addi x0, x31, 0
```



```
12 addi x0, x31, 1
13 addi x0, x31, 2
14 lw x1, 0(x0)
15 addi x0, x31, 3
16 lw x1, 4(x0)
17 addi x0, x31, 4
18 lw x1, 8(x0)
19 addi x0, x31, 5
20 lw x1, 12(x0)
21 addi x0, x31, 6
22 lw x1, 16(x0)
23 addi x0, x31, 7
24 addi x10, x0, 4
25 add x11, x10, x0
26 addi x12, x0, 0
27 lw x13, 0(x12)
28 lw x14, 4(x12)
29 bge x14, x13, 12
30 sw x14, 0(x12)
31 sw x13, 4(x12)
32 addi x12, x12, 4
33 addi x11, x11, -1
34 bne x11, x0, -28
35 addi x10, x10, -1
36 bne x10, x0, -44
37 addi x0, x31, 8
38 addi x0, x31, 9
39 lw x1, 0(x0)
40 addi x0, x31, 10
41 lw x1, 4(x0)
42 addi x0, x31, 11
```

```
43 lw x1, 8(x0)
44 addi x0, x31, 12
45 lw x1, 12(x0)
46 addi x0, x31, 13
47 lw x1, 16(x0)
48 addi x0, x31, 14
49 addi x0, x31, 15
50 addi x0, x31, 16
```

Loading Success

You now can run the program with command “run”

Terminal TwoSeparate Output Buffer

AssemblerListingsExecutableSimulatorFlowClearPrintSave

```
BUBBLE SORT DEMONSTRATION

Original values:
42
17
68
9
31

Sorted values:
9
17
31
42
68

SORT COMPLETE
Code Completed

Enter: |
```

Output buffer: 170 characters, 24 lines

BUBBLE SORT DEMONSTRATION

Original values:

42

17

68

9

31

Sorted values:

9

17

31

42

68

SORT COMPLETE Code Completed

Program can also be stepped through. This allow you to view regs and memory as you step

Terminal Two

Separate Output Buffer

Assembler

Listings

Executable

Simulator

Flow

Clear

Print

Save

RISC-V STEP DISPLAY

BYTE MEMORY

+0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F

0x0000000000 2A 00 00 00 11 00 00 00 1F 00 00 00 2A 00 00 00

INTEGER MEMORY

+0 +1 +2 +3

0x0000000000 0x0000002A 0x00000011 0x0000001F 0x0000002A

x0 (zero) 0

x1 (ra) 42

x2 (sp) 0

x3 (gp) 0

x4 (tp) 0

x5 (t0) 0

x6 (t1) 0

x7 (t2) 0

x8 (s0/fp) 0

x9 (s1) 0

x10 (a0) 0

x11 (a1) 0

x12 (a2) 4

x13 (a3) 9

x14 (a4) 17

x15 (a5) 0

x16 (a6) 0

x17 (a7) 0

x18 (s2) 0

x19 (s3) 0

x20 (s4) 0

x21 (s5) 0

x22 (s6) 0

x23 (s7) 0

x24 (s8) 0

x25 (s9) 0

x26 (s10) 0

x27 (s11) 0

x28 (t3) 0

x29 (t4) 0

x30 (t5) 0

x31 (t6) 0

.

000001 [1] addi x1, x0, 42

000002 [2] sw x1, 0(x0)

Commands: step | step num |run | memory addr

Enter: