

KONZEPTE SYSTEMNAHER PROGRAMMIERUNG

Technische Hochschule Mittelhessen

Andre Rein

– Unterprogrammaufrufe und Rücksprünge –

UNTERPROGRAMM-AUFRUFE UND RÜCKSPRÜNGE

Beispiel eines Unterprogrammaufrufs in C:

```
int main(int argc, char *argv[]) {
    printf("Hallo"); ①
    printf("Welt\n"); ②
    return 0;
}
```

- 1 Springe zur Funktion `printf()`, führe alle Instruktionen von `printf()` aus und kehre **anschließend** wieder zur `main`-Funktion zurück um
- 2 die zweite `printf()`-Funktion auszuführen.



Wir haben die 2 bedingten Sprünge (`brf`, `brt`) und einen unbedingten Sprung (`jmp`) besprochen. Bei der Verwendung geht allerdings die Information verloren, **von welcher Position** aus wir gesprungen sind

UNTERPROGRAMM-AUFRUFE UND RÜCKSPRÜNGE

Einführung neuer Instruktionen für Unterprogrammaufrufe:

- `call <n> → ... -> ... ra` – legt eine **Rücksprungadresse** `ra` auf den Stack und setzt den Programmzähler auf `<n>` ($PC=n$)
- `ret → ... ra -> ...` – nimmt die Rücksprungadresse `ra` vom Stack und setzt den Programmzähler auf den Wert von `ra` ($PC=ra$)



Insgesamt gibt es 4 Varianten von Funktionsaufrufen: Mit und ohne **Funktionsparameter** jeweils mit und ohne **Rückgabewerte**.

FUNKTIONSAUFRUFE OHNE ARGUMENTE UND RÜCKGABEWERTE

- Notation: **Caller** Aufrufende Funktion
- Notation: **Callee** Aufgerufene Funktion
- Funktionslabel: Ähnlich zum normalen Label *eine Adresse im Programm*

Caller

```
call Funktionslabel; ①  
add; // beliebige nachfolgende  
// Instruktion ②
```

Callee

```
Funktionslabel:  
    asf <numLocals> // numLocals bezeichnet die Anzahl der  
                      // benötigten lokalen Variablen der Funktion  
  
    <body>           // unbestimmte Anzahl an Instruktionen  
    rsf              // zurücksetzen des aktuellen Stackframes ③  
    ret              // zurückkehren zum Aufrufer (Caller!) ④
```

1 Legt die Adresse
von 2 auf den
Stack

3 Auf dem Stack liegt nun die Rücksprungadresse ra
(2) als oberster Wert!

4 ret nimmt die Rücksprungadresse und springt
zurück in den Caller zur Position 2.

FUNKTIONSAUFRUF: BEISPIEL

```
pushc 12
rdint
mul
wrint
call new_line
halt
new_line:
    asf 0
    pushc 10
    wrchr
    rsf
    ret
```

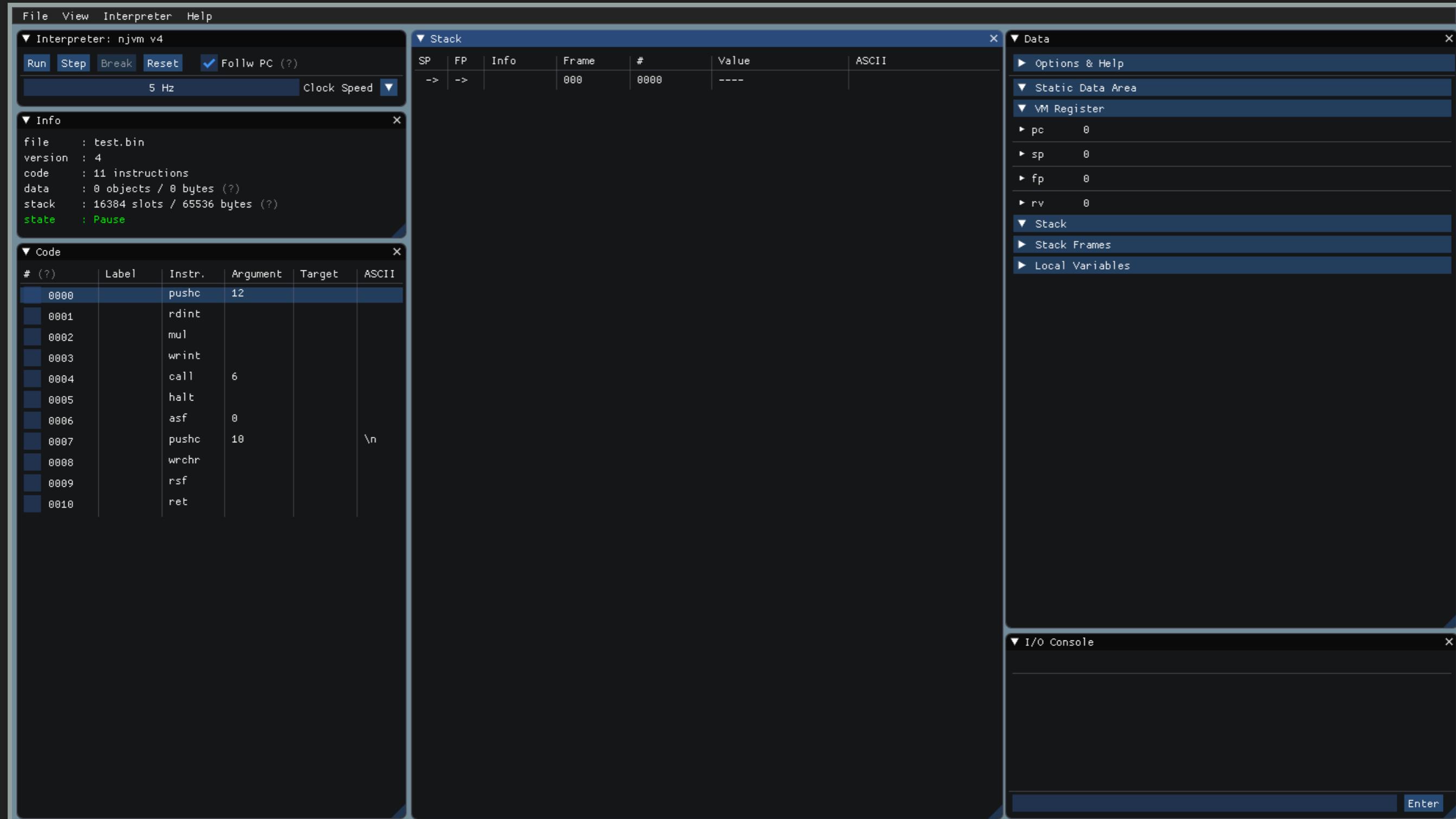
Frage: Was macht diese Funktion ?

Einlesen einer Zahl, Multiplikation mit 12, Ausgabe der Zahl, Ausgabe \n

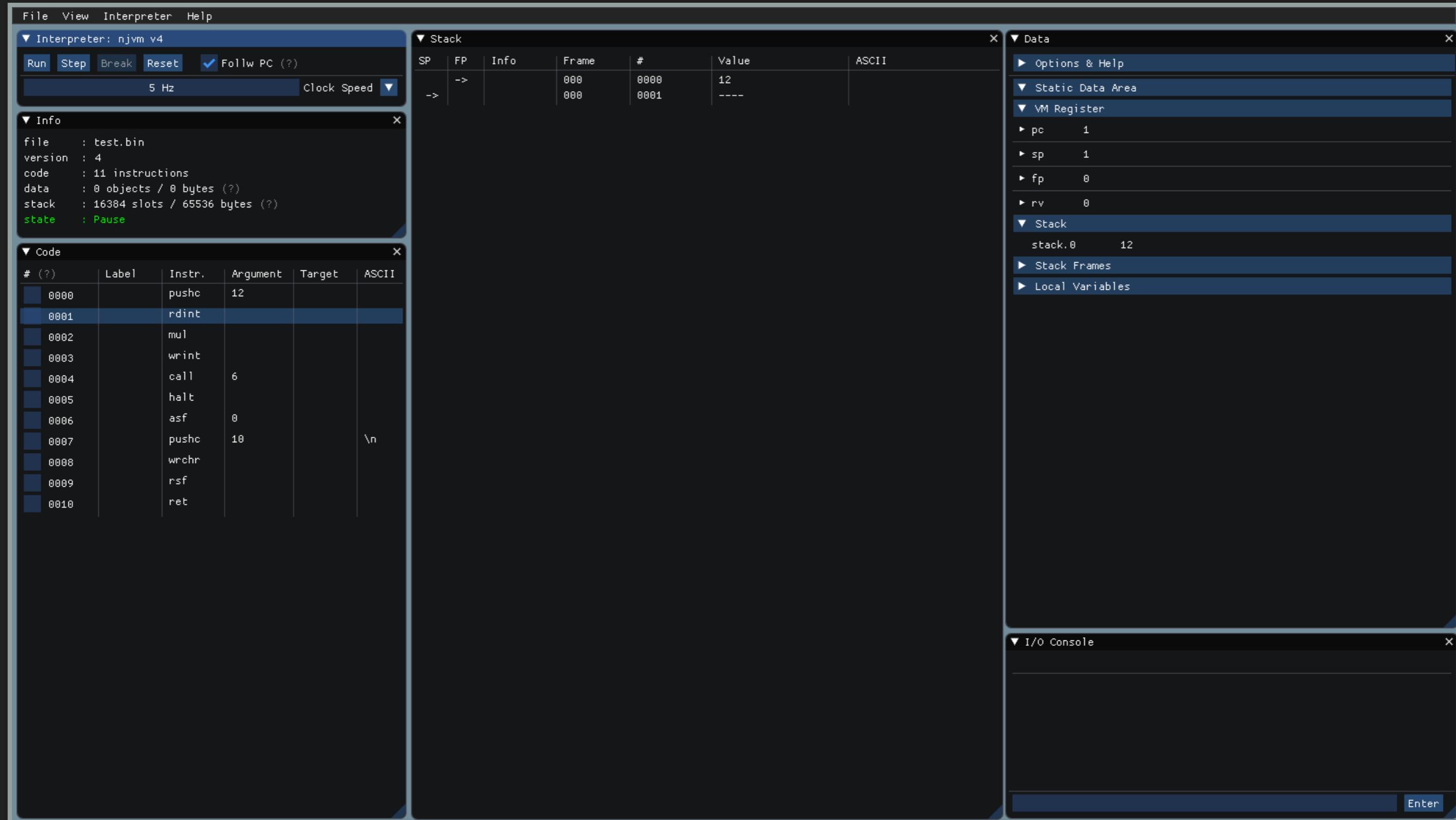
NINJA DEBUGGER

https://homepages.thm.de/~arin07/lectures/ksp/njdb/?font_size=23&interp=4

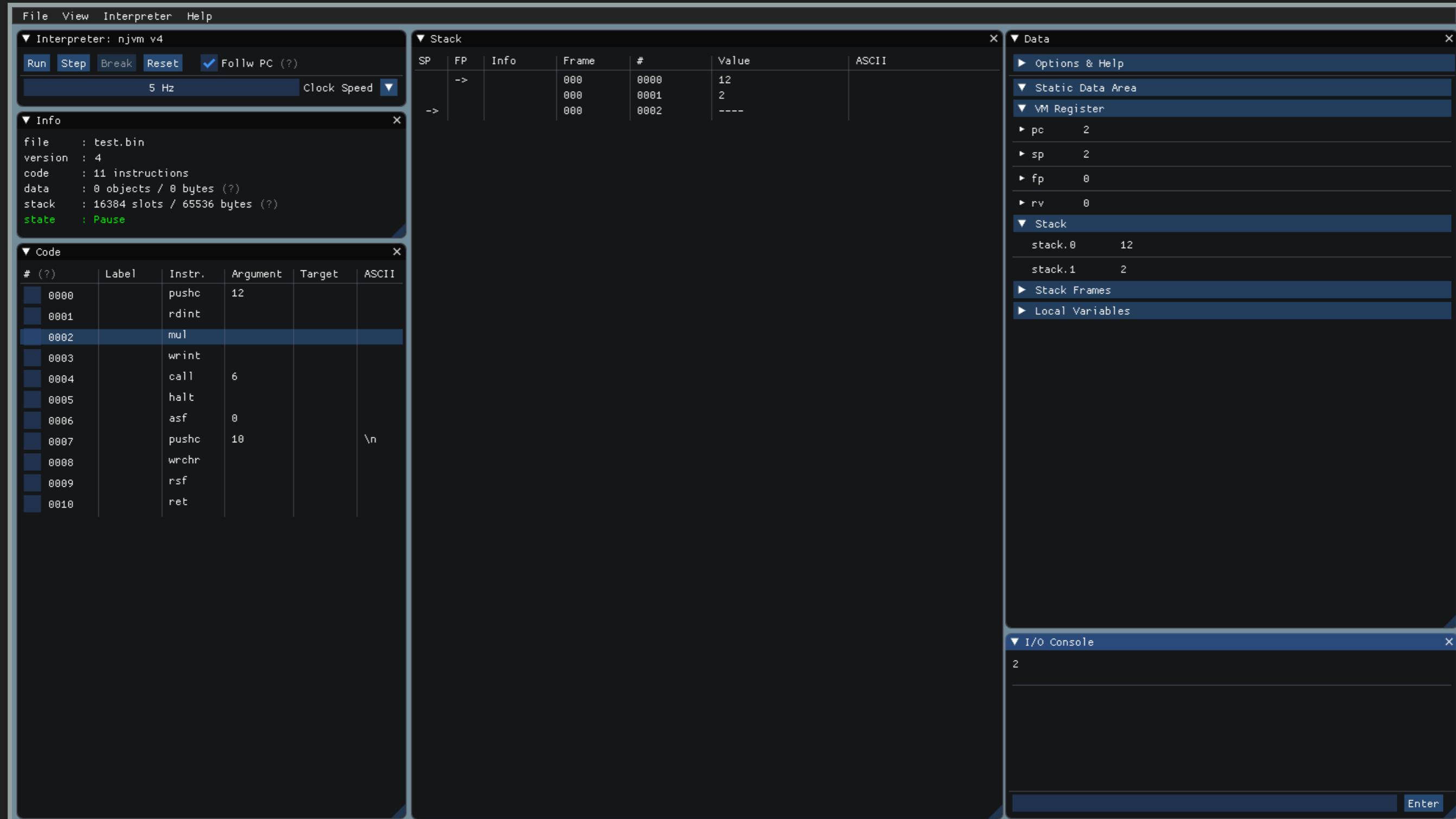
FUNKTIONSAUFRUF: BEISPIEL



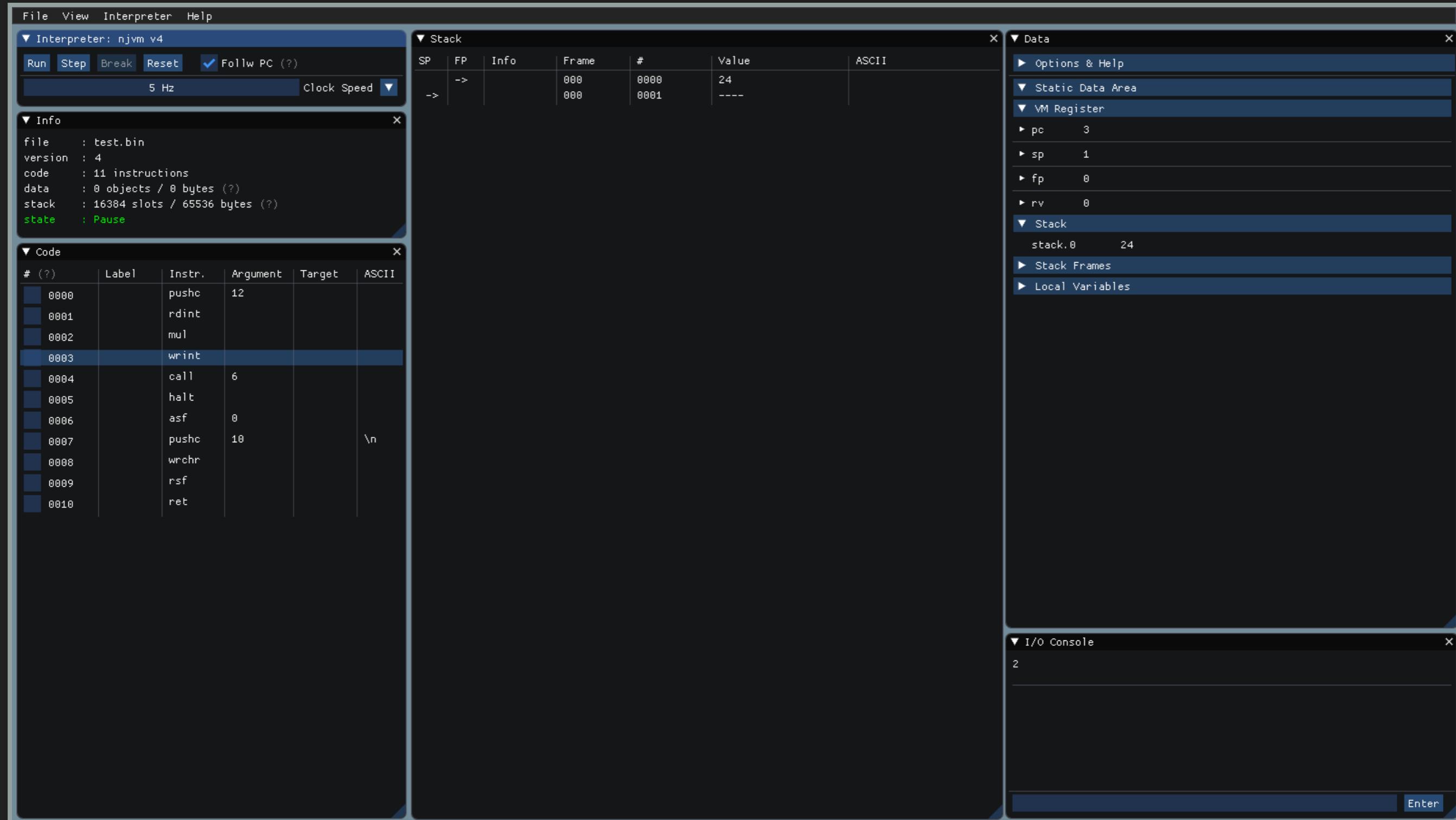
FUNKTIONSAUFRUF: BEISPIEL



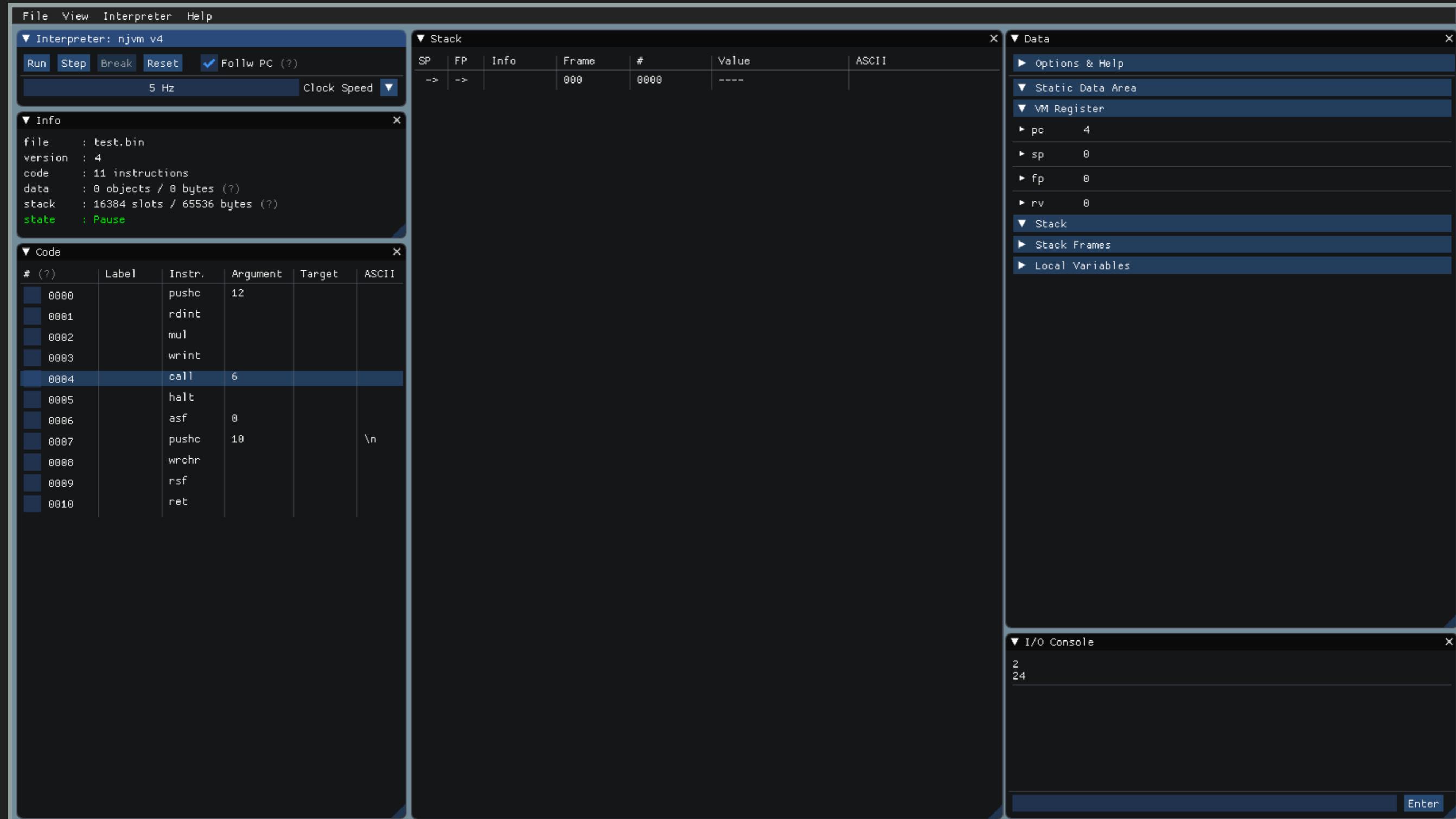
FUNKTIONSAUFRUF: BEISPIEL



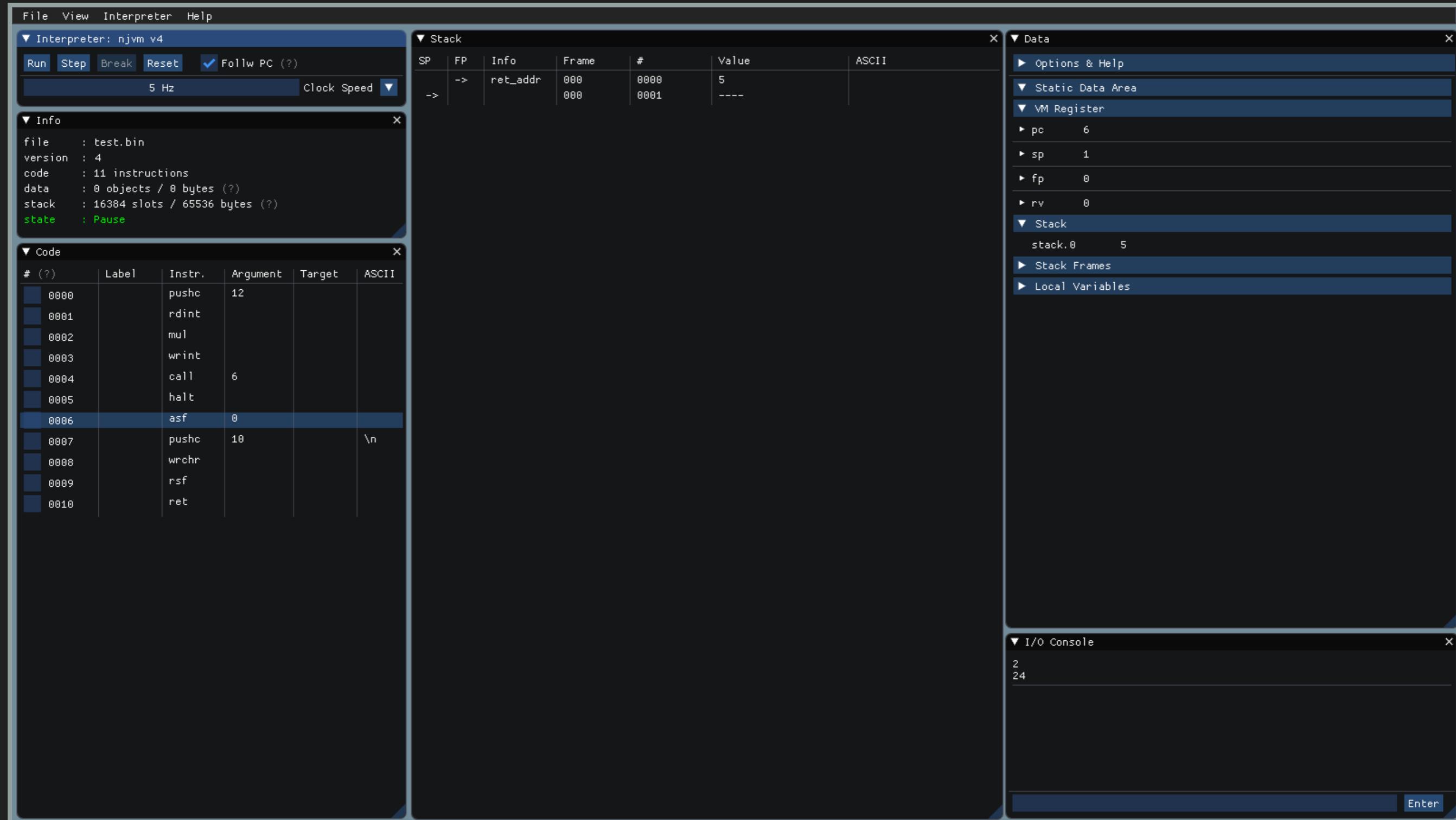
FUNKTIONSAUFRUF: BEISPIEL



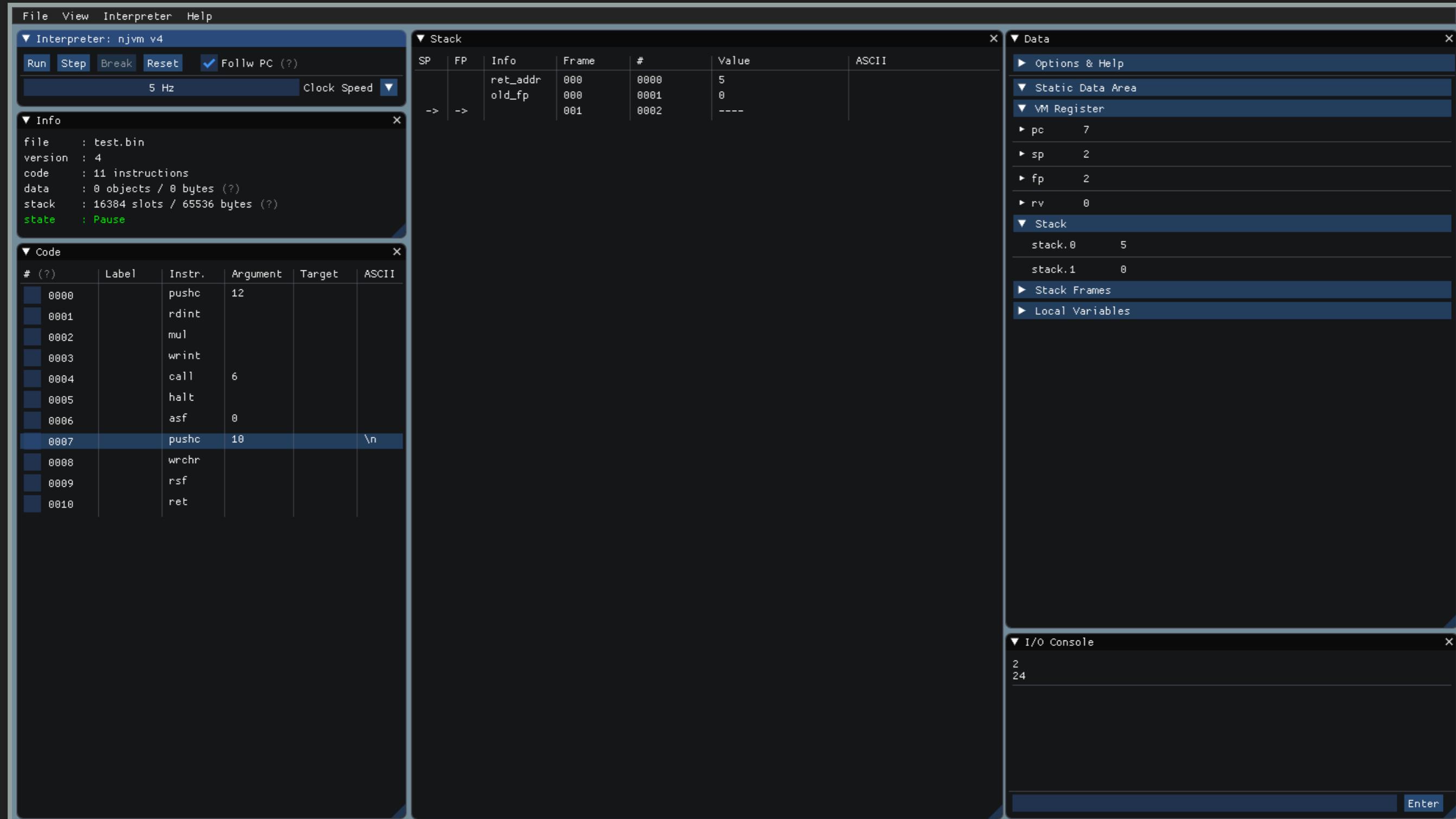
FUNKTIONSAUFRUF: BEISPIEL



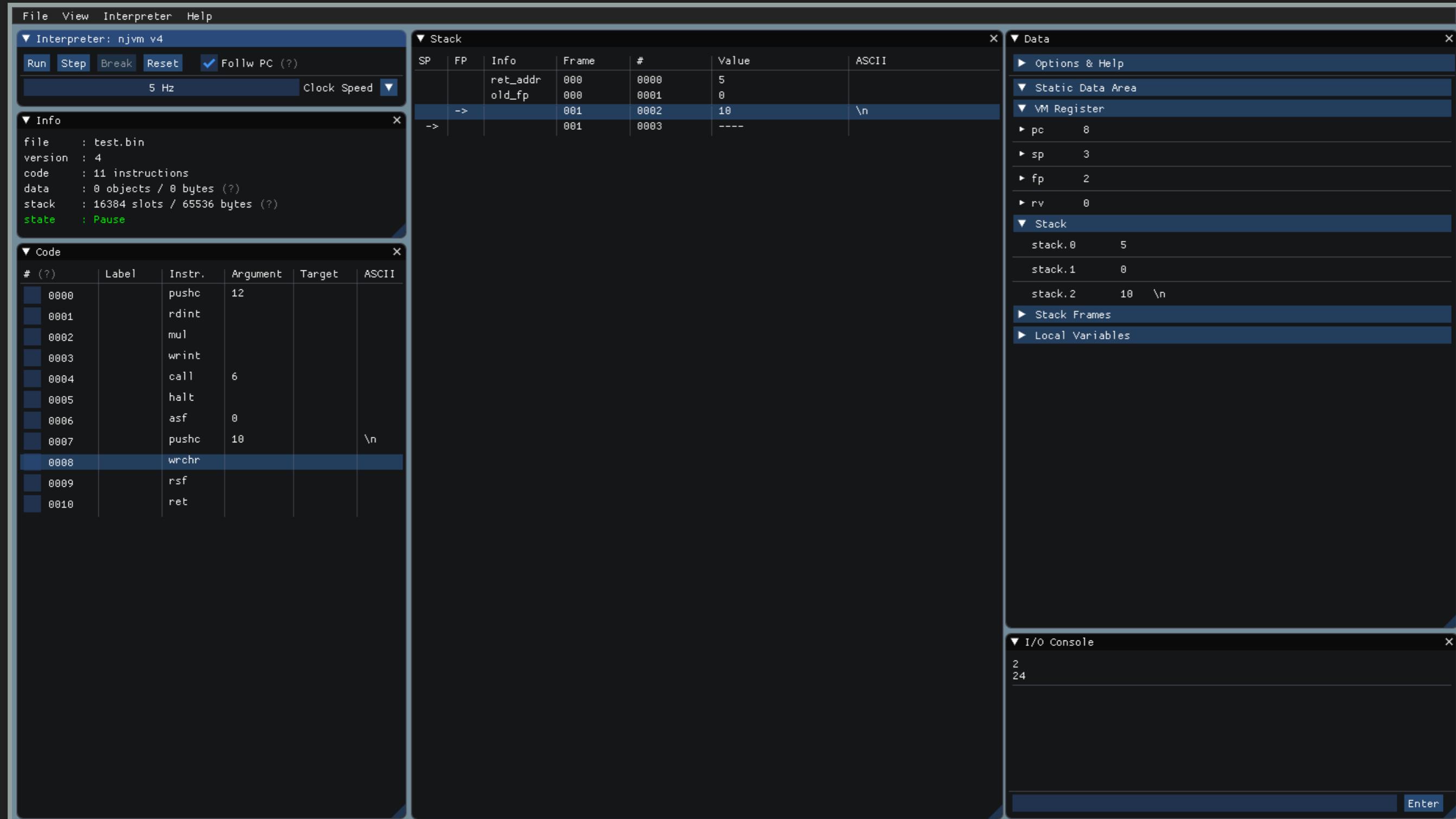
FUNKTIONSAUFRUF: BEISPIEL



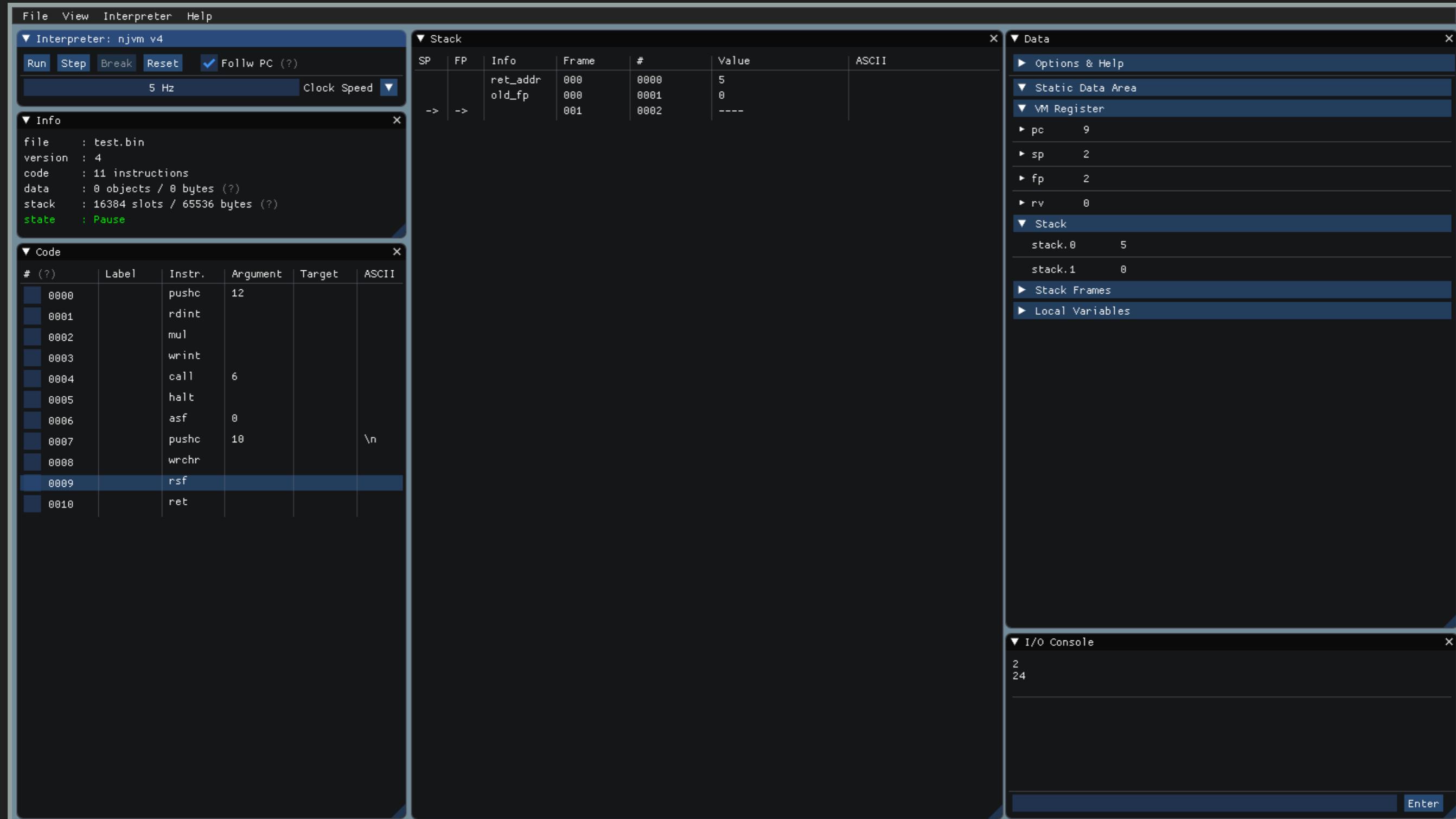
FUNKTIONSAUFRUF: BEISPIEL



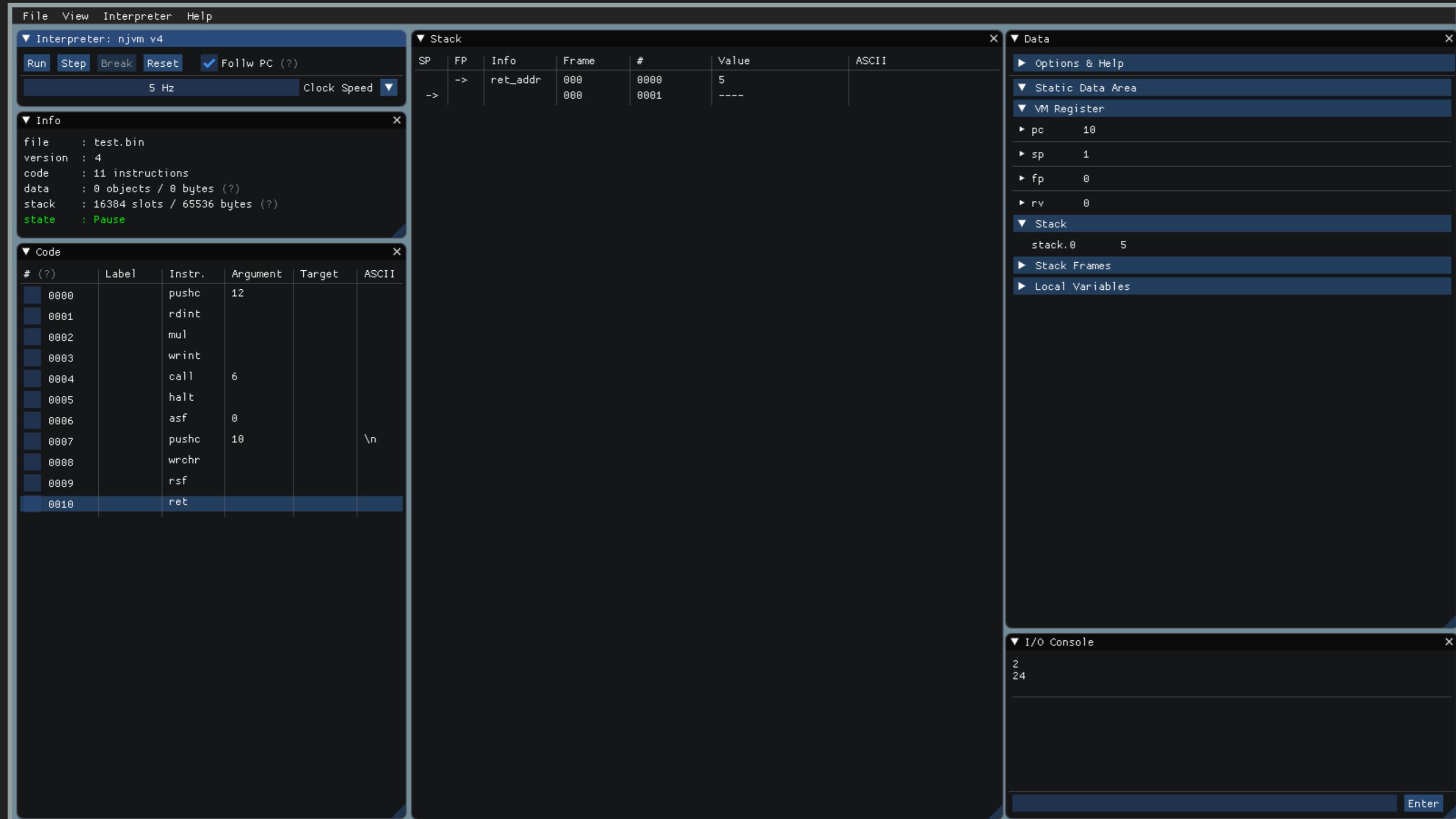
FUNKTIONSAUFRUF: BEISPIEL



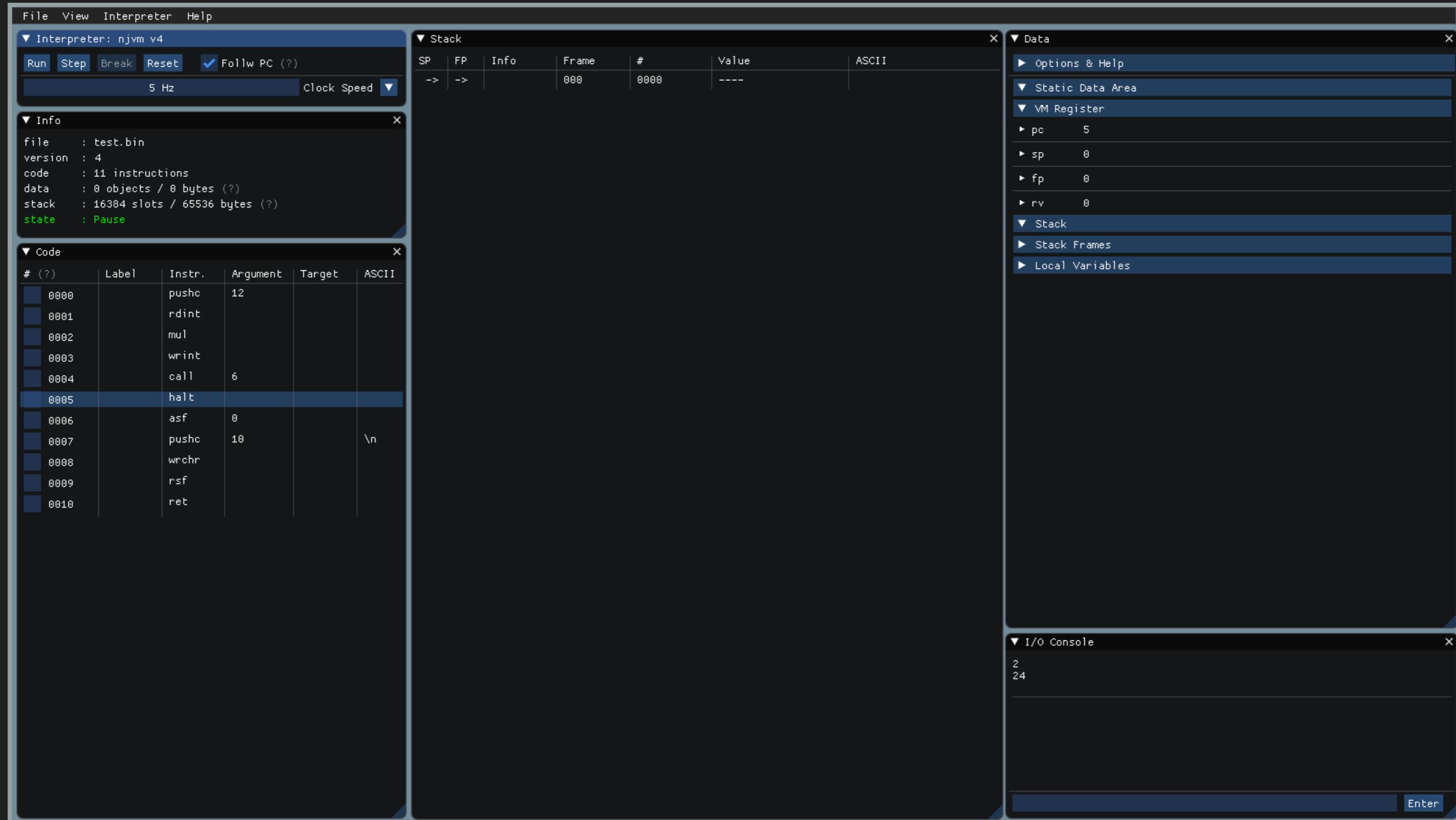
FUNKTIONSAUFRUF: BEISPIEL



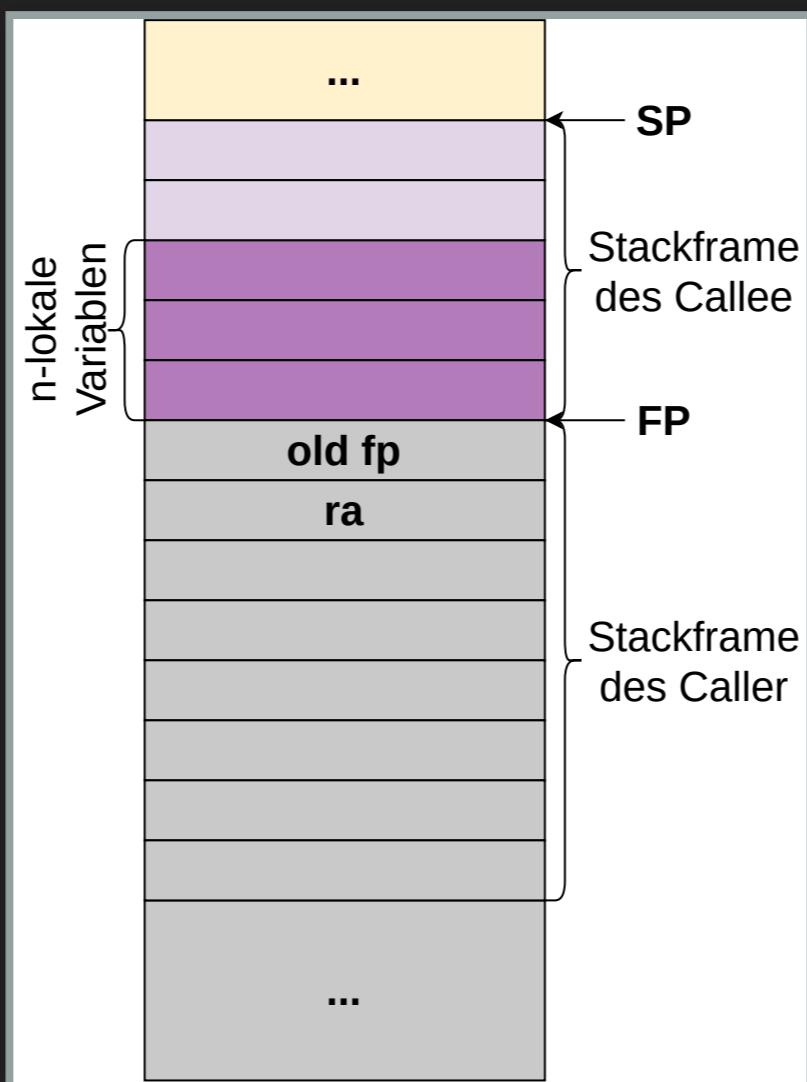
FUNKTIONSAUFRUF: BEISPIEL



FUNKTIONSAUFRUF: BEISPIEL



STACK BEI FUNKTIONSAUFRUF OHNE ARGUMENTE



FUNKTIONSAUFRUF MIT ARGUMENTEN, OHNE RÜCKGABEWERT

Problem: Wie können Argumente an eine Funktion übergeben werden?

Grundsätzlich gibt es verschiedene Varianten, wie eine Argumentübergabe erfolgen kann. Im Sprachdesign von Ninja wurde sich dafür entschieden, dass die Argumentübergabe vollständig über den Stack erfolgt. Diese Designentscheidung nennt man die **Aufrufkonvention** (engl. **Calling Convention**).

Beispiele aus der realen Welt:

- X86 32-Bit Architektur implementiert ebenfalls die Argumentübergabe mittels Stack.
- X86 64-Bit Architektur implementiert eine Mischform, die ersten 6 Argumente werden mittels Registern übergeben und die restlichen auf dem Stack.



Wenn in einer Sprache die Anzahl der übergebenen Parameter nicht begrenzt ist, bzw. die Anzahl der potentiell zu übergebenen Argumente größer ist, als Register zur Verfügung stehen, ist es sinnvoll den Stack für die *restlichen* Werte zu verwenden.

FUNKTIONSAUFRUF MIT ARGUMENTEN, OHNE RÜCKGABEWERT

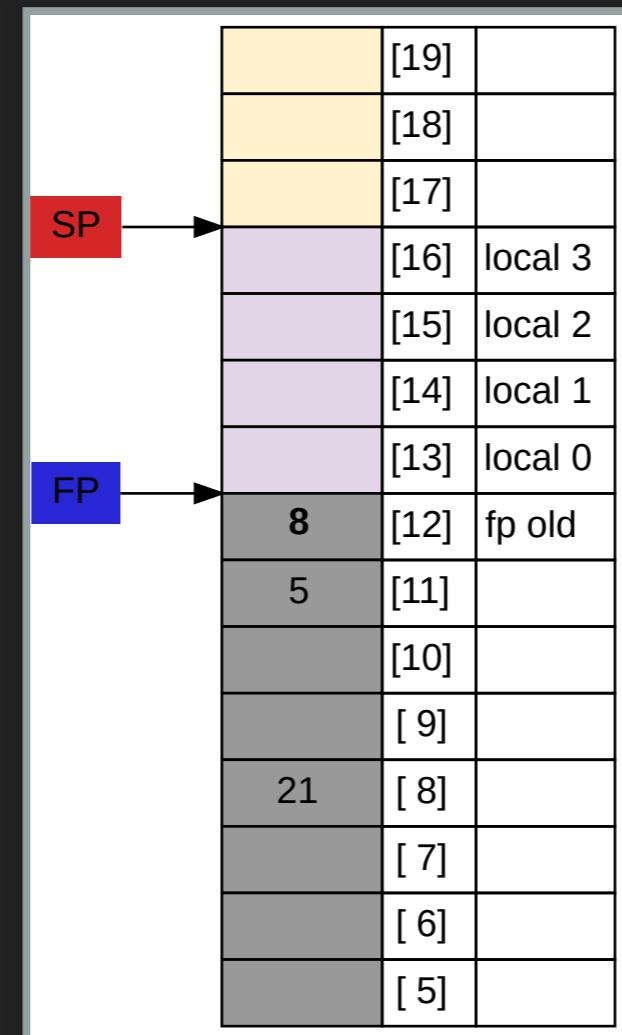
Das Problem der Argumentübergabe mittels Stack ist die **Adressierung** der einzelnen Argumente.

- Unter der Annahme, dass je nach Funktion eine **unterschiedliche** Anzahl an Argumenten übergeben wird, erlaubt hierbei **keine feste** oder gar **absolute** Zuordnung zwischen Argument und Adresse.
- Den Trick den man verwendet ist:
 - Man sucht sich einen **festen Bezugspunkt**, dessen Adresse während der Dauer des Funktionsaufrufs nicht verändert wird, und
 - adressiert die Argumente **relativ** zu diesem Bezugspunkt.

FUNKTIONSAUFRUF MIT ARGUMENTEN, OHNE RÜCKGABEWERT

Wir haben dies schon einmal gemacht, erinnern Sie sich wobei?

Wir adressieren die **lokalen Variablen** als positiven Offset zum Framepointer (FP).



FUNKTIONSAUFRUF MIT ARGUMENTEN, OHNE RÜCKGABEWERT

Angenommen wir haben n -Argumente, dann nummerieren wir diese durch. Das 1. Argument (links in der Liste), ist das 0-te Argument und das n -te Argument (rechts in der Liste) hat die Nummer $n-1$.

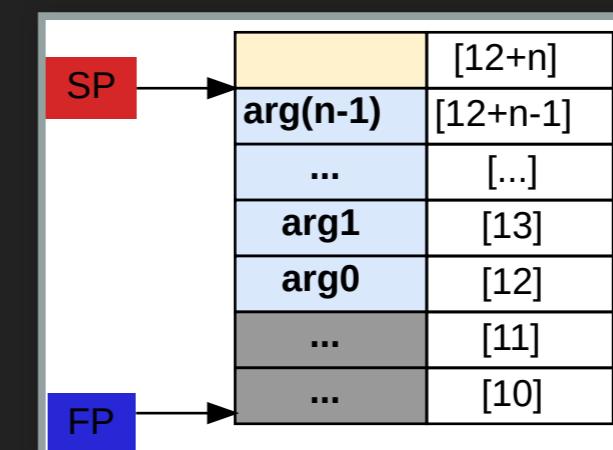
Beispiel:

$f(arg_0, arg_1, arg_2, \dots, arg_{n-1}) \rightarrow n\text{-Argumente}$

Caller:

```
// FP = 10, SP = 12
push arg0
push arg1
...
push arg(n-1)
// FP = 10, SP = 12+n
call Funktionslabel
// FP = 10, SP = 12+n
```

Stack push $arg(n-1)$



Der Stack sieht vor und nach dem Funktionsaufruf `call Funktionslabel` identisch aus!

drop <n>-INSTRUKTION

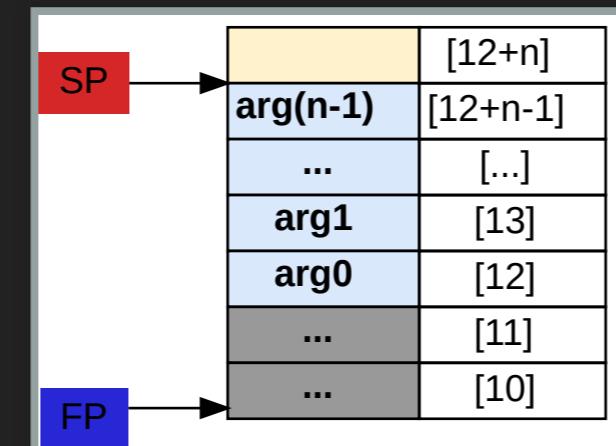
Nach dem Funktionsaufruf (`call Funktionslabel`) werden die Argumente des Funktionsaufrufs allerdings nicht länger benötigt und werden mit der Instruktion `drop <n>` verworfen/gelöscht.

`drop <n> → ... a0 a1...a(n-1) -> ...`

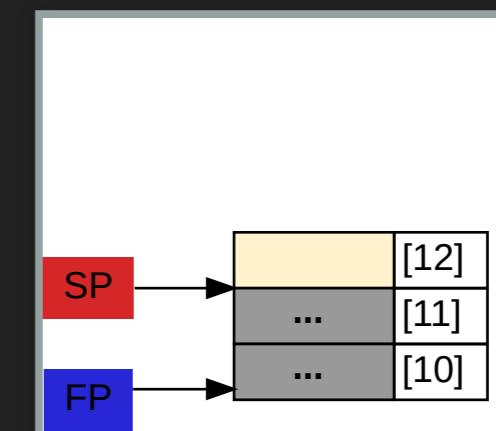
Caller:

```
// FP = 10, SP = 12
push arg0
push arg1
...
push arg(n-1)
// FP = 10, SP = 12+n
call Funktionslabel
// FP = 10, SP = 12+n
drop n //löscht n-Einträge v.Stack
// FP = 10, SP = 12
```

Stack `push arg(n-1)`



Stack (`drop n`)



Nach dem Funktionsaufruf (`call Funktionslabel`) werden die Argumente nicht länger benötigt und mit `drop <n>` verworfen.

FUNKTIONSAUFRUF MIT ARGUMENTEN, OHNE RÜCKGABEWERT

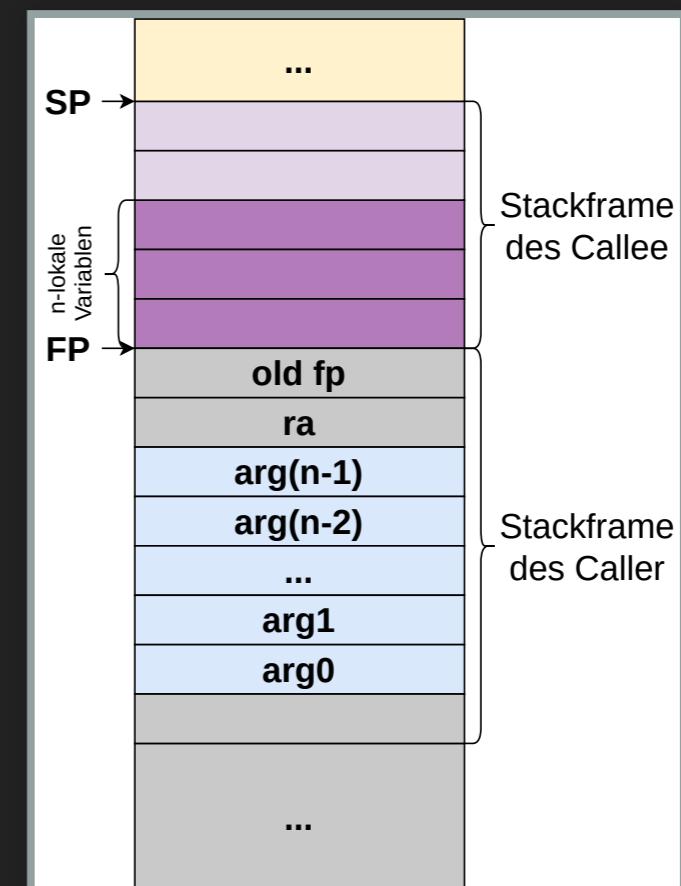
Callee

```
Funktionslabel:  
  asf <numLocals> // numLocals bezeichnet die Anzahl der  
                    // benötigten lokalen Variablen der Funktion  
  <body>           // unbestimmte Anzahl an Instruktionen  
  rsf              // zurücksetzen des aktuellen Stackframes  
  ret              // zurückkehren zum Aufrufer (Caller!)
```

Der Zugriff auf die Argumente erfolgt über einen negativen Offset an `pushl <offset>` und `popl <offset>` als Immediate Wert.

Die Berechnung des negativen Offsets für ein Argument `i` erfolgt durch `stack[FP-2-n+i]`. Da `popl` und `pushl` bereits eine relative Adressierung mittels Framepointer verwenden, ist für uns nur die Berechnung des Offset-Wertes (`-2-n+i`) von Interesse, da dieser Wert das negative Offset *relativ* zum aktuellen `FP` angibt.

Stack



ANMERKUNG: OFFSET-BERECHNUNG



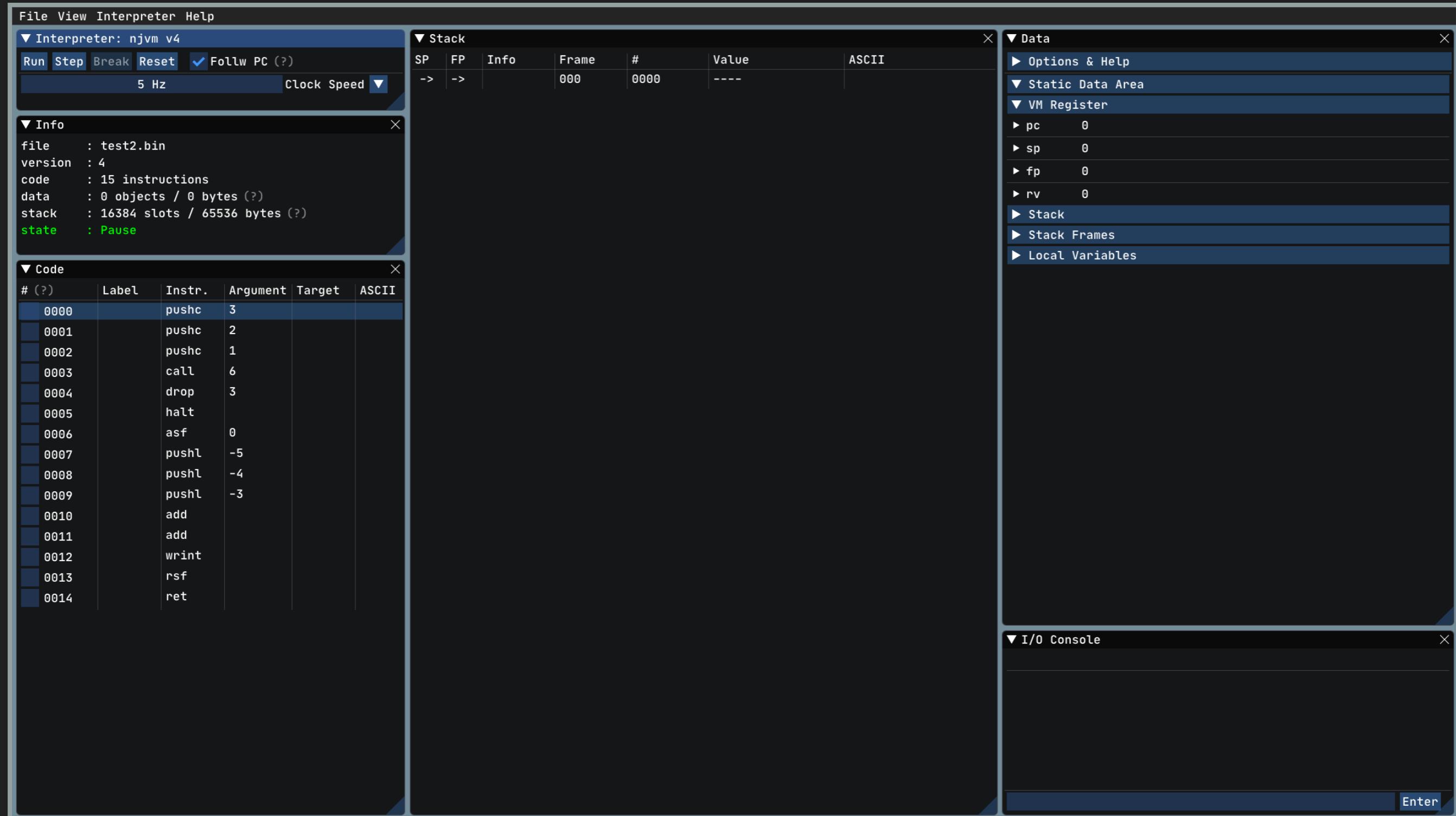
Die Berechnung des korrekten (negativen) Offset-Wertes an `pushl` und `popl` wird vom Compiler berechnet. Wir müssen uns um die Anpassung also nur dann kümmern, wenn wir direkt Ninja-Assembler Code schreiben und diesen mit `nja` in Bytecode umwandeln.

BEISPIEL: FUNKTIONSAUFRUF MIT 3 ARGUMENTEN

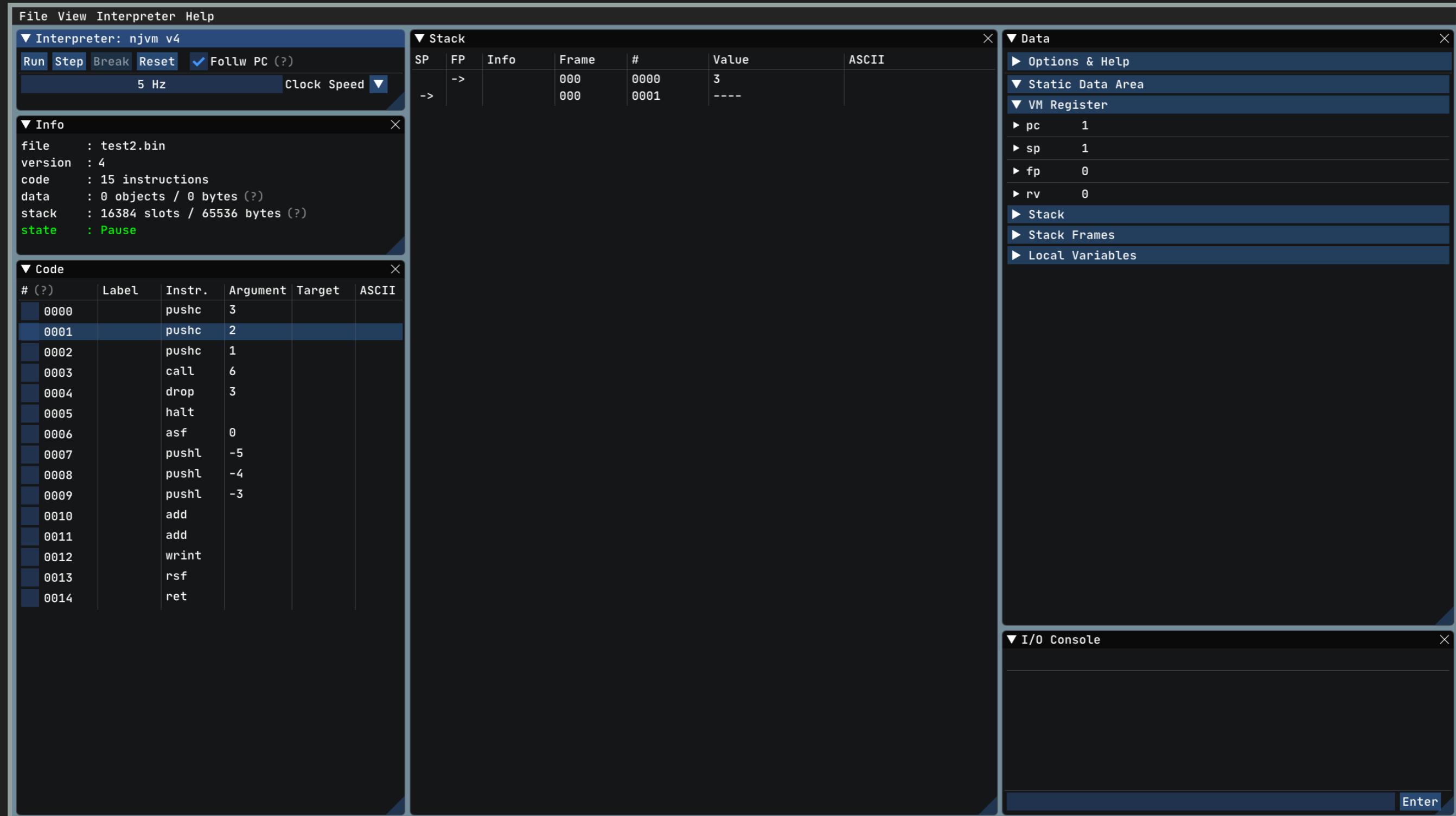
Beispiel: Ausgabe von `f(3, 2, 1)` mit `f(a, b, c) {wrint (a+b+c);}`

```
pushc 3
pushc 2
pushc 1
call f
drop 3
halt
f:
    asf 0
    pushl -5
    pushl -4
    pushl -3
    add
    add
    wrint
    rsf
    ret
```

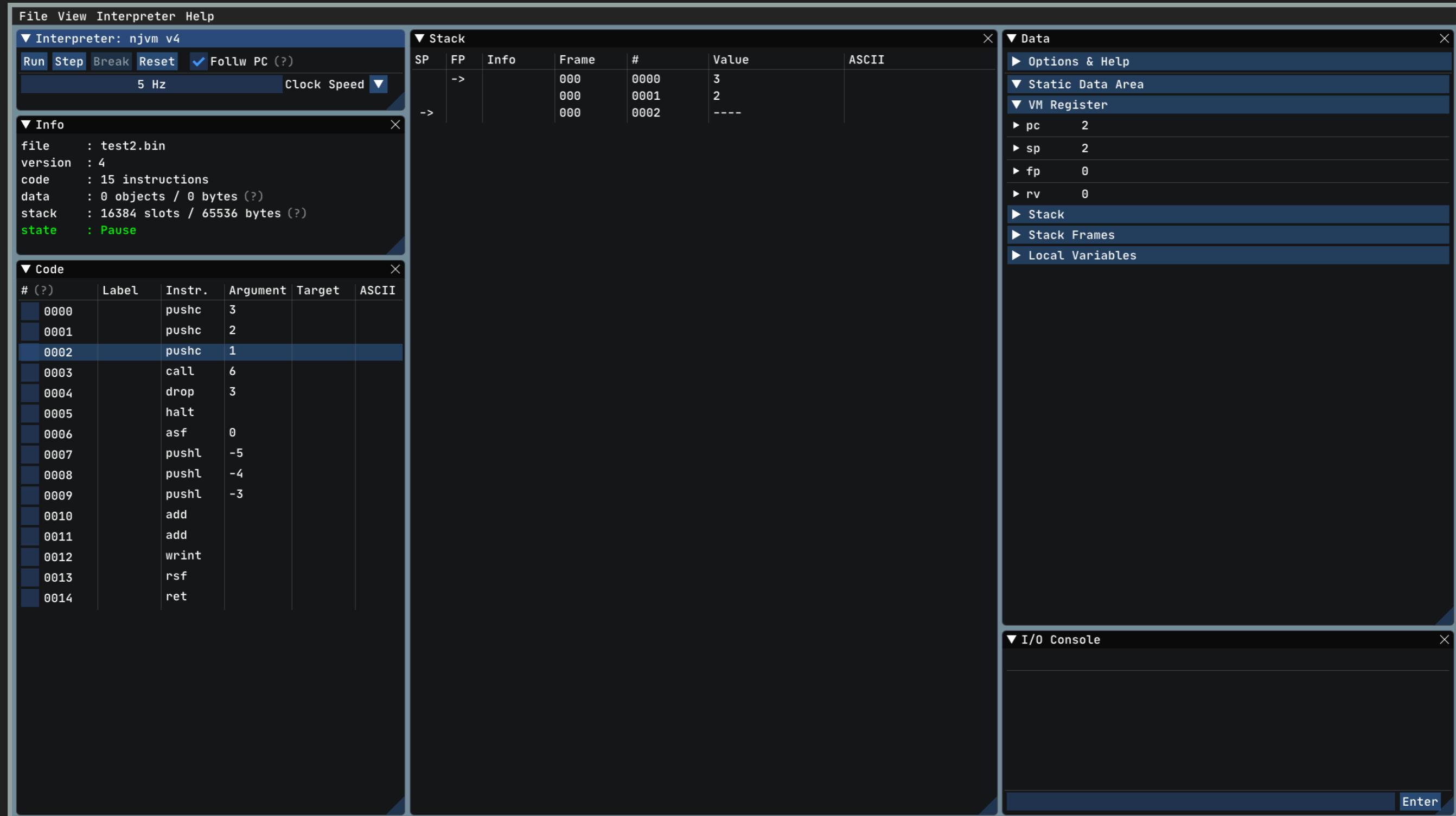
BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

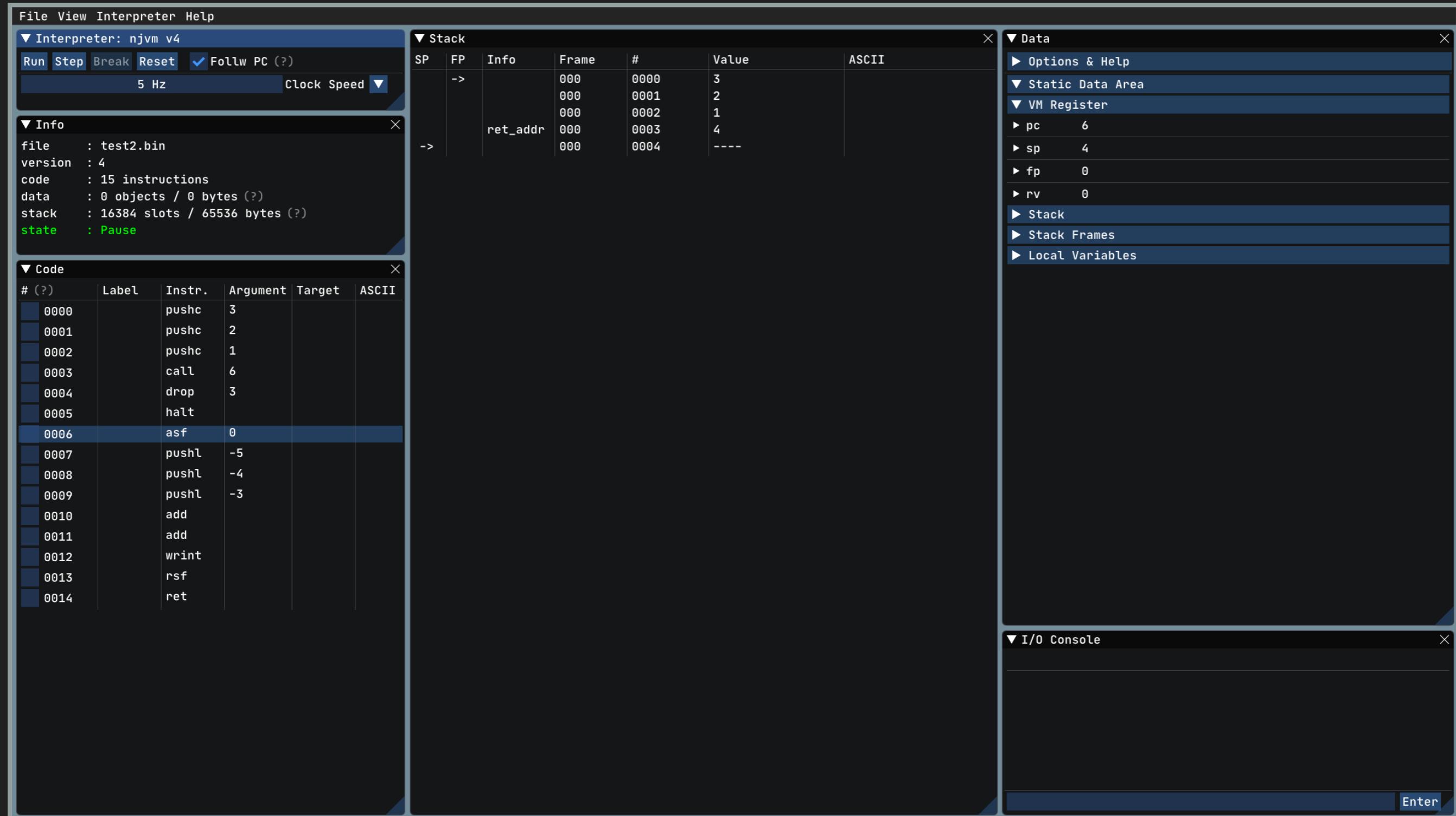


BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several windows open:

- Interpreter: njvm v4**: Shows buttons for Run, Step, Break, Reset, and Follow PC (checked). A dropdown for Clock Speed is set to 5 Hz.
- Info**: Displays file: test2.bin, version: 4, code: 15 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), and state: Pause.
- Code**: A table showing assembly-like instructions. The instruction at address 0003 is highlighted in blue, showing it's a call to address 0006. Other instructions include pushc, drop, halt, asf, pushl, add, wrint, rsf, and ret.
- Stack**: A table showing the stack structure. The stack grows from higher addresses (000) to lower addresses (->). It contains four frames with values 3, 2, 1, and a null value (----).
- Data**: A tree view showing VM Register, Stack, Stack Frames, and Local Variables. The pc register is set to 3, sp to 3, fp to 0, and rv to 0.
- I/O Console**: An empty text input field with an Enter button.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 debugger interface with several windows open:

- Interpreter: njvm v4**: Shows buttons for Run, Step, Break, Reset, and Follow PC. The state is set to "Pause".
- Stack**: A table showing the stack structure. Columns include SP, FP, Info, Frame, #, Value, and ASCII. The stack contains values 3, 2, 1, 4, 0, and a blank line (----). Registers ret_addr, old_fp, and fp are also listed.
- Data**: A tree view showing VM Register, Stack, Stack Frames, and Local Variables. Registers pc, sp, fp, and rv are listed with their current values.
- Code**: An assembly code window showing instructions from address 0000 to 0014. The instruction at address 0007 is highlighted: pushl -5. Other instructions include pushc, call, drop, halt, and various arithmetic operations.
- I/O Console**: An empty window for input/output.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several panes:

- Interpreter: njvm v4**: Top-left pane with buttons: Run, Step, Break, Reset, Follow PC (?). It also shows Clock Speed (5 Hz).
- Info**: Top-right pane displaying file: test2.bin, version: 4, code: 15 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), state: Pause.
- Code**: Bottom-left pane showing assembly-like code:

# (?)	Label	Instr.	Argument	Target	ASCII
0000		pushc	3		
0001		pushc	2		
0002		pushc	1		
0003		call	6		
0004		drop	3		
0005		halt			
0006		ASF	0		
0007		pushl	-5		
0008		pushl	-4		
0009		pushl	-3		
0010		add			
0011		add			
0012		wrint			
0013		rsf			
0014		ret			
- Stack**: Middle-right pane showing the stack structure:

SP	FP	Info	Frame	#	Value	ASCII
			000	0000	3	
			000	0001	2	
			000	0002	1	
		ret_addr	000	0003	4	
		old_fp	000	0004	0	
->			001	0005	3	
->			001	0006	----	
- Data**: Rightmost pane showing VM Register values:

pc	8
sp	6
fp	5
rv	0
- I/O Console**: Bottom-right pane with an Enter button.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 debugger interface with several windows open:

- Interpreter: njvm v4**: Top-left window with tabs for Run, Step, Break, Reset, and Follow PC. It also shows Clock Speed set to 5 Hz.
- Info**: Shows file: test2.bin, version: 4, code: 15 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, and state: Pause.
- Code**: A table showing assembly-like code with columns for # (?), Label, Instr., Argument, Target, and ASCII. The code includes pushc, call, drop, halt, asf, pushl, add, wrint, rsf, and ret instructions.
- Stack**: A table showing the stack structure with columns SP, FP, Info, Frame, #, Value, and ASCII. The stack contains values 3, 2, 1, 4, 0, 3, 2, and a blank entry for frame 007.
- Data**: A tree view showing VM Register, Stack, Stack Frames, and Local Variables. The VM Register section shows pc (9), sp (7), fp (5), and rv (0).
- I/O Console**: Bottom-right window for input and output.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several panes:

- Interpreter: njvm v4**: Top-left pane with buttons: Run, Step, Break, Reset, Follow PC (?), Clock Speed (5 Hz).
- Info**: Top-right pane showing file: test2.bin, version: 4, code: 15 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), state: Pause.
- Code**: Bottom-left pane showing assembly-like code:

# (?)	Label	Instr.	Argument	Target	ASCII
0000		pushc	3		
0001		pushc	2		
0002		pushc	1		
0003		call	6		
0004		drop	3		
0005		halt			
0006		ASF	0		
0007		pushl	-5		
0008		pushl	-4		
0009		pushl	-3		
0010		add			
0011		add			
0012		wrint			
0013		rsf			
0014		ret			
- Stack**: Middle-right pane showing the stack structure:

SP	FP	Info	Frame	#	Value	ASCII
			000	0000	3	
			000	0001	2	
			000	0002	1	
		ret_addr	000	0003	4	
		old_fp	000	0004	0	
->			001	0005	3	
->			001	0006	2	
->			001	0007	1	
->			001	0008	----	
- Data**: Right-side pane showing VM register values:

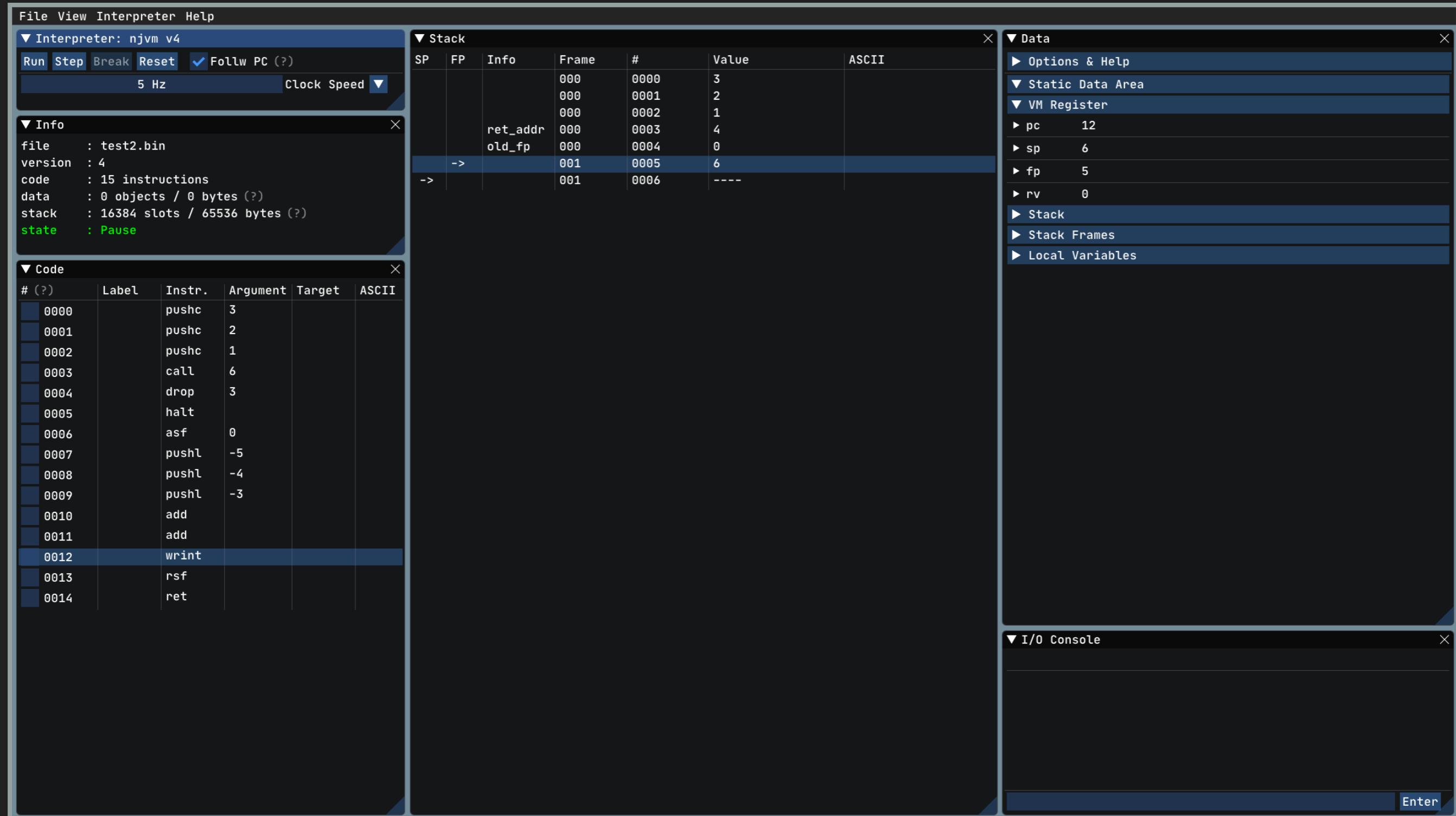
pc	sp	fp	rv
10	8	5	0
- I/O Console**: Bottom-right pane with an Enter button.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several windows open:

- Interpreter: njvm v4**: Top-left window with tabs for Run, Step, Break, Reset, and Follow PC (checked). It also shows Clock Speed (5 Hz) and Info: file : test2.bin, version : 4, code : 15 instructions, data : 0 objects / 0 bytes (?), stack : 16384 slots / 65536 bytes (?), state : Pause.
- Stack**: Table showing the stack structure. Columns include SP, FP, Info, Frame, #, Value, and ASCII. Rows show values for ret_addr, old_fp, and two frames (001 and 0007).
- Data**: Right-hand sidebar with sections for Options & Help, Static Data Area, VM Register, Stack, Stack Frames, and Local Variables. It lists VM registers like pc (11), sp (7), fp (5), and rv (0).
- Code**: Bottom-left window showing assembly-like code. Columns include # (?), Label, Instr., Argument, Target, and ASCII. Instructions include pushc, call, drop, halt, asf, pushl, add, wrint, rsf, and ret.
- I/O Console**: Bottom-right window for I/O operations, with an Enter button at the bottom.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 debugger interface with several windows open:

- Interpreter: njvm v4**: Top-left window showing run controls (Run, Step, Break, Reset, Follow PC) and a clock speed of 5 Hz.
- Info**: Window displaying file: test2.bin, version: 4, code: 15 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, state: Pause.
- Code**: Window showing assembly code with columns: # (?), Label, Instr., Argument, Target, ASCII. The code includes instructions like pushc, call, drop, halt, asf, pushl, add, wrint, rsf, and ret.
- Stack**: Window showing the stack structure with columns: SP, FP, Info, Frame, #, Value, ASCII. It lists frames 000 through 005, with values 3, 2, 1, 4, 0, and ---- respectively. Local variables shown are ret_addr, old_fp, and pc.
- Data**: Right-hand window showing VM register values: pc (13), sp (5), fp (5), rv (0). It also includes links to Options & Help, Static Data Area, VM Register, Stack, Stack Frames, and Local Variables.
- I/O Console**: Bottom-right window showing the output "6".

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

The screenshot shows the njvm v4 debugger interface with several windows open:

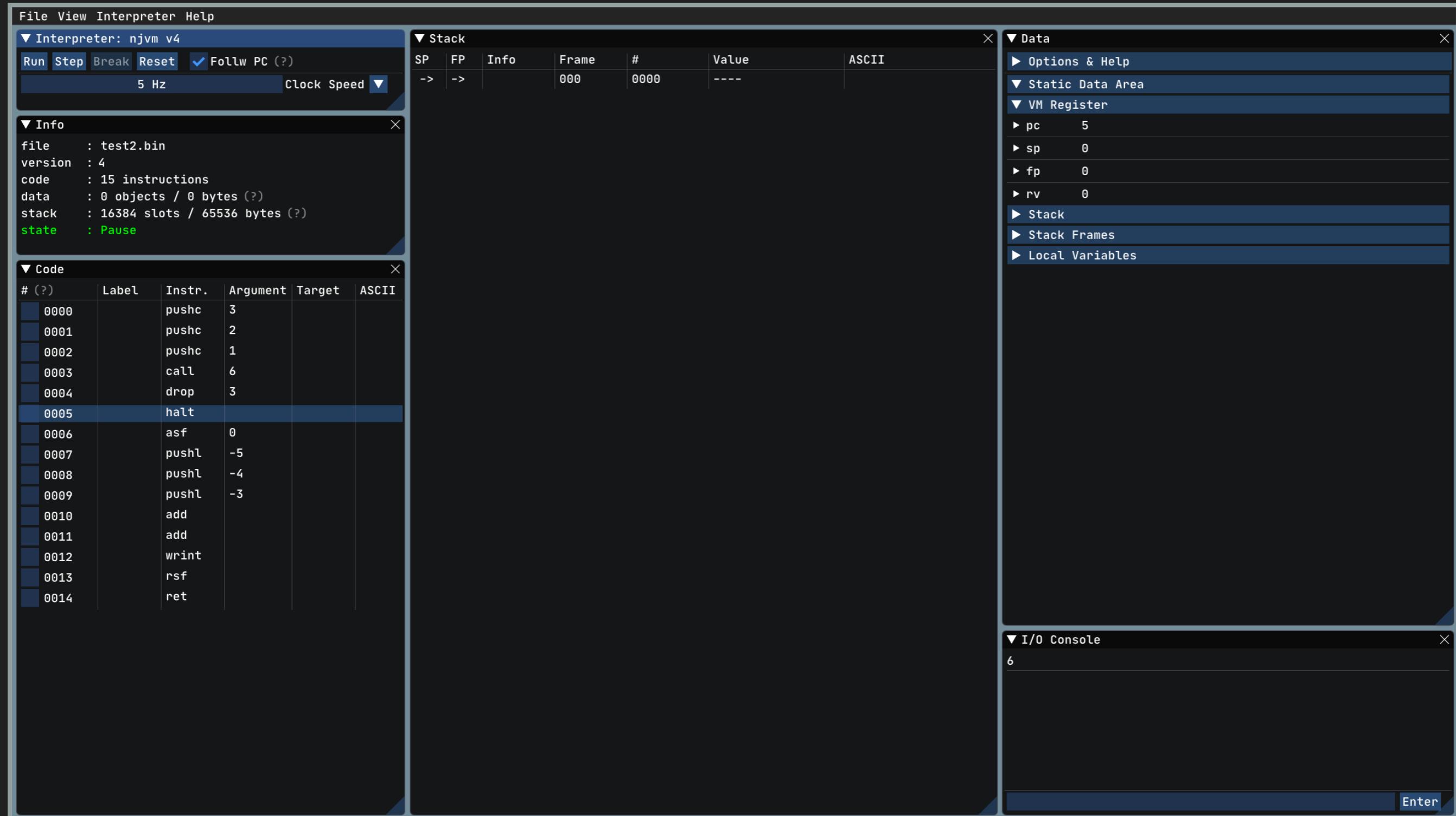
- Interpreter: njvm v4**: Top-left window with tabs for Run, Step, Break, Reset, and Follow PC. It also shows Clock Speed set to 5 Hz.
- Info**: Shows file: test2.bin, version: 4, code: 15 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, and state: Pause.
- Code**: A table showing assembly instructions from address 0000 to 0014. The columns are # (?), Label, Instr., Argument, Target, and ASCII. The instructions include pushc, call, drop, halt, asf, pushl, add, wrint, rsf, and ret.
- Stack**: A table showing the stack structure. The columns are SP, FP, Info, Frame, #, Value, and ASCII. The stack contains values 3, 2, 1, 4, and a null value (----) at frame 0004.
- Data**: A tree view showing VM Register, Stack, Stack Frames, and Local Variables. The VM Register section shows pc (14), sp (4), fp (0), and rv (0).
- I/O Console**: Bottom-right window showing the number 6.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT

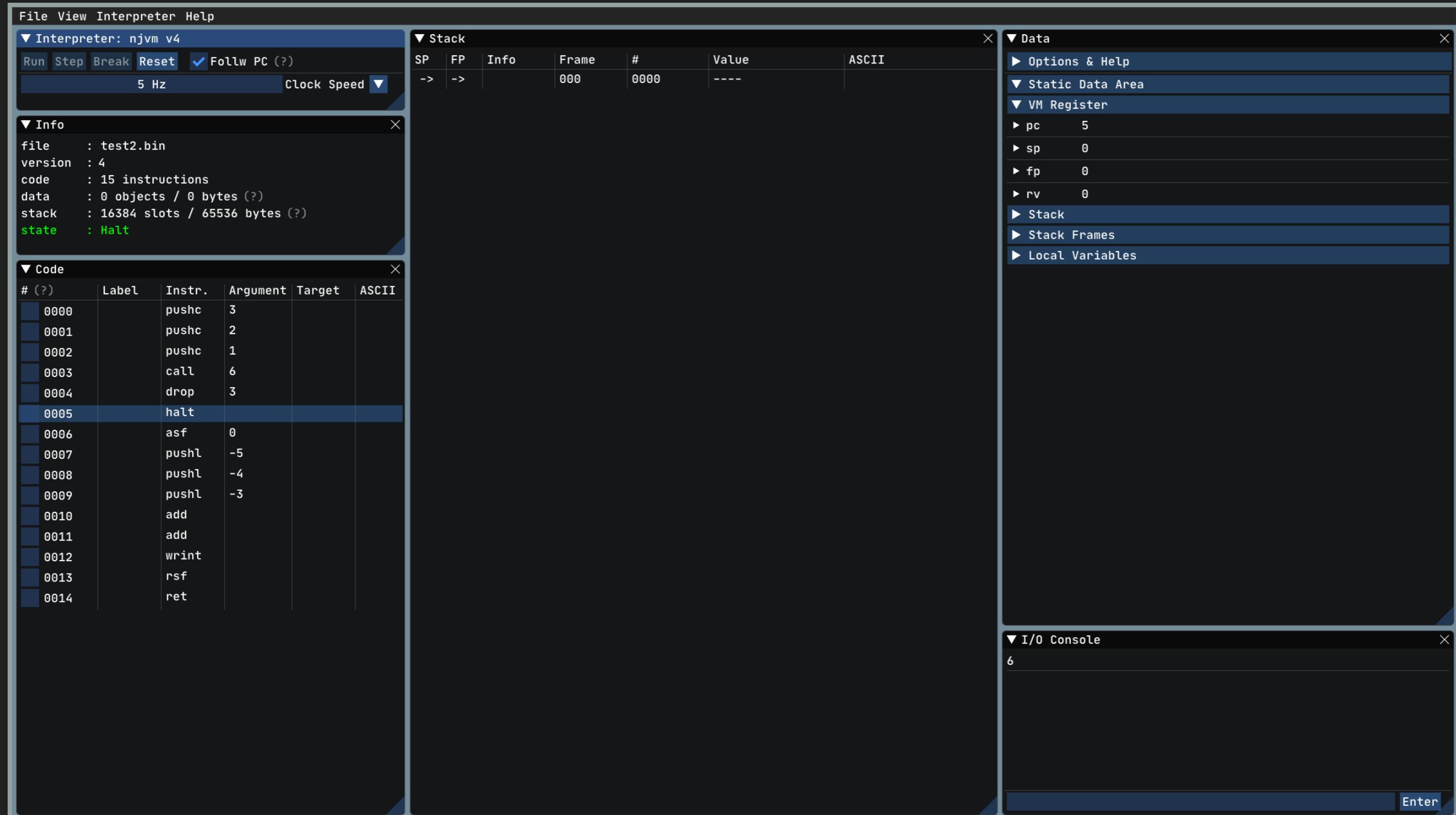
The screenshot shows the njvm v4 interpreter interface with several windows open:

- Interpreter: njvm v4**: Shows buttons for Run, Step, Break, Reset, and Follow PC (checked). A dropdown for Clock Speed is set to 5 Hz.
- Info**: Displays file: test2.bin, version: 4, code: 15 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), and state: Pause.
- Code**: An assembly-like table with columns: # (?), Label, Instr., Argument, Target, and ASCII. The rows show instructions from 0000 to 0014. Instruction 0004 (drop) has argument 3 highlighted.
- Stack**: A table with columns: SP, FP, Info, Frame, #, Value, and ASCII. It shows four frames with values 3, 2, 1, and a blank line.
- Data**: A tree view showing Options & Help, Static Data Area, VM Register, Stack, Stack Frames, and Local Variables. Under VM Register, pc is 4, sp is 3, fp is 0, and rv is 0.
- I/O Console**: Shows the number 6.

BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN, OHNE RÜCKGABEWERT



FUNKTIONSAUFRUF OHNE ARGUMENTEN, MIT RÜCKGABEWERT

- Beispiel: `1+2*f()`, mit `int f(){ return 5; }`



Wir benötigen 2 neue Instruktionen und das neue **Return Value Register (RVR)**, welches als Zwischenspeicher fungiert.

- `pushr` → `... -> ... rv` – holt Wert aus dem RVR-Register und speichert ihn auf dem Stack ab
- `popr` → `... rv -> ...` – holt einen Wert vom Stack und speichert ihn im RVR-Register

FUNKTIONSAUFRUF OHNE ARGUMENTEN, MIT RÜCKGABEWERT

Caller

```
call Funktionslabel  
pushr
```

Callee

Funktionslabel:

```
ASF <numLocalVars>  
<body>  
<push retval> // Vorbereitung für popr  
popr  
rsf  
ret
```

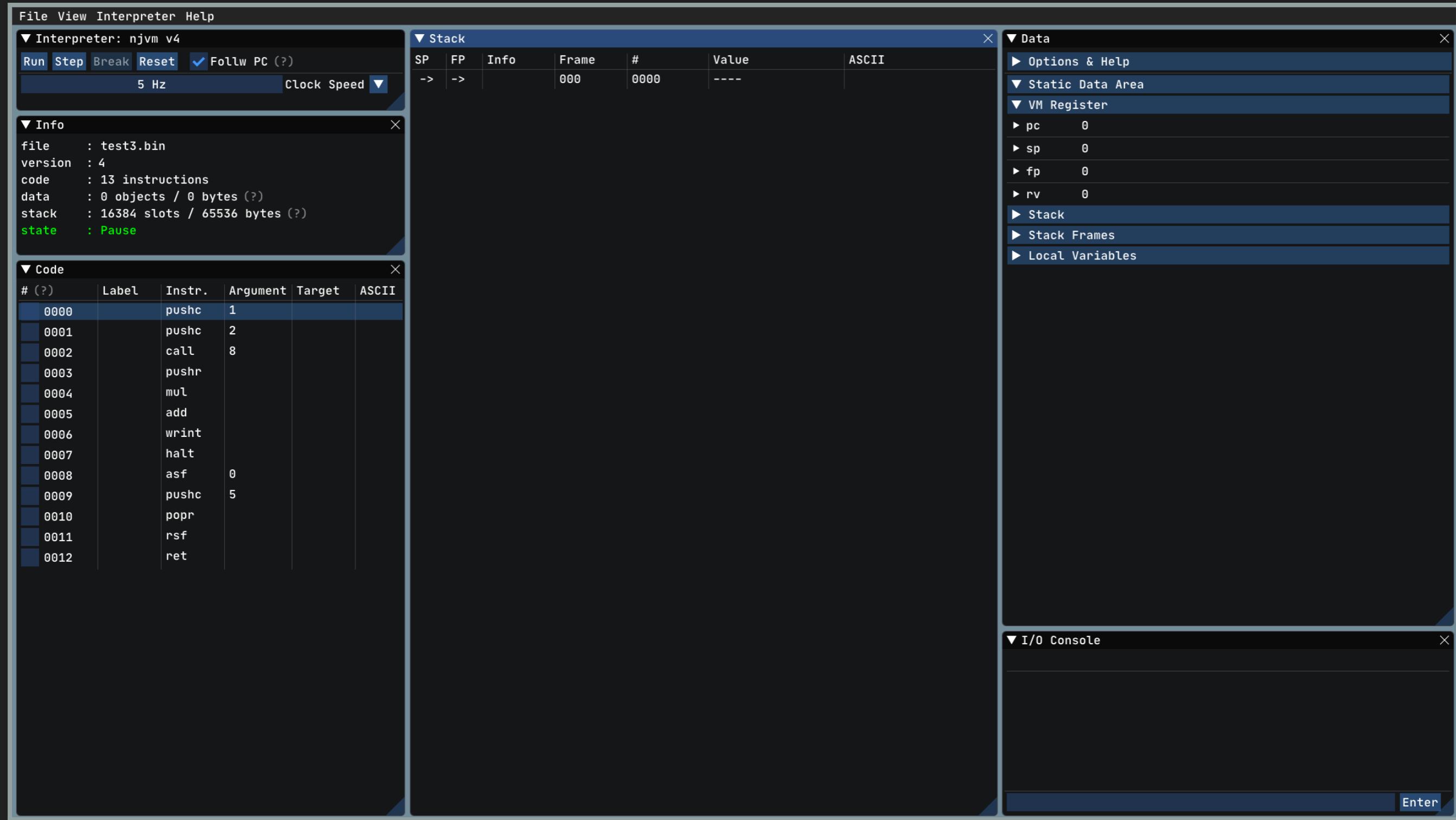
BEISPIEL: FUNKTIONSAUFRUF NUR MIT RÜCKGABEWERT

Beispiel: Ausgabe von $1+2*f()$, mit `int f(){ return 5; }`

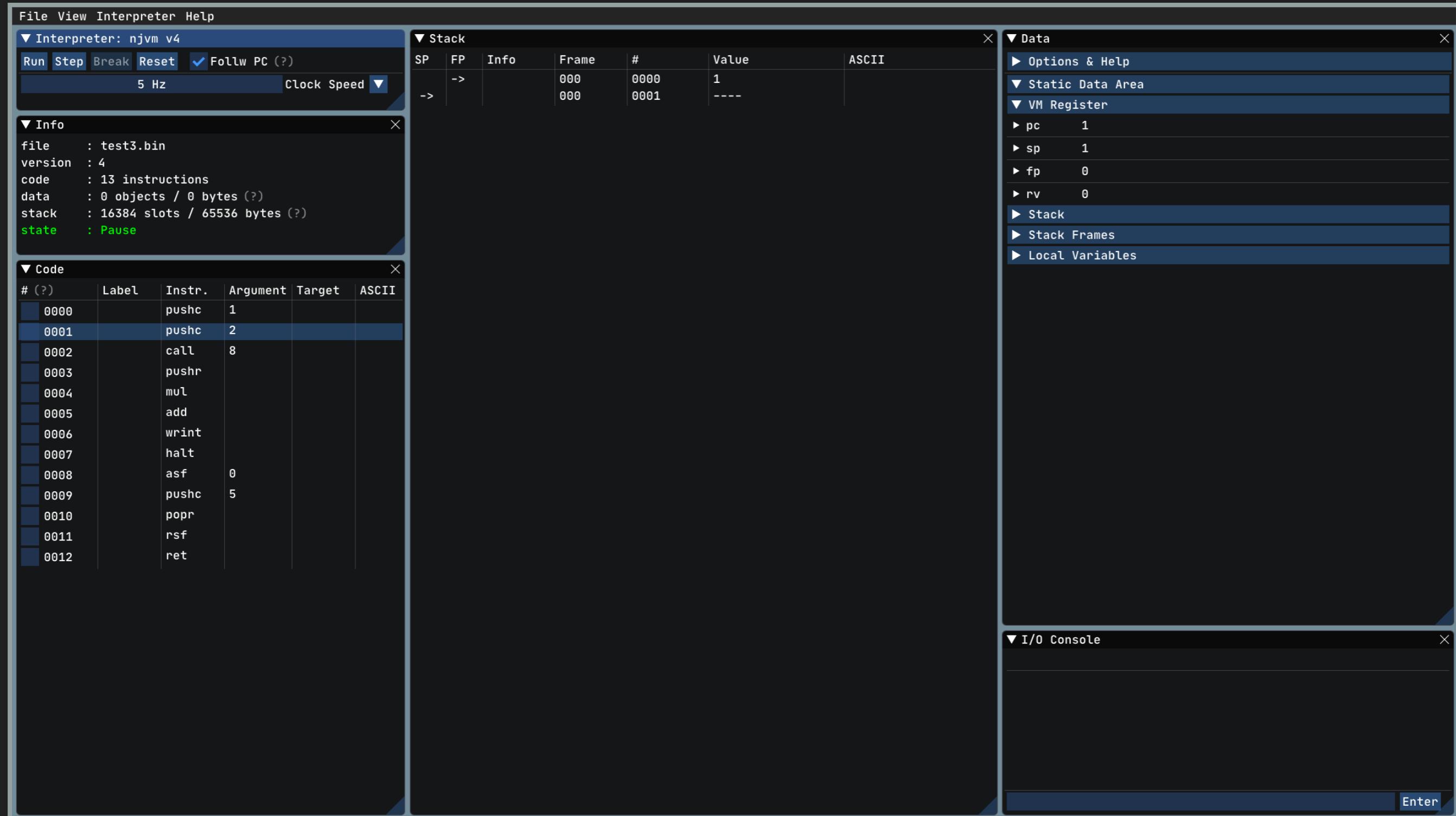
```
pushc 1
pushc 2
call f
pushr
mul
add
wrint
halt
f:
    asf 0
    pushc 5
    popr
    rsf
    ret
```

BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT

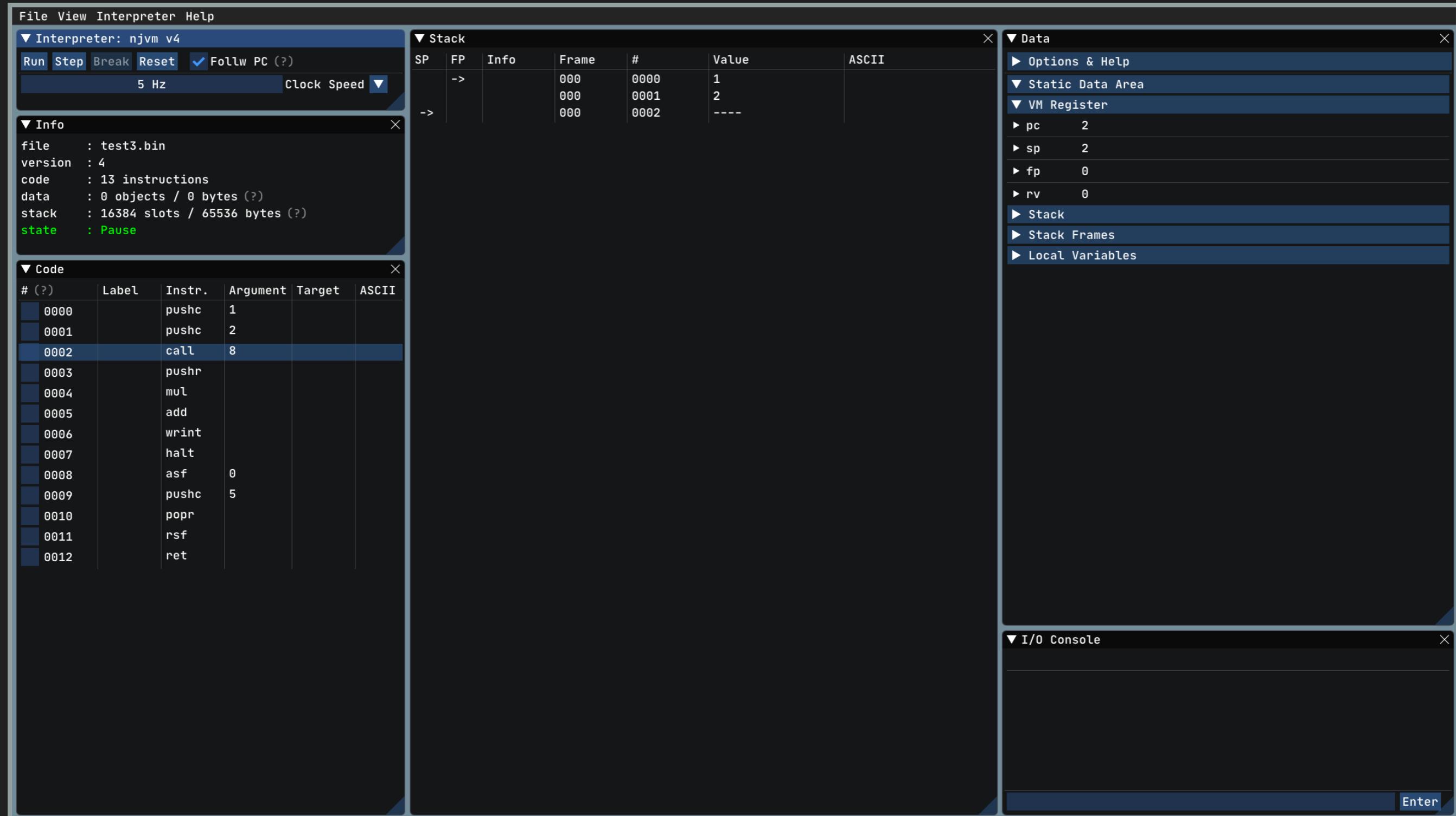
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



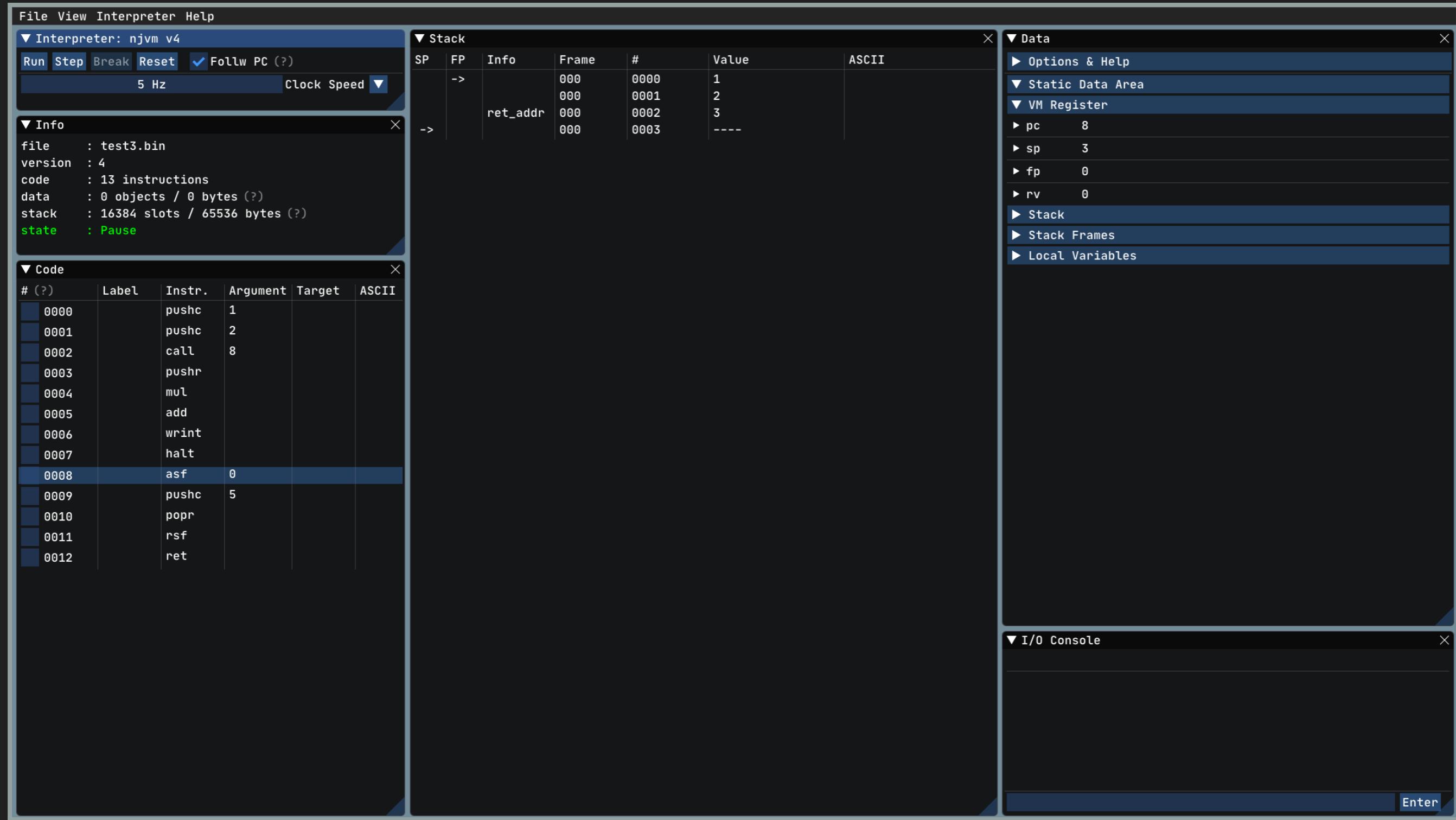
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



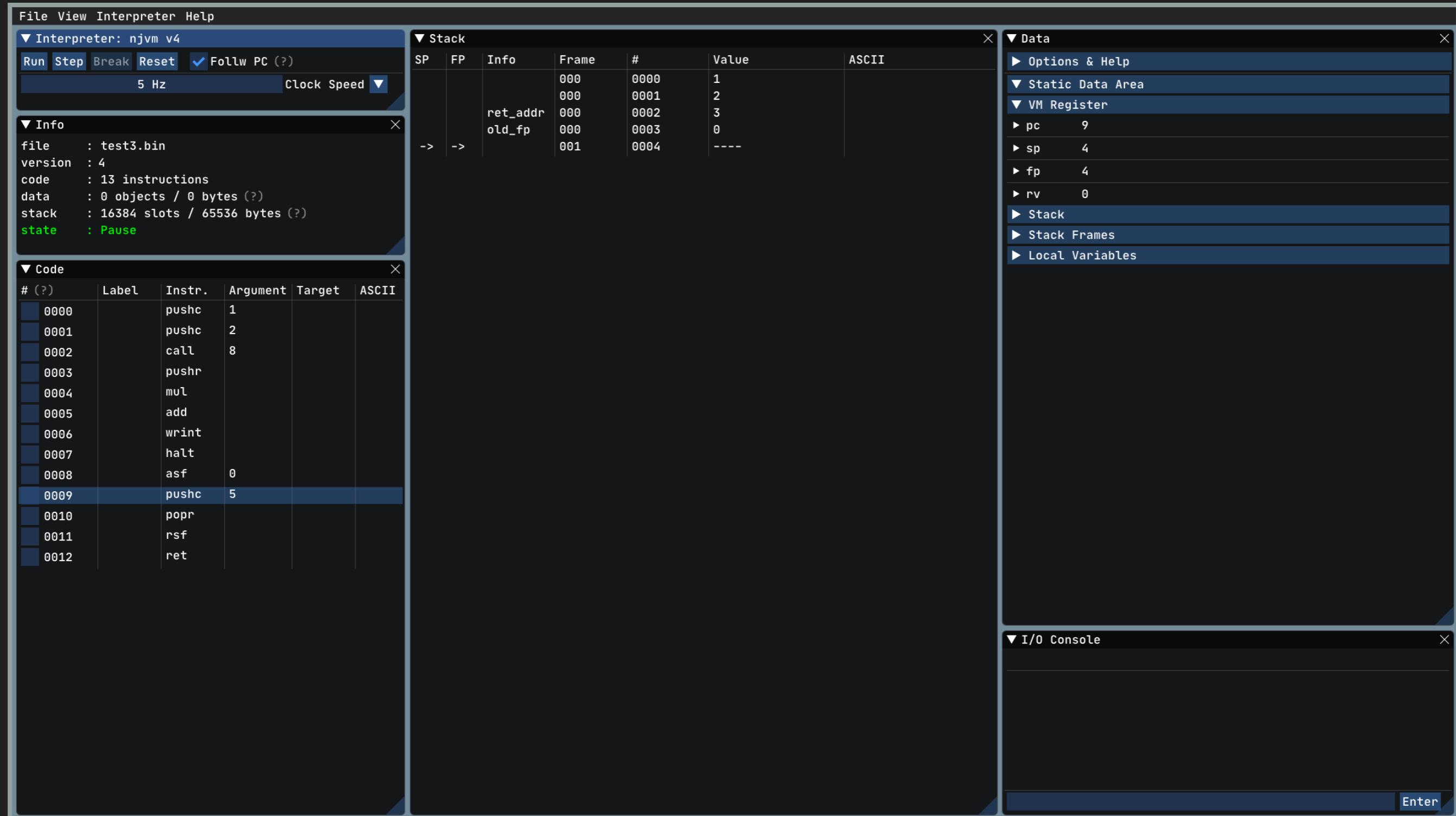
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



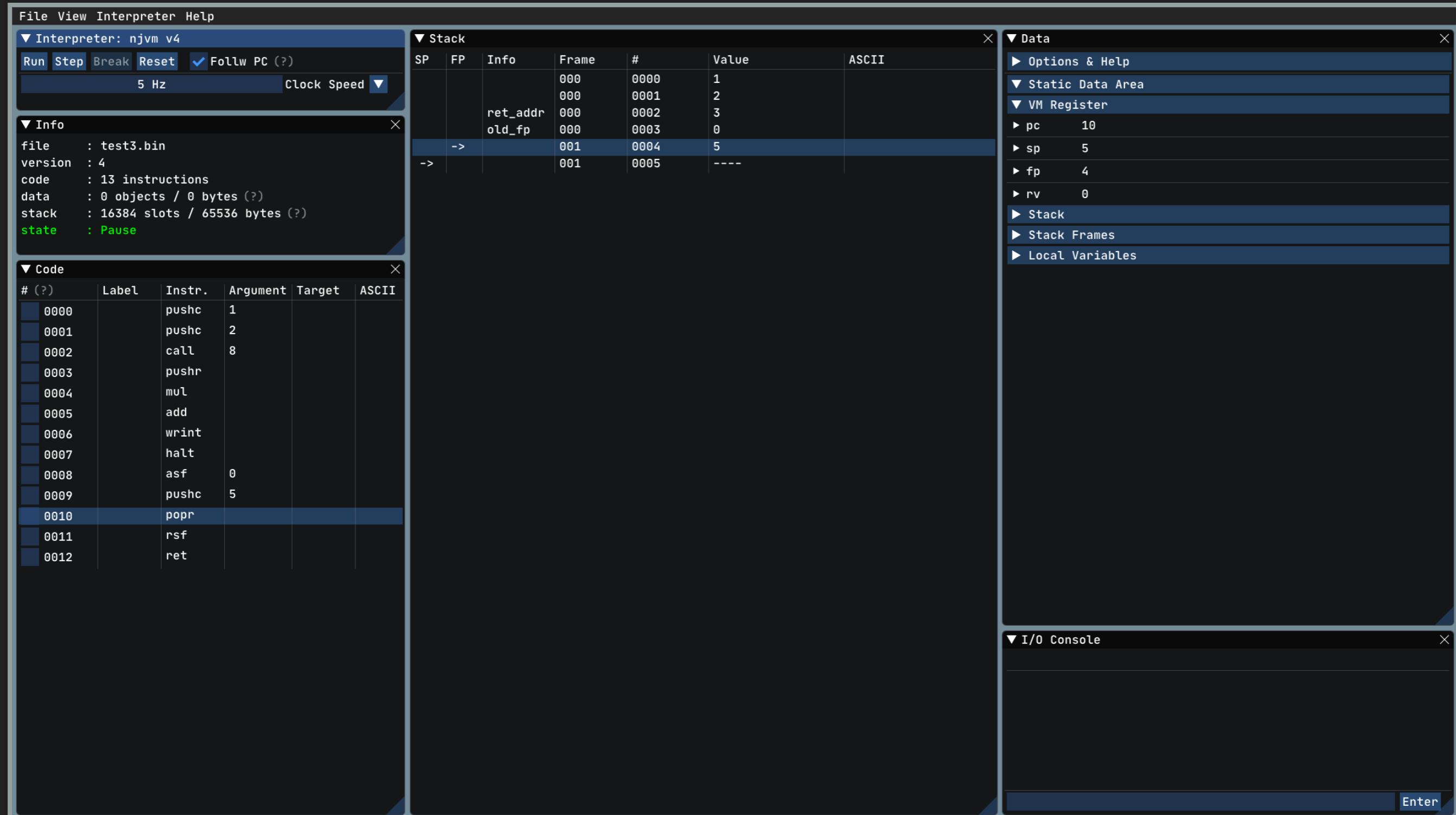
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



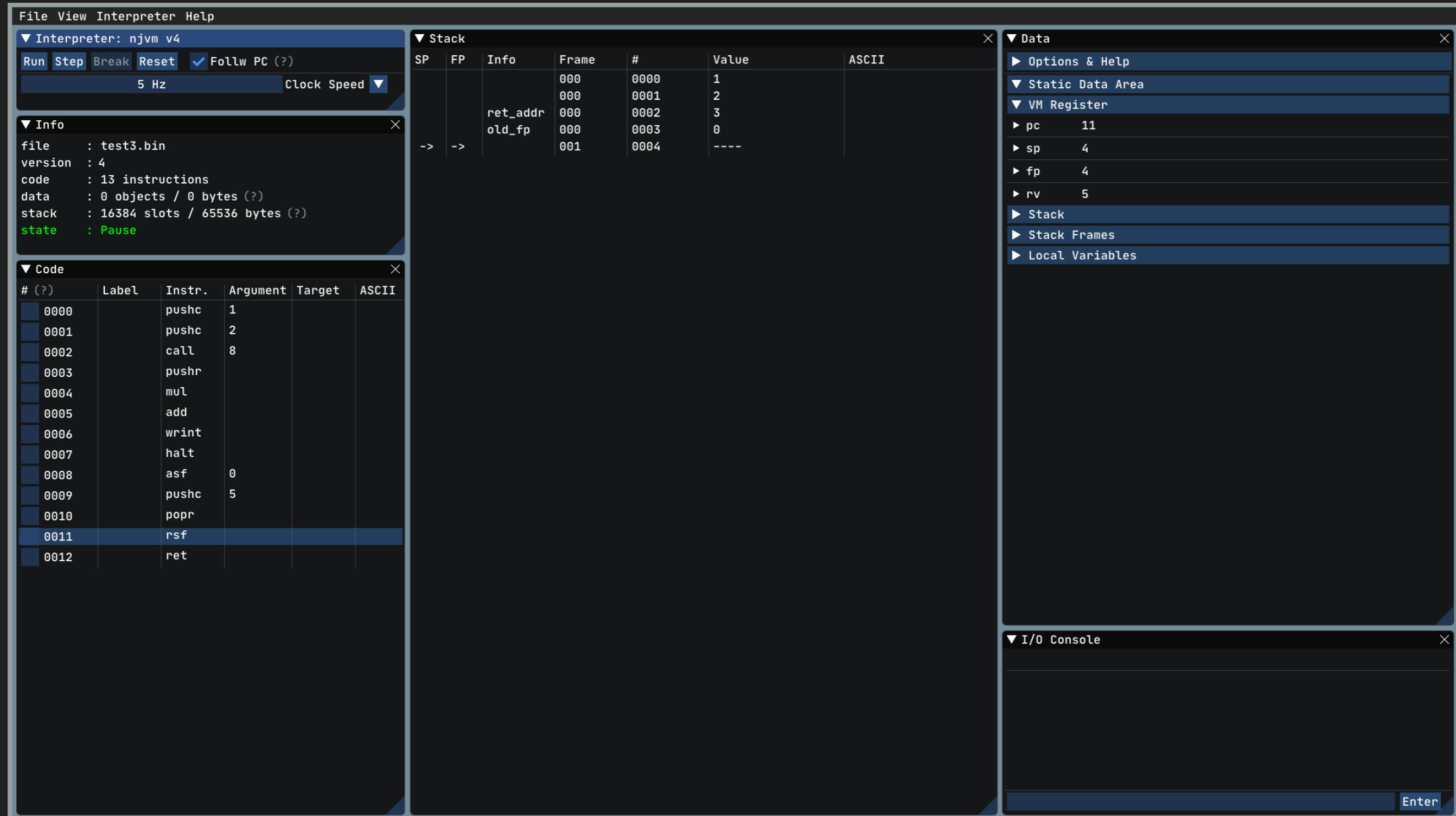
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



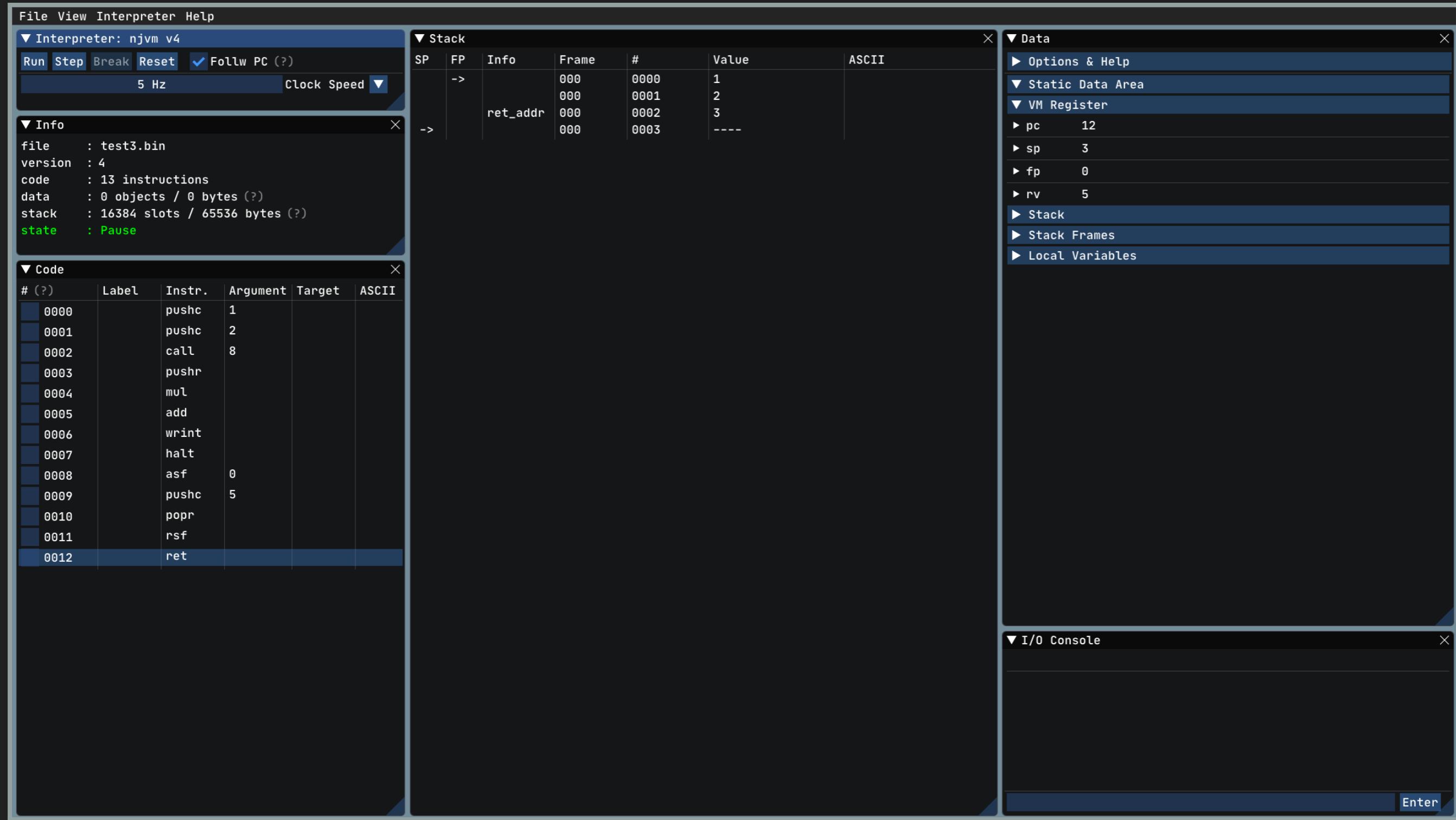
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



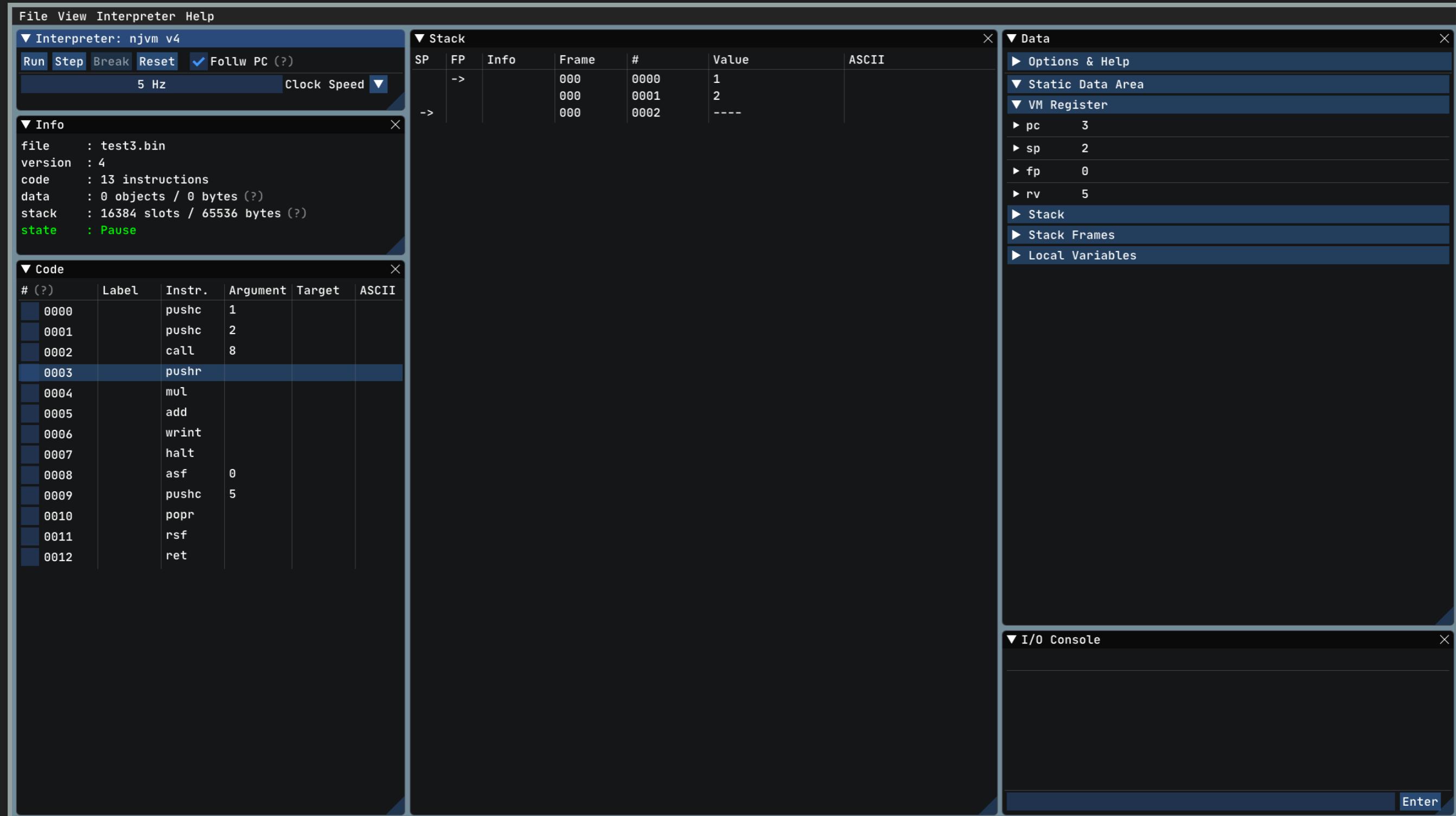
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



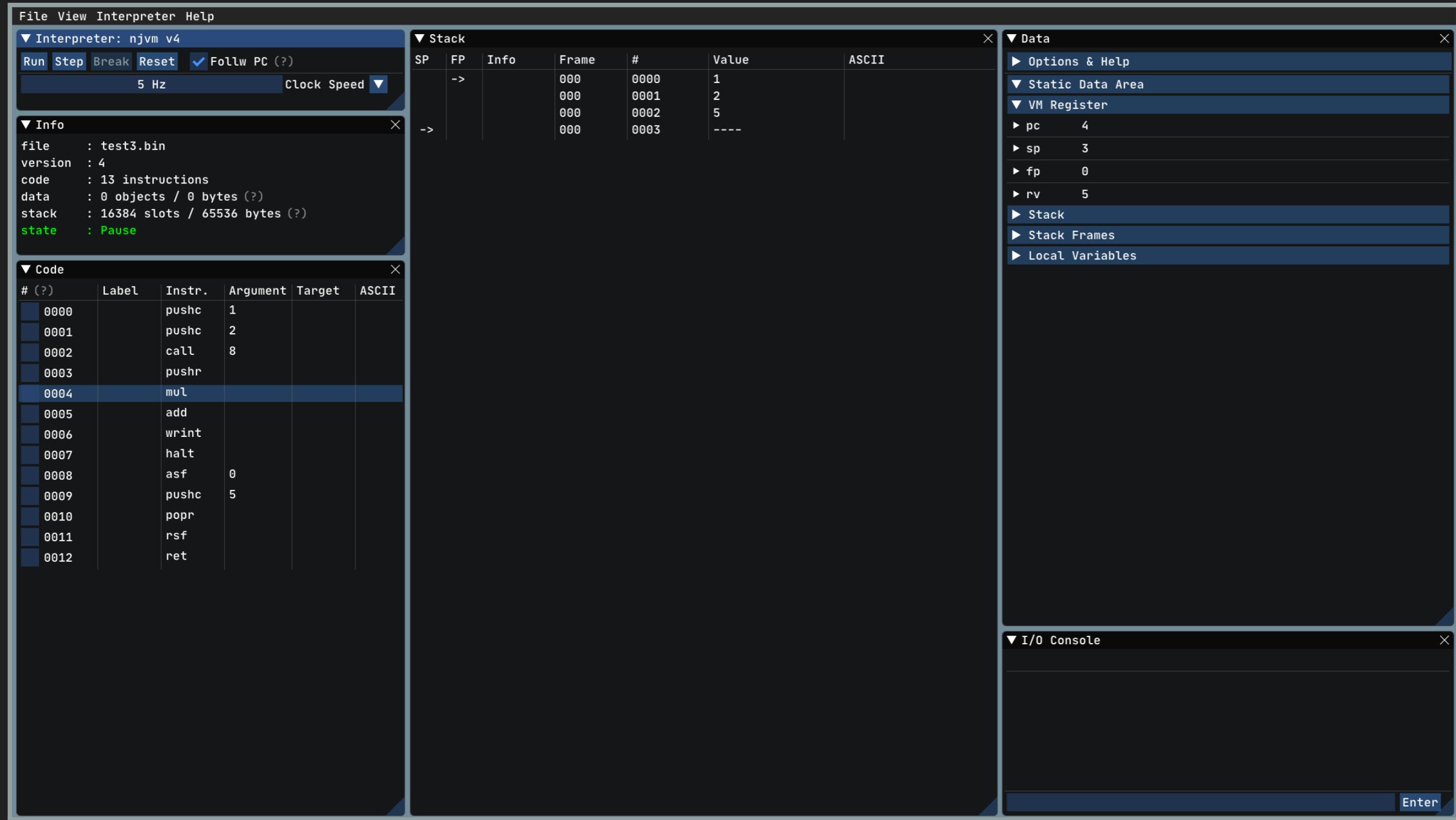
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT

The screenshot shows the njvm v4 debugger interface with several windows open:

- Interpreter: njvm v4**: Top-left window with tabs for Run, Step, Break, Reset, and Follow PC (checked). It also shows Clock Speed set to 5 Hz.
- Info**: Shows file: test3.bin, version: 4, code: 13 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), and state: Pause.
- Code**: A table showing assembly instructions from address 0000 to 0012. The columns are # (?), Label, Instr., Argument, Target, and ASCII. The instruction at address 0005 is highlighted.

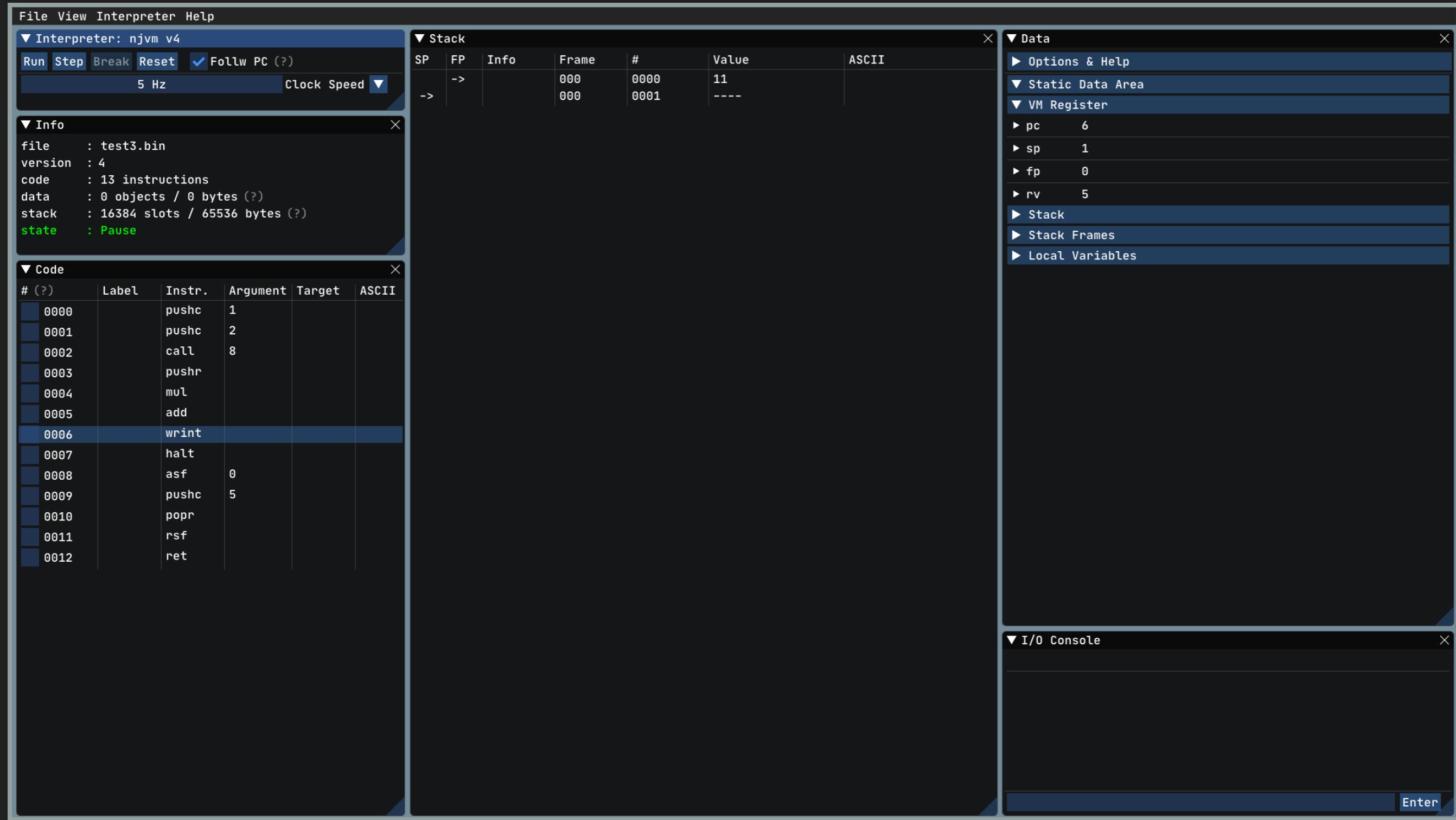
# (?)	Label	Instr.	Argument	Target	ASCII
0000		pushc	1		
0001		pushc	2		
0002		call	8		
0003		pushr			
0004		mul			
0005		add			
0006		wrint			
0007		halt			
0008		ASF	0		
0009		pushc	5		
0010		popr			
0011		rsf			
0012		ret			

- Stack**: A table showing the stack structure. The columns are SP, FP, Info, Frame, #, Value, and ASCII. The stack has three frames (Frame 000, 001, 002) with values 1, 10, and \n respectively.

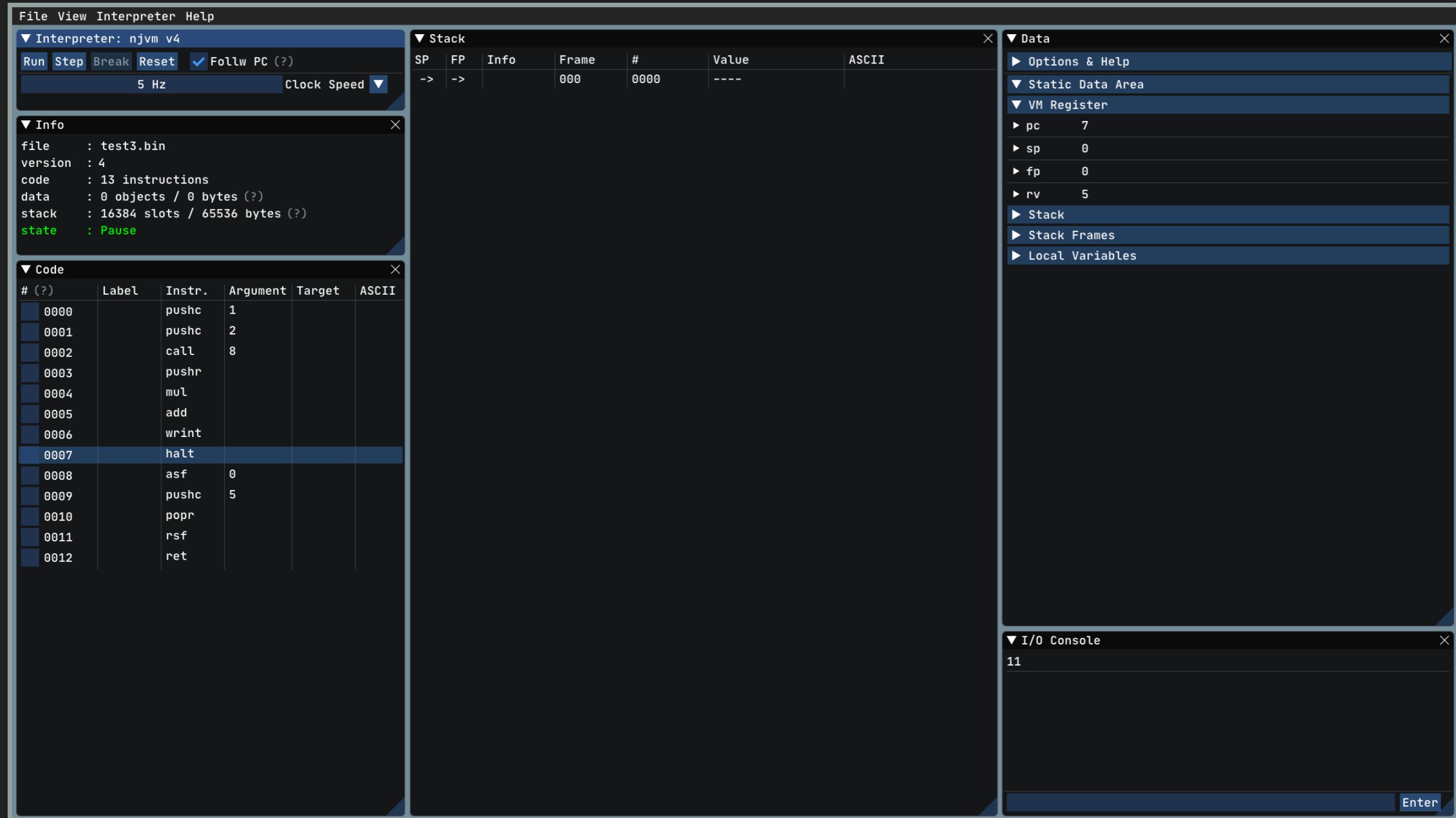
SP	FP	Info	Frame	#	Value	ASCII
->			000	0000	1	
->			000	0001	10	\n
->			000	0002	----	

- Data**: A tree view showing VM Register, Stack, Stack Frames, and Local Variables. The VM Register section shows pc: 5, sp: 2, fp: 0, and rv: 5.
- I/O Console**: Bottom-right window with an Enter button.

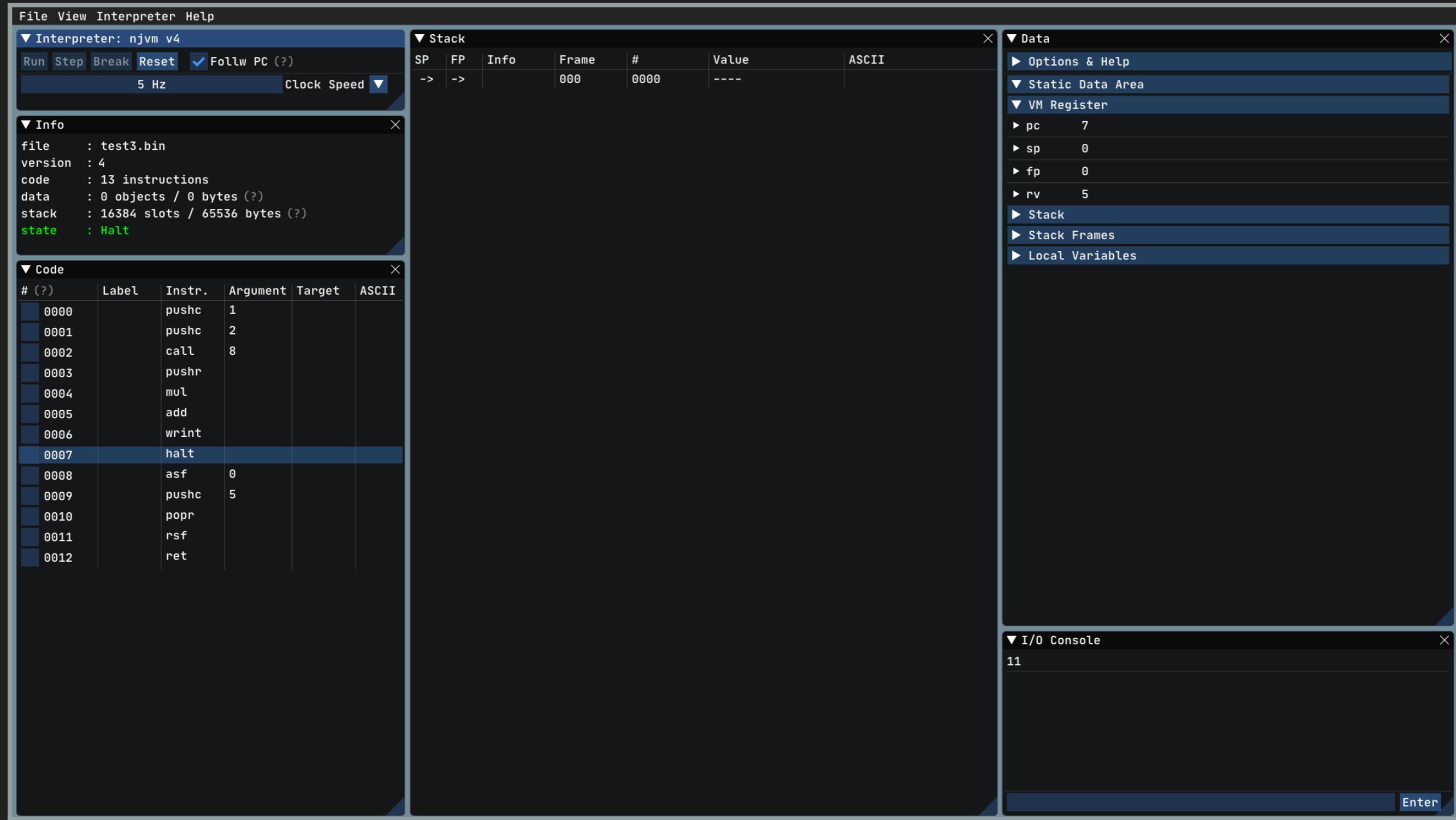
BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



BEISPIEL OHNE ARGUMENTEN, MIT RÜCKGABEWERT



FUNKTIONSAUFRUF MIT ARGUMENTEN, MIT RÜCKGABEWERT

Der Aufruf einer Funktion **mit Argumenten** und **mit einem Rückgabewert**, ist die Kombination der zuletzt gezeigten Aufrufe.

Caller:

```
push arg0  
push arg1  
...  
push arg(n-1)  
call Funktionslabel  
drop n  
pushr
```

Callee:

```
Funktionslabel:  
  asf <numLocalVars>  
  <body>  
  <push retval> // Vorbereitung für popr  
  popr  
  rsf  
  ret
```

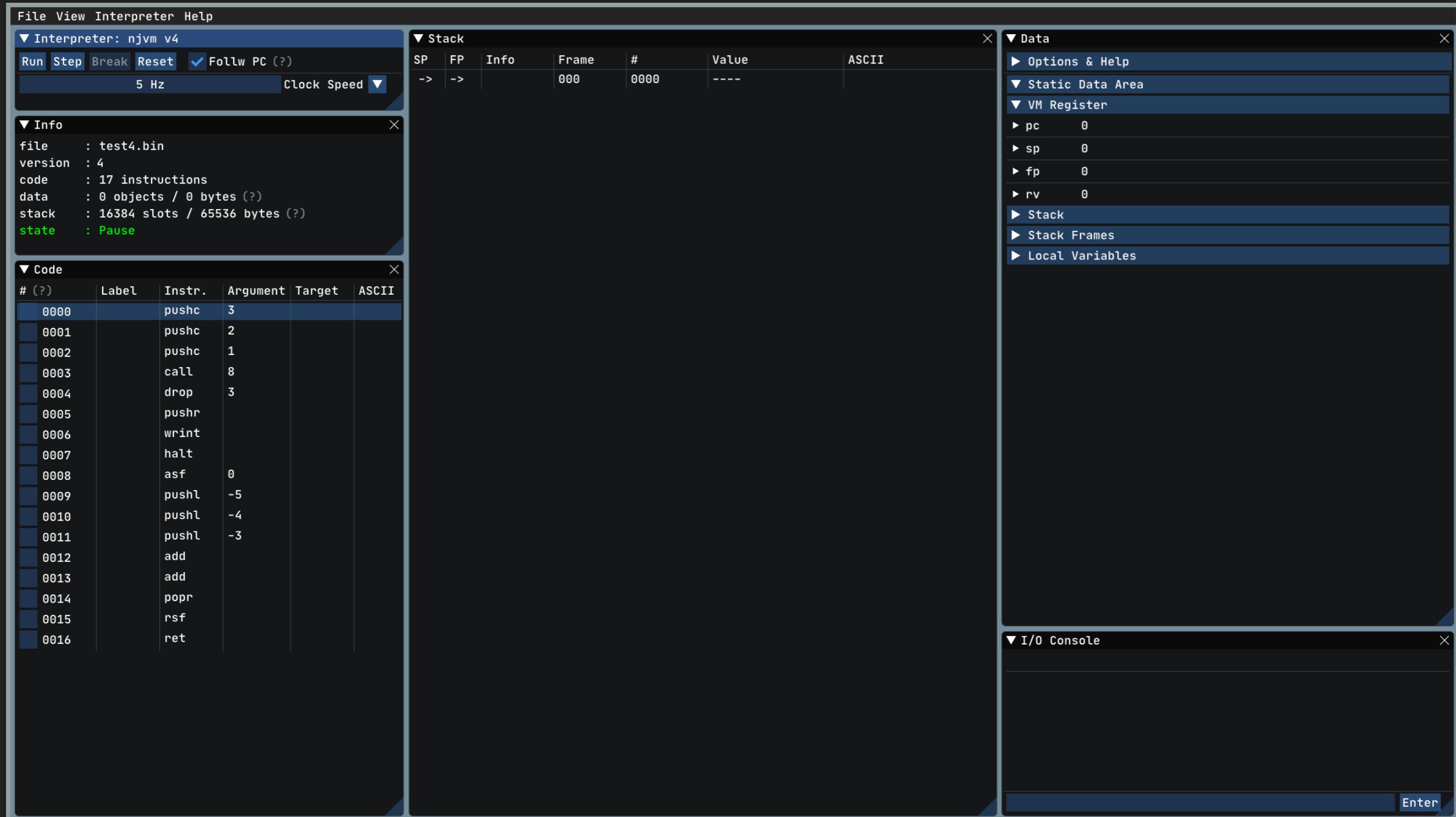
Zugriff auf Argumente erfolgt mit `pushl` und `pushl` mittels negativen Immediate Wert (`-2-n+i`) für `i`-tes Argument

BEISPIEL: FUNKTIONSAUFRUF MIT ARGUMENTEN UND RÜCKGABEWERT

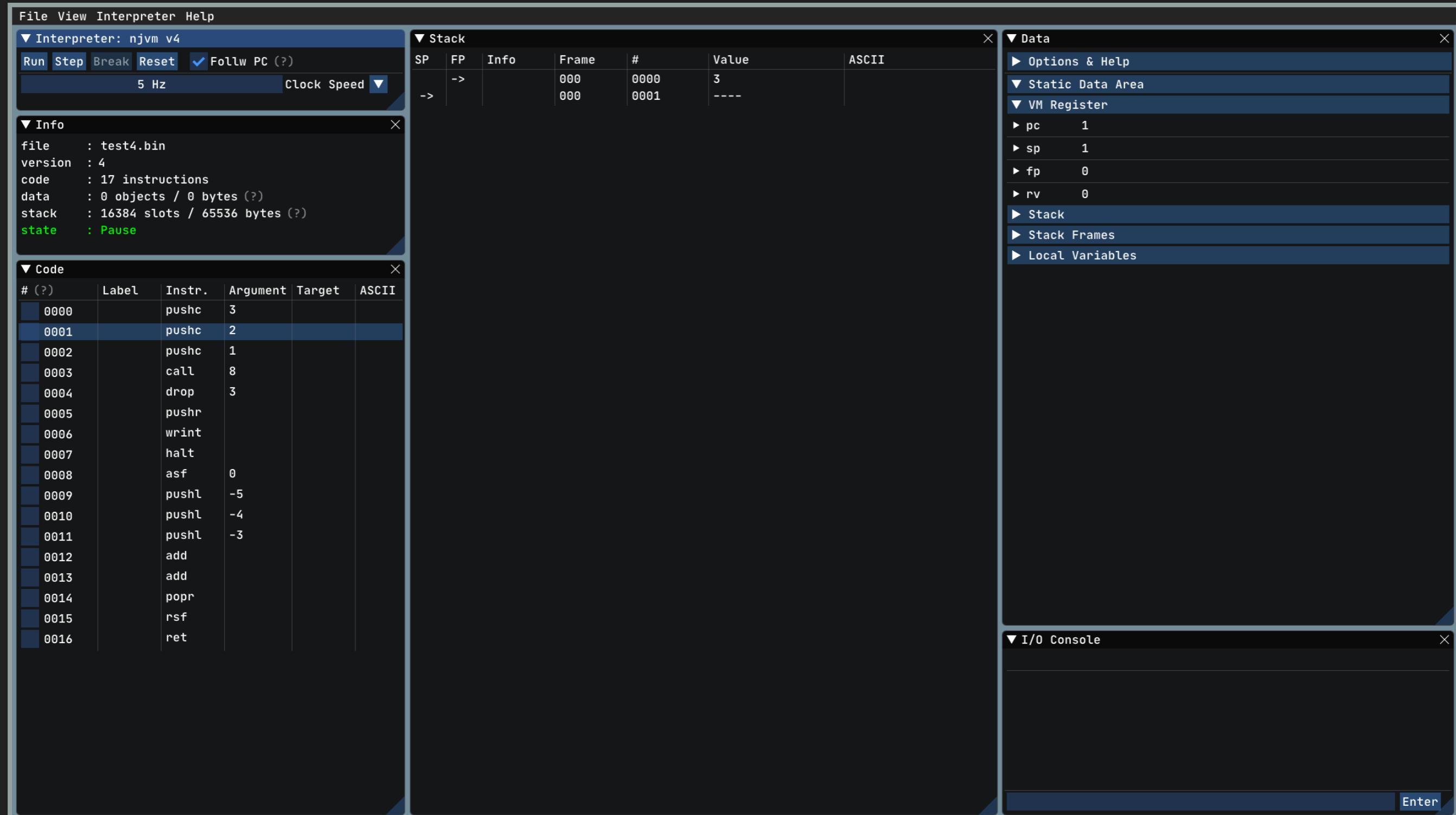
Beispiel: Ausgabe von `wrint f(3,2,1)` mit `int f(a,b,c) {return a+b+c;}`

```
pushc 3
pushc 2
pushc 1
call f
pushr
drop 3
wrint
halt
f:
    asf 0
    pushl -5
    pushl -4
    pushl -3
    add
    add
    popr
    rsf
    ret
```

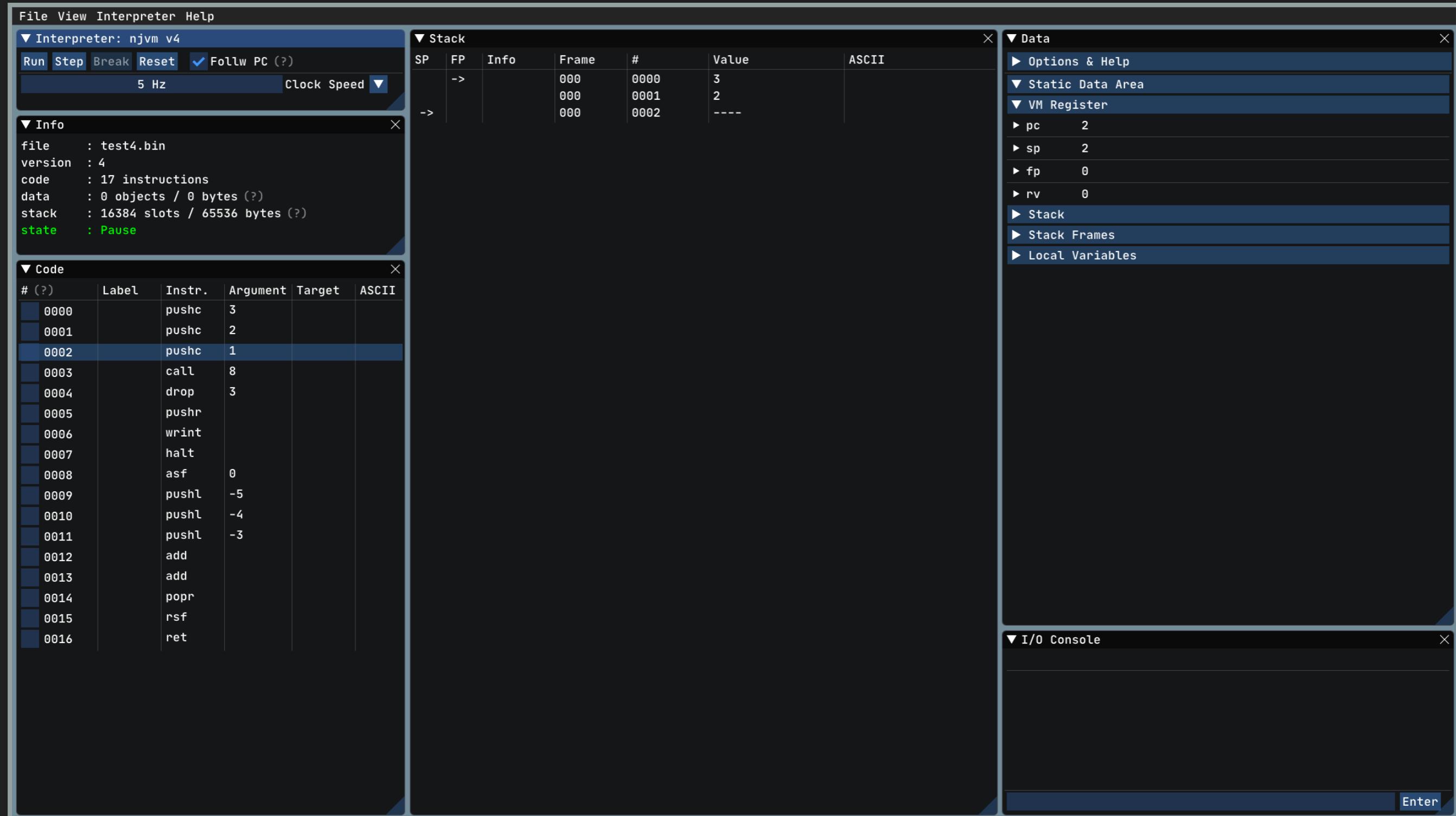
BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with the following panels:

- Interpreter: njvm v4**: Top navigation bar with Run, Step, Break, Reset, and Follow PC checkboxes. A dropdown for Clock Speed is set to 5 Hz.
- Info**: File details: file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, state: Pause.
- Code**: Instruction table with columns: # (?), Label, Instr., Argument, Target, ASCII. The instruction at address 0003 is highlighted: call 8.
- Stack**: Stack table with columns: SP, FP, Info, Frame, #, Value, ASCII. The stack contains four frames with values 3, 2, 1, and a null entry.
- Data**: VM Register table showing pc (3), sp (3), fp (0), and rv (0).
- I/O Console**: Bottom panel for input.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with the following sections:

- Interpreter: njvm v4**: Includes buttons for Run, Step, Break, Reset, and Follow PC (checked). A dropdown for Clock Speed is set to 5 Hz.
- Info**: Displays file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), and state: Pause.
- Code**: A table showing 17 instructions with columns: # (?), Label, Instr., Argument, Target, and ASCII. The instructions include pushc, pushr, call, drop, wrint, halt, asf, pushl, add, popr, rsf, and ret.
- Stack**: A table showing the stack structure with columns: SP, FP, Info, Frame, #, Value, and ASCII. The stack has frames at addresses 000 and 000 (ret_addr), with values 3, 2, 1, 4, and ---- respectively.
- Data**: A tree view showing Options & Help, Static Data Area, VM Register, Stack, Stack Frames, and Local Variables. Under VM Register, pc is 8, sp is 4, fp is 0, and rv is 0.
- I/O Console**: An empty text input field with an Enter button.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several windows open:

- Interpreter: njvm v4**: Top-left window with tabs for Run, Step, Break, Reset, and Follow PC. It also shows Clock Speed set to 5 Hz.
- Info**: Shows file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, and state: Pause.
- Code**: A table showing assembly-like code with columns for #, Label, Instr., Argument, Target, and ASCII. The row at index 9 is selected, showing pushl -5.
- Stack**: A table showing the stack structure with columns SP, FP, Info, Frame, #, Value, and ASCII. It includes entries for ret_addr, old_fp, and frame 001.
- Data**: A tree view showing VM Register, Stack, Stack Frames, and Local Variables. The pc register is set to 9.
- I/O Console**: Bottom-right window for input and output.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several panes:

- Interpreter: njvm v4**: Top-left pane with buttons: Run, Step, Break, Reset, Follow PC (?), Clock Speed (5 Hz).
- Info**: Top-right pane showing file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), state: Pause.
- Code**: Bottom-left pane showing assembly-like code:

# (?)	Label	Instr.	Argument	Target	ASCII
0000		pushc	3		
0001		pushc	2		
0002		pushc	1		
0003		call	8		
0004		drop	3		
0005		pushr			
0006		wrint			
0007		halt			
0008		ASF	0		
0009		pushl	-5		
0010		pushl	-4		
0011		pushl	-3		
0012		add			
0013		add			
0014		popr			
0015		rsf			
0016		ret			
- Stack**: Middle-right pane showing the stack structure:

SP	FP	Info	Frame	#	Value	ASCII
			000	0000	3	
			000	0001	2	
			000	0002	1	
		ret_addr	000	0003	4	
		old_fp	000	0004	0	
->			001	0005	3	
->			001	0006	----	
- Data**: Rightmost pane showing VM register values:

pc	sp	fp	rv
10	6	5	0
- I/O Console**: Bottom-right pane for input.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several panes:

- Interpreter: njvm v4**: Top-left pane with buttons: Run, Step, Break, Reset, Follow PC (?), Clock Speed (5 Hz).
- Info**: Top-right pane showing file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), state: Pause.
- Code**: Bottom-left pane showing assembly-like code:

# (?)	Label	Instr.	Argument	Target	ASCII
0000		pushc	3		
0001		pushc	2		
0002		pushc	1		
0003		call	8		
0004		drop	3		
0005		pushr			
0006		wrint			
0007		halt			
0008		ASF	0		
0009		pushl	-5		
0010		pushl	-4		
0011		pushl	-3		
0012		add			
0013		add			
0014		popr			
0015		rsf			
0016		ret			

- Stack**: Middle-right pane showing the stack structure:

SP	FP	Info	Frame	#	Value	ASCII
			000	0000	3	
			000	0001	2	
			000	0002	1	
		ret_addr	000	0003	4	
		old_fp	000	0004	0	
->			001	0005	3	
->			001	0006	2	
->			001	0007	---	

- Data**: Rightmost pane showing VM register values:

pc	11
sp	7
fp	5
rv	0

- I/O Console**: Bottom-right pane for input.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several panes:

- Interpreter: njvm v4**: Top-left pane with buttons: Run, Step, Break, Reset, Follow PC (?), Clock Speed (5 Hz).
- Info**: Top-right pane showing file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), state: Pause.
- Code**: Bottom-left pane listing assembly instructions:

# (?)	Label	Instr.	Argument	Target	ASCII
0000		pushc	3		
0001		pushc	2		
0002		pushc	1		
0003		call	8		
0004		drop	3		
0005		pushr			
0006		wrint			
0007		halt			
0008		ASF	0		
0009		pushl	-5		
0010		pushl	-4		
0011		pushl	-3		
0012		add			
0013		add			
0014		popr			
0015		rsf			
0016		ret			

- Stack**: Middle-right pane showing the stack structure:

SP	FP	Info	Frame	#	Value	ASCII
			000	0000	3	
			000	0001	2	
			000	0002	1	
		ret_addr	000	0003	4	
		old_fp	000	0004	0	
->			001	0005	3	
->			001	0006	2	
->			001	0007	1	
->			001	0008	----	

- Data**: Right-side pane showing VM register values:

pc	12
sp	8
fp	5
rv	0

- I/O Console**: Bottom-right pane for input.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several windows open:

- Interpreter: njvm v4**: Top-left window with tabs for Run, Step, Break, Reset, and Follow PC. It also shows Clock Speed set to 5 Hz.
- Info**: Shows file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, and state: Pause.
- Code**: A table showing assembly-like code with columns for #, Label, Instr., Argument, Target, and ASCII. The rows show various instructions from 0000 to 0016, including pushc, call, add, and ret.
- Stack**: A table showing the stack structure with columns SP, FP, Info, Frame, #, Value, and ASCII. The stack has frames 000, 001, and 002, with values 3, 2, 1 respectively. The current frame is 001.
- Data**: A tree view showing VM Register, Stack, Stack Frames, and Local Variables. Under VM Register, pc is 13, sp is 7, fp is 5, and rv is 0.
- I/O Console**: Bottom-right window for input and output.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several windows open:

- Interpreter: njvm v4**: Shows buttons for Run, Step, Break, Reset, and Follow PC. A dropdown for Clock Speed is set to 5 Hz.
- Info**: Displays file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, and state: Pause.
- Code**: A table showing assembly-like code with columns: # (?), Label, Instr., Argument, Target, and ASCII. The code consists of 16 instructions, starting with pushc and ending with ret.
- Stack**: A table showing the stack structure with columns: SP, FP, Info, Frame, #, Value, and ASCII. It includes entries for ret_addr, old_fp, and two frames (000 and 001) containing values 3, 2, 1, 4, 0, 6, and ----.
- Data**: A window showing VM Register values: pc (14), sp (6), fp (5), and rv (0). It also has sections for Options & Help, Static Data Area, Stack, Stack Frames, and Local Variables.
- I/O Console**: An empty window for input/output.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with the following sections:

- Interpreter: njvm v4**: Includes buttons for Run, Step, Break, Reset, and Follow PC. A dropdown for Clock Speed is set to 5 Hz.
- Info**: Displays file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, and state: Pause.
- Code**: A table showing assembly instructions. The current instruction at address 0015 is rsf, which has arguments 3, 2, and 1. Other instructions include pushc, call, drop, pushr, wrint, halt, asf, pushl, add, popr, and ret.
- Stack**: A table showing the stack structure. It includes columns for SP, FP, Info, Frame, #, Value, and ASCII. The stack contains values 3, 2, 1, 4, 0, and a separator dash. Local variables are labeled ret_addr, old_fp, and 001.
- Data**: A section showing VM Register values: pc (15), sp (5), fp (5), and rv (6). It also includes links to Options & Help, Static Data Area, VM Register, Stack, Stack Frames, and Local Variables.
- I/O Console**: An empty text input field with an Enter button.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

The screenshot shows the njvm v4 interpreter interface with several windows open:

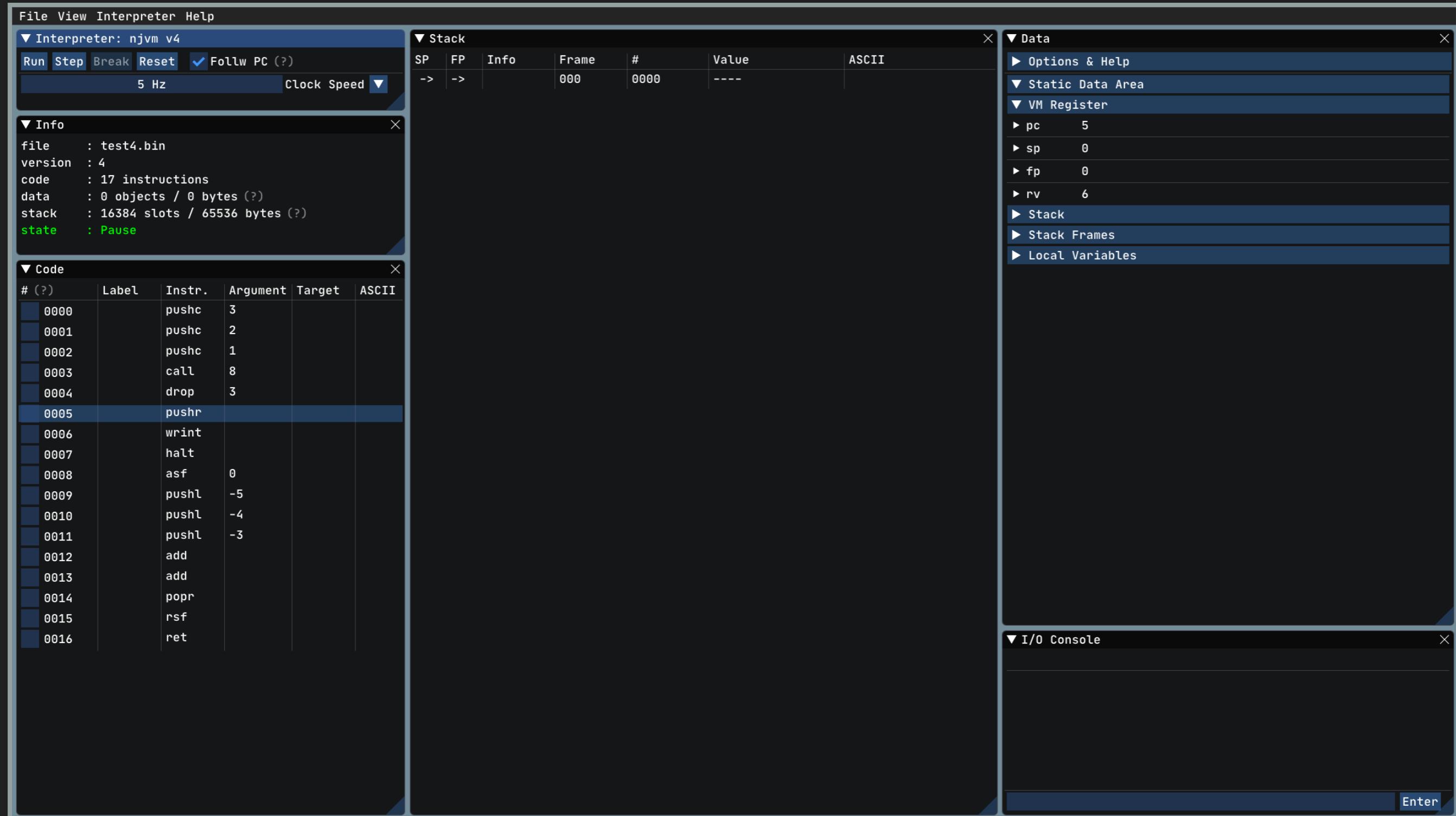
- Interpreter: njvm v4**: Shows buttons for Run, Step, Break, Reset, and Follow PC. A dropdown for Clock Speed is set to 5 Hz.
- Info**: Displays file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes, stack: 16384 slots / 65536 bytes, and state: Pause.
- Code**: A table showing 17 instructions with columns for #, Label, Instr., Argument, Target, and ASCII. The instructions include pushc, pushr, call, drop, wrint, halt, asf, pushl, add, popr, rsf, and ret.
- Stack**: A table showing the stack with columns SP, FP, Info, Frame, #, Value, and ASCII. The stack has frames 000 and 000, with values 3, 2, 1, 4, and ---- respectively. A ret_addr entry is also present.
- Data**: A tree view showing Options & Help, Static Data Area, VM Register, Stack, Stack Frames, and Local Variables. Under VM Register, pc is 16, sp is 4, fp is 0, and rv is 6.
- I/O Console**: An empty text input field with an Enter button.

BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT

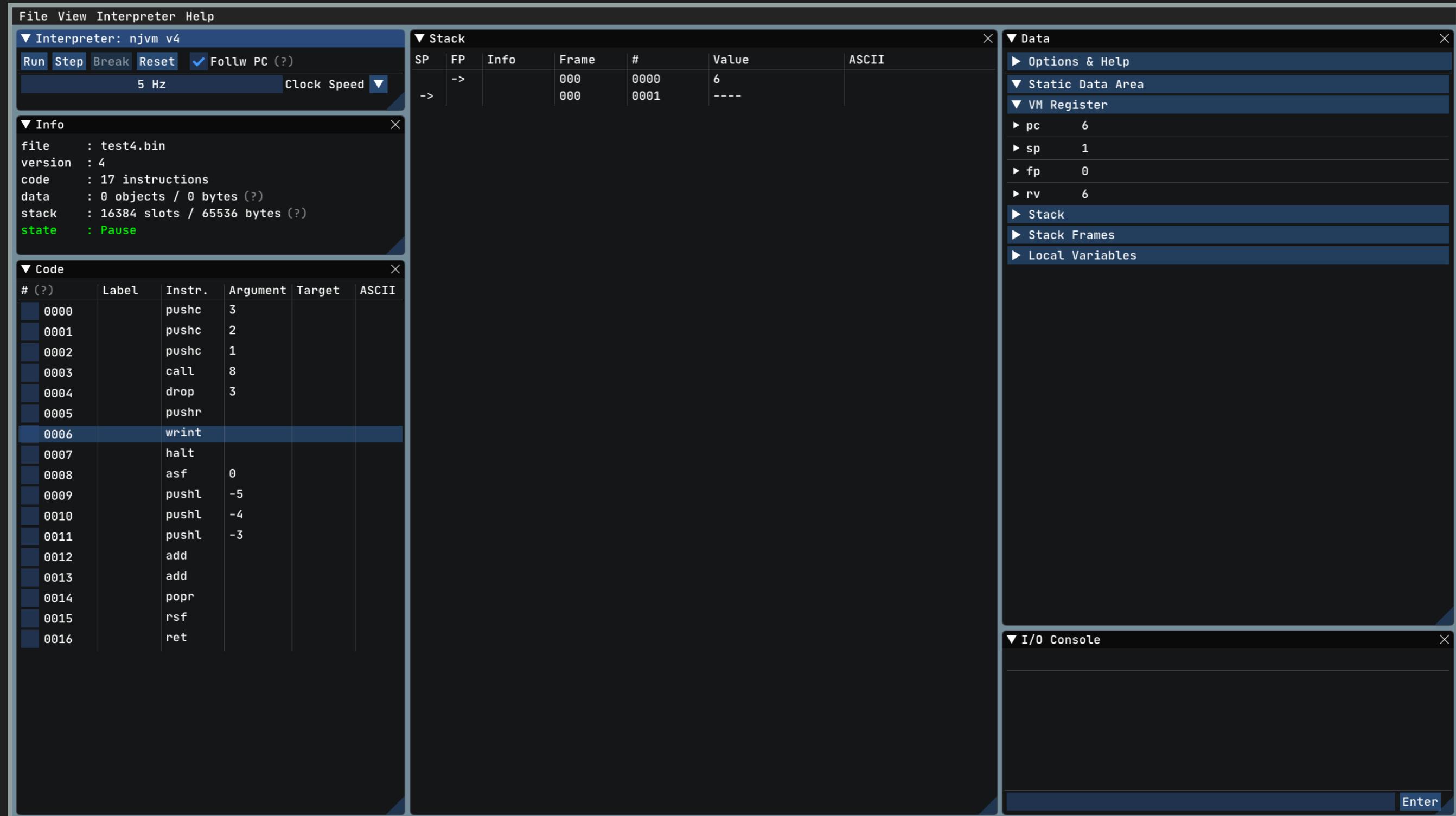
The screenshot shows the njvm v4 interpreter interface with the following panels:

- Interpreter: njvm v4**: Top-left panel with buttons: Run, Step, Break, Reset, Follow PC (?), Clock Speed (5 Hz).
- Info**: Panel showing file: test4.bin, version: 4, code: 17 instructions, data: 0 objects / 0 bytes (?), stack: 16384 slots / 65536 bytes (?), state: Pause.
- Code**: Panel showing assembly code with columns: # (?), Label, Instr., Argument, Target, ASCII. The instruction at address 0004 is highlighted: drop 3.
- Stack**: Panel showing the stack structure with columns: SP, FP, Info, Frame, #, Value, ASCII. The stack contains four frames with values 3, 2, 1, and a null-terminated string.
- Data**: Right-hand panel showing VM Register values: pc=4, sp=3, fp=0, rv=6. Sub-sections include Options & Help, Static Data Area, VM Register, Stack, Stack Frames, and Local Variables.
- I/O Console**: Bottom-right panel for input and output.

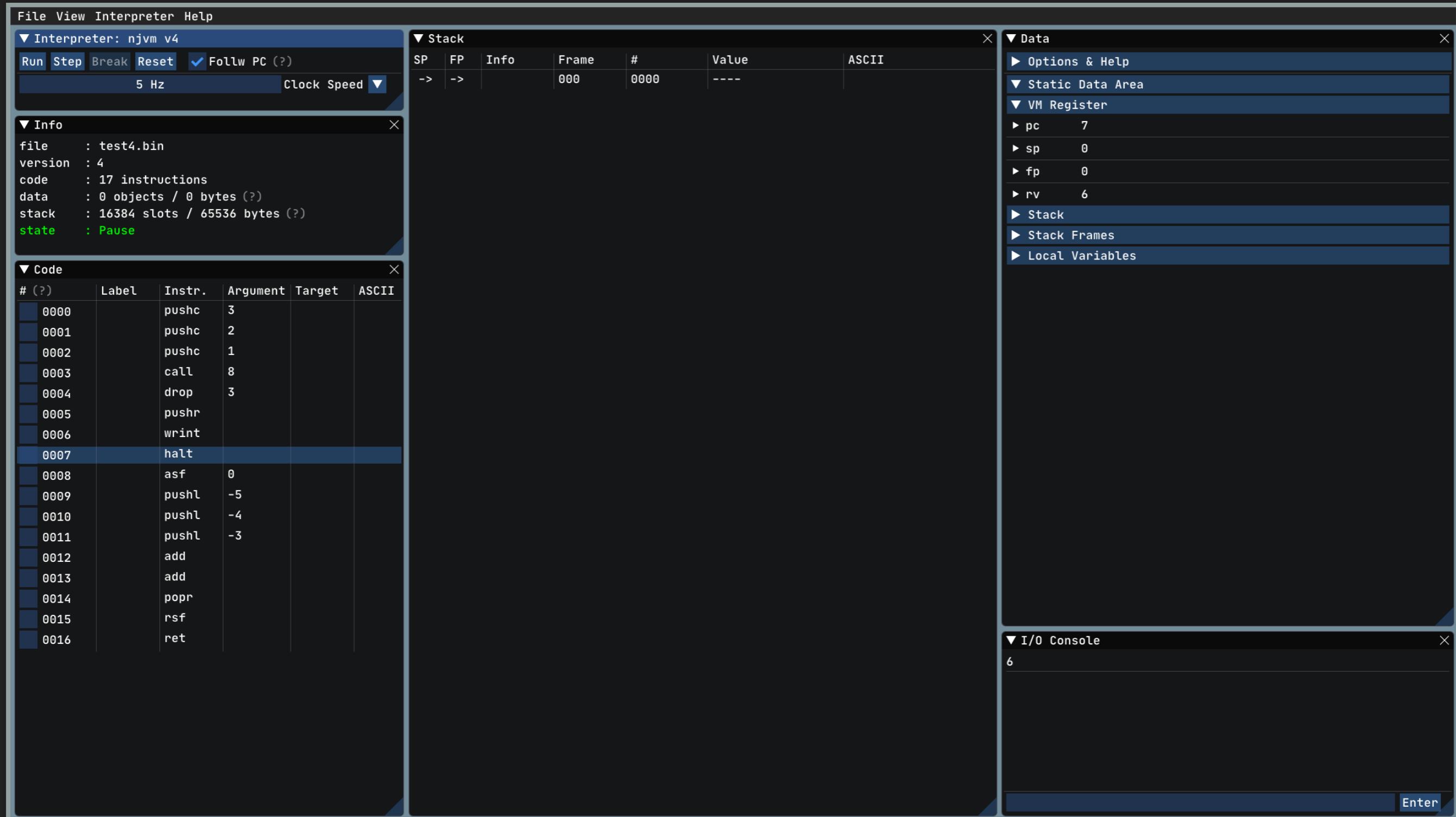
BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT



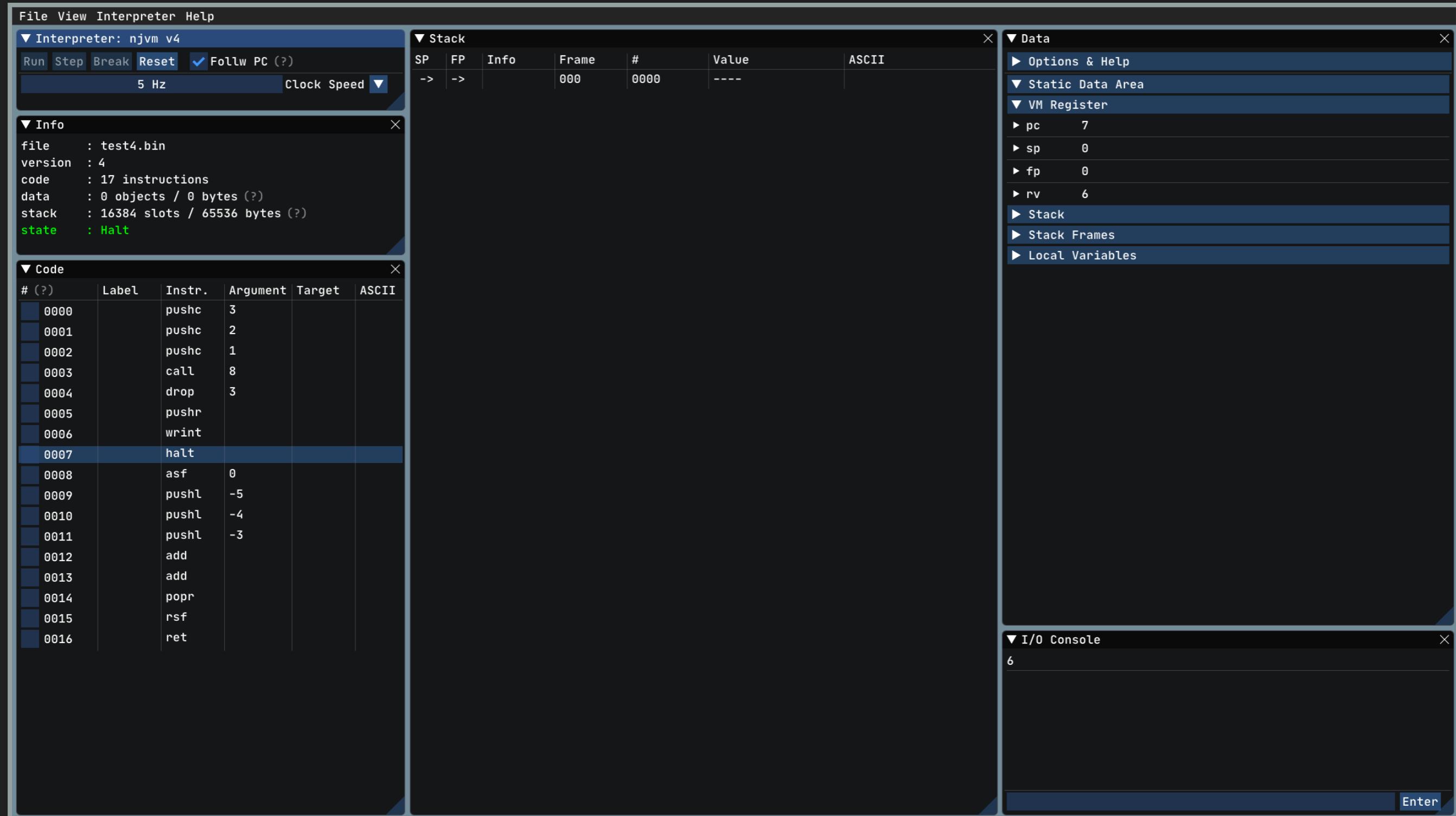
BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT



BEISPIEL MIT ARGUMENTEN UND RÜCKGABEWERT



ÜBERSICHT NINJAVM

