universität freiburg

Development of an Educational Tool for ReTI as a VS Code Extension

Bachelorproject for the B. Sc. Informatik at University of Freiburg

Presenter: Malte Pullich

Chair of Computer Architecture

Examiner: Prof. Dr. Armin Biere
Adviser: M. Sc. Tobias Faller
Freiburg, 16.07.2025

Agenda

1. **Motivation:** State of the Art and Usage
2. **Background:** ReTI Architecture and Language
3. **Approach:** Improving on Interactivity and Adding Features
4. **Results:** Improvement on existing Features, new Features
5. **Conclusion:** Goals Reached & Future Work

Motivation

```
root@Malte-Laptop:/mnt/c/BachelorprojektBiere/reticode# vim mul.reti
root@Malte-Laptop:/mnt/c/BachelorprojektBiere/reticode# ./asreti mul.reti | ./emreti -s
STEPS    PC       CODE     IN1      IN2      ACC      INSTRUCTION  ACTION
1        00000000 71000001 00000000 00000000 00000000 LOADI IN1 1  IN1 = 0x1
2        00000001 72000001 00000001 00000000 00000000 LOADI IN2 1  IN2 = 0x1
3        00000002 bb000000 00000001 00000001 00000000 MOVE IN2 ACC ACC = IN2 = 0x1
4        00000003 f000000a 00000001 00000001 00000001 JUMP<= 10    no jump as 1 = [0x1] = ACC > 0
5        00000004 b7000000 00000001 00000001 00000001 MOVE IN1 ACC ACC = IN1 = 0x1
6        00000005 2f000001 00000001 00000001 00000001 ADD ACC 1    ACC = ACC + M(<0x1>) = ACC + [0x0] = 1 + 0 = 1 = [0x1]
emreti: warning: continuing after reading uninitialized 'data[0x1]' (use '-i' so squelch such messages, or '-g' to stop)
7        00000006 80000001 00000001 00000001 00000001 STORE 1      M(<1>) = M(0x1) = 0x1
8        00000007 43000000 00000001 00000001 00000001 LOAD ACC 0   ACC = M(<0x0>) = M(0x0) = 0x0
emreti: warning: continuing after reading uninitialized 'data[0x0]' (use '-i' so squelch such messages, or '-g' to stop)
9        00000008 0f000001 00000001 00000001 00000000 ADDI ACC 1   ACC = ACC + [0x1] = 0 + 1 = 1 = [0x1]
10       00000009 80000000 00000001 00000001 00000001 STORE 0      M(<0>) = M(0x0) = 0x1
11       0000000a bb000000 00000001 00000001 00000001 MOVE IN2 ACC ACC = IN2 = 0x1
12       0000000b 2b000000 00000001 00000001 00000001 SUB ACC 0    ACC = ACC - M(<0x0>) = ACC - [0x1] = 1 - 1 = 0 = [0x0]
13       0000000c f8fffff7 00000001 00000001 00000000 JUMP -9      PC = PC + [0xffffff7] = 12 - 9 = 3 = 0x3
14       00000003 f000000a 00000001 00000001 00000000 JUMP<= 10    PC = PC + [0xa] = 3 + 10 = 13 = 0xd as 0 = [0x0] = ACC <= 0
15       0000000d f8000000 00000001 00000001 00000000 JUMP 0       PC = PC + [0x0] = 13 + 0 = 13 = 0xd
15       0000000d f8000000 00000001 00000001 00000000 <infinite-loop>
ADDRESS  DATA     BYTES       ASCII  UNSIGNED       SIGNED
00000000 00000001 01 00 00 00 ....           1            1
00000001 00000001 01 00 00 00 ....           1            1
```

Fig. 1: Screenshot of running an example ReTI-Program with Emulator by Prof. Dr. Biere [1]

Goal: Improve accessibility and responsiveness to provide a better educational experience.

ReTi-Architecture

```
JUMP<= 12
    LOAD IN1 2
    LOAD ACC 3
    STORE 2
    ADD IN1 3
    MOVE IN1 ACC
    STORE 3
    ADDI ACC 1
    STORE 1
    ; Check if i <
    LOAD ACC 0
    SUB ACC 1
JUMP -11
NOP
```

Fig: 2 Excerpt of a ReTI example program that calculates the Fibonacci number of 5

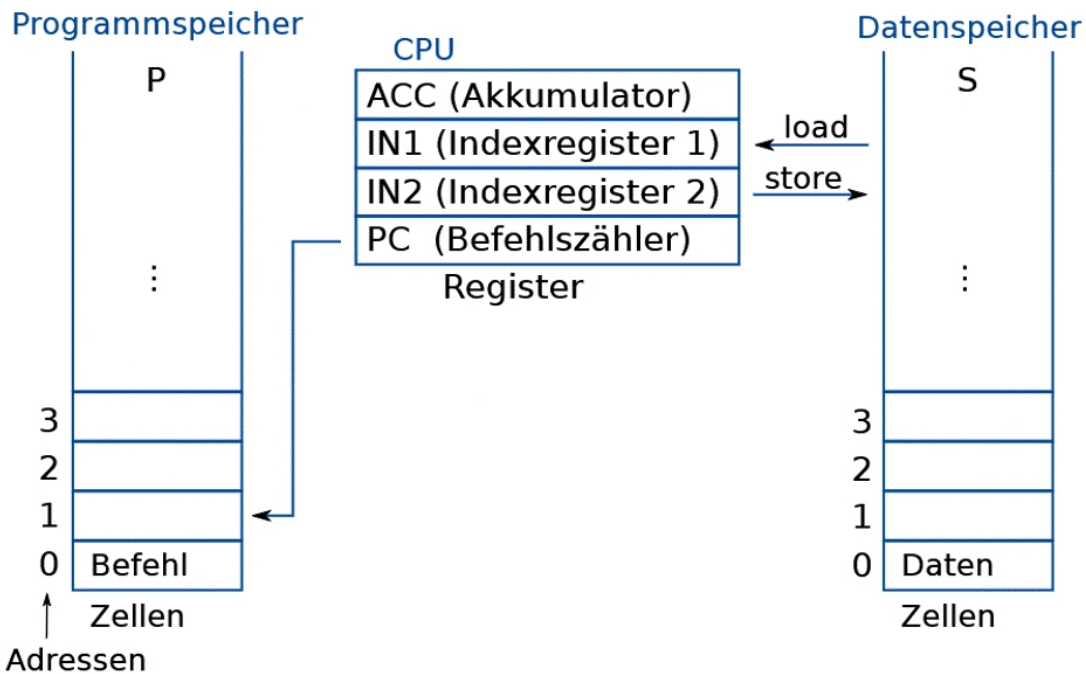


Fig 3: ReTI architecture [2]

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Quiz:

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Quiz:

- Rebuild a Website for the Quiz

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Quiz:

- Rebuild a Website for the Quiz
- Add Explanation to Result for each Question

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Quiz:

- Rebuild a Website for the Quiz
- Add Explanation to Result for each Question
- VS Code Webview →Easy integration of Quiz-Website

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Quiz:

- Rebuild a Website for the Quiz
- Add Explanation to Result for each Question
- VS Code Webview →Easy integration of Quiz-Website

Additional features:

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Quiz:

- Rebuild a Website for the Quiz
- Add Explanation to Result for each Question
- VS Code Webview →Easy integration of Quiz-Website

Additional features:

Debugger:

- Improves programming flow
- Enables experimentation
- Microsoft Debug-Adapter-Protocol

Approach

Goal: Improve accessibility and responsiveness to provide a better educational experience.

Build upon current features:

Implement as VS Code Extension:

- Most used IDE
- Integrate Emulator, Assembler, Disassembler
- VS Code Extension API

Quiz:

- Rebuild a Website for the Quiz
- Add Explanation to Result for each Question
- VS Code Webview →Easy integration of Quiz-Website

Additional features:

Debugger:

- Improves programming flow
- Enables experimentation
- Microsoft Debug-Adapter-Protocol

Language Server:

- Syntax Highlighting, Inline errors, Tooltips
- Microsoft Language-Server-Protocol

Results

Integration of Old Features

```
root@Malte-Laptop:/mnt/c/BachelorprojektBiere/reticode# vim mul.r
eti
root@Malte-Laptop:/mnt/c/BachelorprojektBiere/reticode# ./asreti
mul.reti | ./emreti -s
STEPS    PC       CODE     IN1      IN2      ACC      INSTRUCTION
  ACTION
1        00000000 71000001 00000000 00000000 00000000 LOADI IN1 1
  IN1 = 0x1
2        00000001 72000001 00000001 00000000 00000000 LOADI IN2 1
  IN2 = 0x1
3        00000002 bb000000 00000001 00000001 00000000 MOVE IN2 AC
C ACC = IN2 = 0x1
4        00000003 f000000a 00000001 00000001 00000001 JUMP<= 10
  no jump as 1 = [0x1] = ACC > 0
5        00000004 b7000000 00000001 00000001 00000001 MOVE IN1 AC
C ACC = IN1 = 0x1
6        00000005 2f000001 00000001 00000001 00000001 ADD ACC 1
  ACC = ACC + M(<0x1>) = ACC + [0x0] = 1 + 0 = 1 = [0x1]
```

Fig 3: Example Usage of the ReTI Emulation by Prof. Dr. Biere [1]

```
15       0000000d f8000000 00000001 00000001 00000000 JUMP 0
  PC = PC + [0x0] = 13 + 0 = 13 = 0xd
15       0000000d f8000000 00000001 00000001 00000000 <infinite-l
oop>
ADDRESS  DATA     BYTES       ASCII  UNSIGNED       SIGNED
00000000 00000001 01 00 00 00 ....          1            1
00000001 00000001 01 00 00 00 ....          1            1
```

Fig 4: Output of the ReTI Emulation by Prof. Dr. Biere [1]

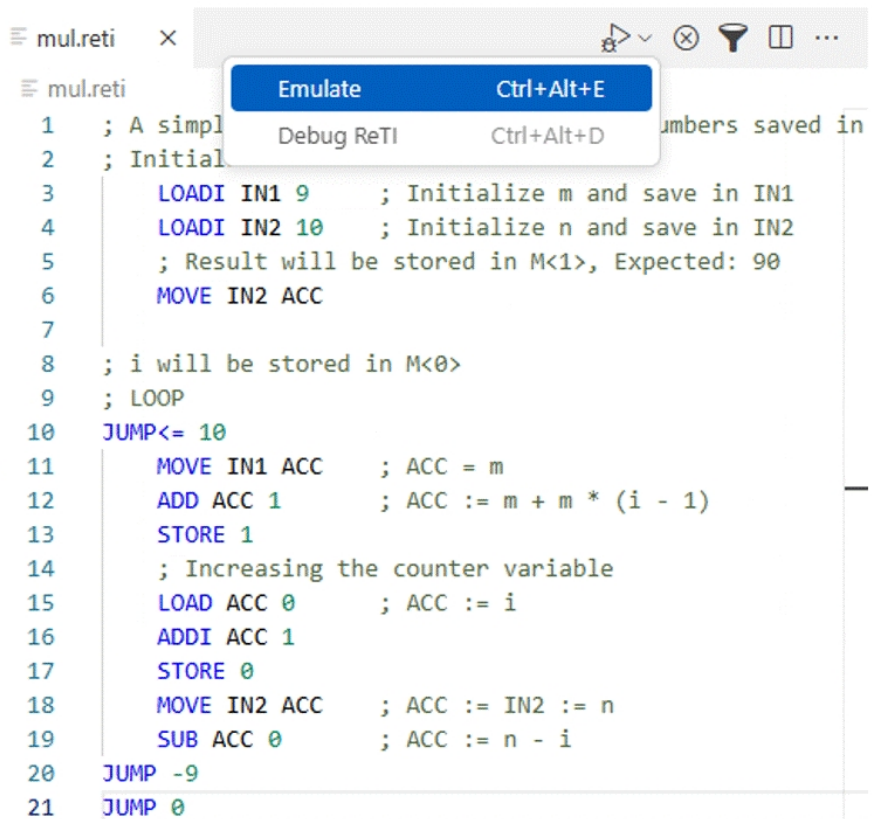


Fig 5: Example Usage of the ReTI Emulation in VS Code

```
Emulation finished.
PC: 20
IN1: 5
IN2: 3
ACC: 268435455

Data:
0: 73000005
1: 80000000
2: 73000002
3: 80000001
4: 73000001
5: 80000003
6: 72000003
7: f000000c
8: 41000002
9: 43000003
10: 80000002
11: 2d000003
```

Fig 3: Output of Emulation in VS Code

Results

Integration of Old Features - Quiz

```
ReTI Machine Code Quiz Version 0.0.3-rc.3
retiquiz 69740141930056169777 16
Enter hexadecimal digits as an answer or
space ' ' to skip a question or 'q' to qui
t.
For irrelevant '*' in the machine code use
 '0'.
Asking 16 questions.
INSTRUCTION              ; PC       CODE
LOADIN1 PC 10            ; 00000000 5000000a ✓
SUB IN1 -18              ; 00000001 29ffffe ✗
 expected e in 29ffffee at I[7:4]
STORE 8                  ; 00000002 8000000
```

Fig 6: Screenshot of the old ReTI Quiz opened in Terminal

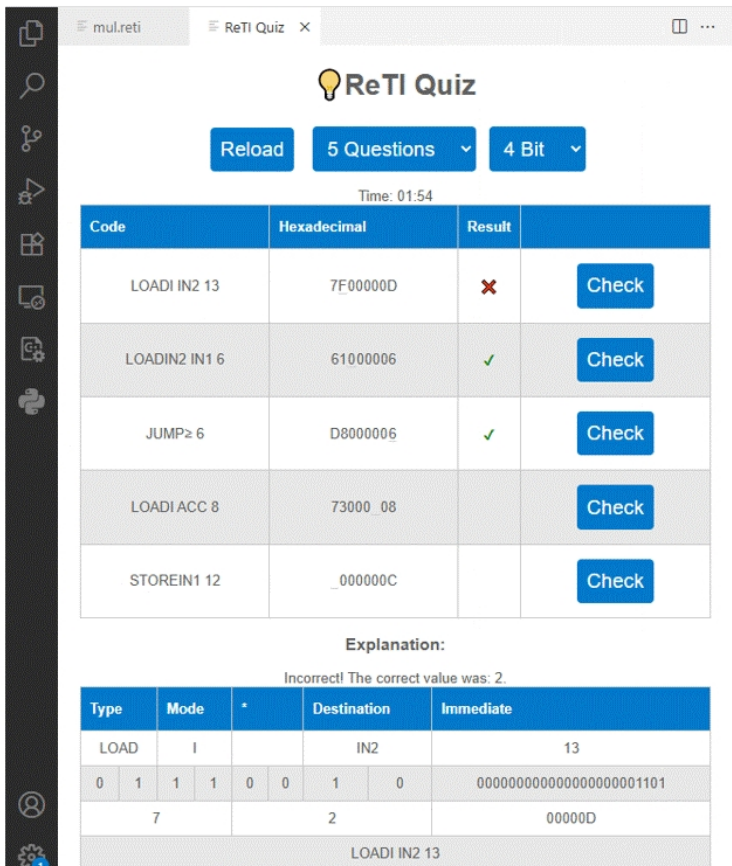


Fig 7: Screenshot of the new ReTI Quiz opened in VS Code

Results

New Features – Language Server

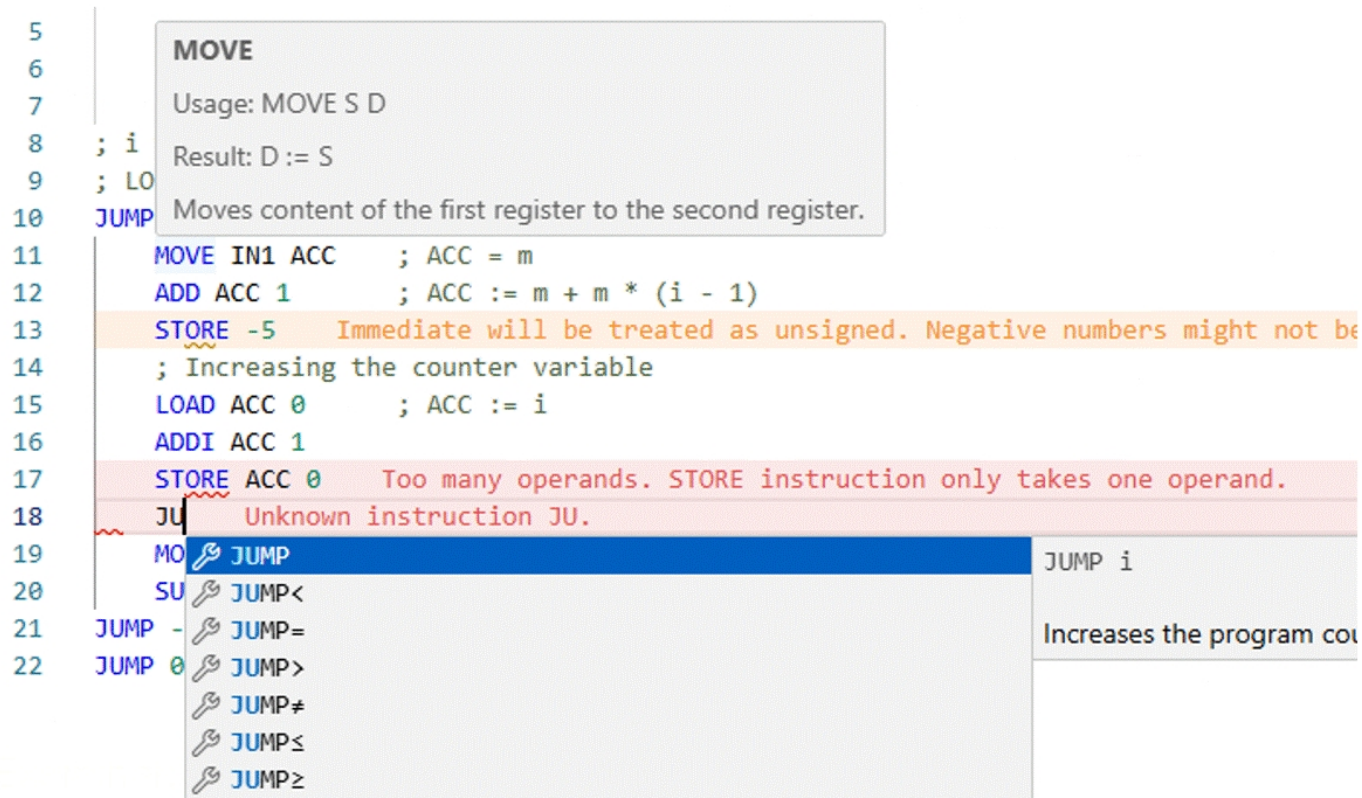


Fig 8: Screenshot from VS Code highlighting LSP Features

Features:

- Syntax Highlighting
- Tooltips
- Realtime compilation and checking
- Autocomplete suggestions

Results

New Features – Debugger

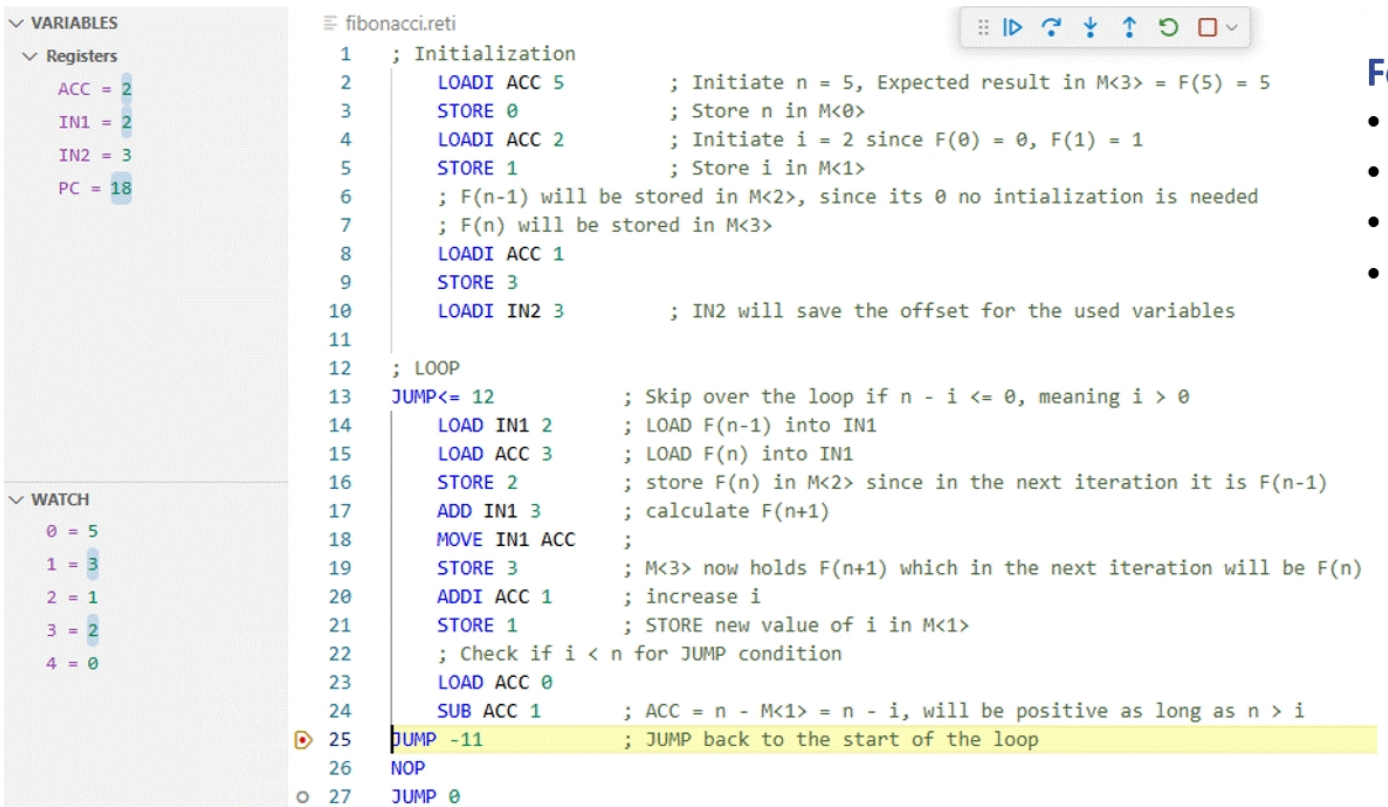


Fig 9: Screenshot of running ReTI-Debug session

Features:

- Reading and writing register values
- Reading memory
- Breakpoints
- Stepping

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

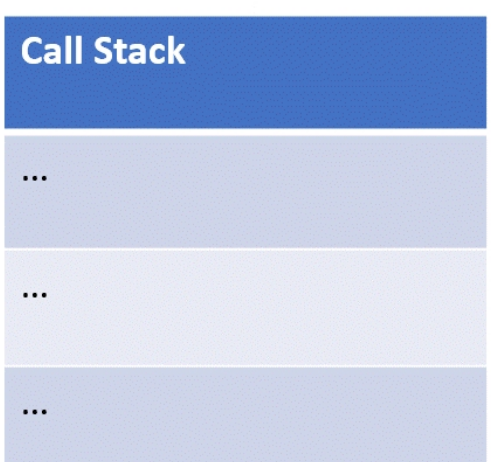
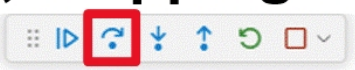


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti



```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

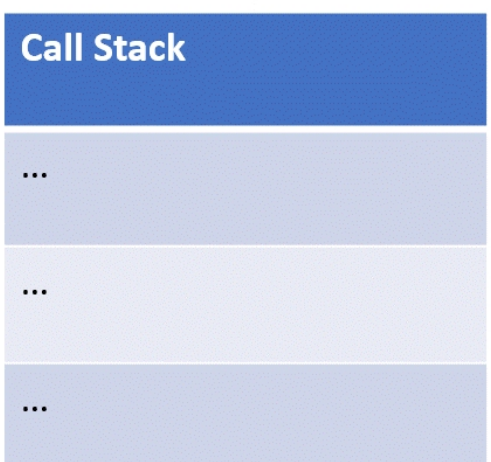


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

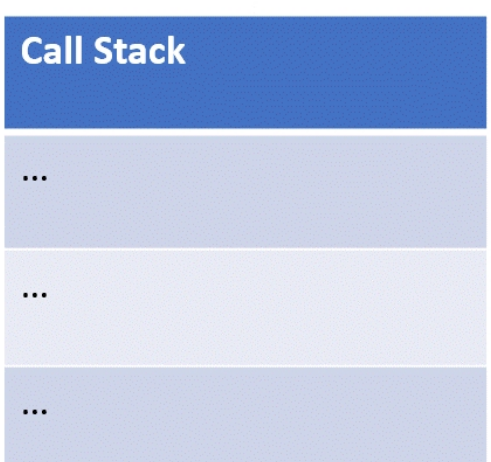
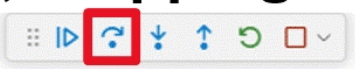


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti



```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

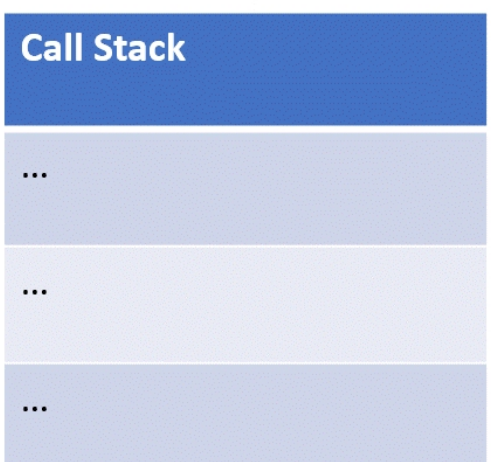


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

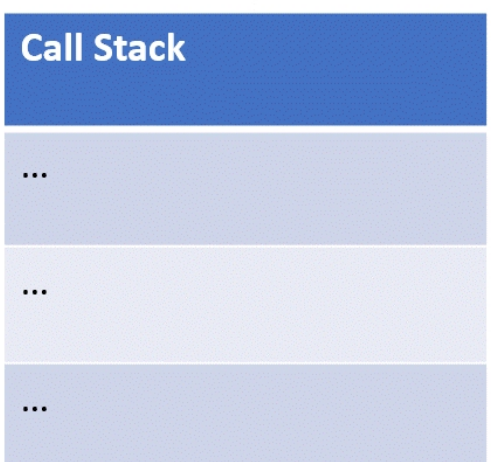


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

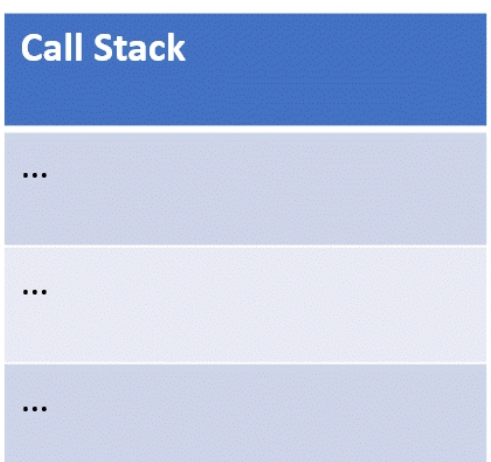


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

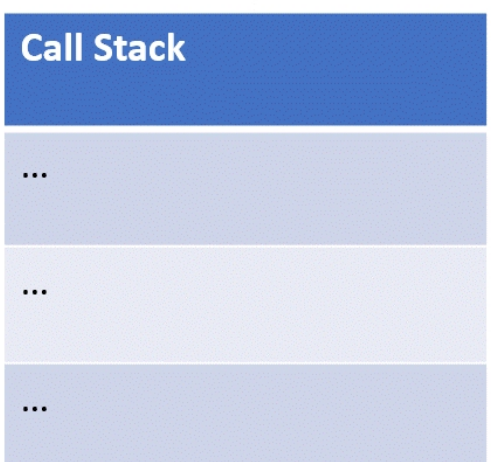


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

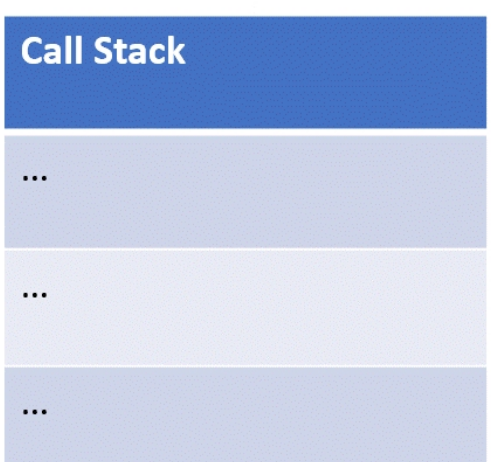


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

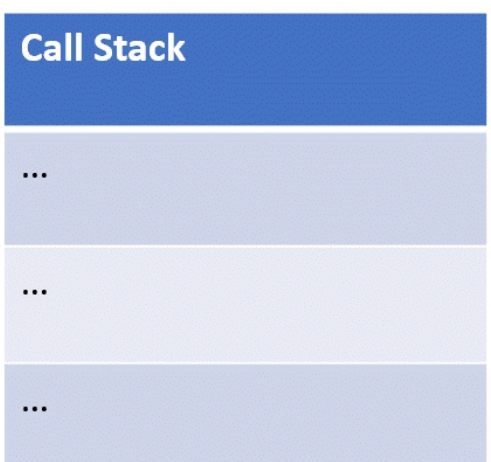


Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

Call Stack
19
...
...

Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

Call Stack
19
...
...

Fig. 3: Visual representation of the call stack

Results

New Features – Debugger, Stepping

fibonacci.reti

```
1   ; Initialization
2       LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3       STORE 0             ; Store n in M<0>
4       LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
5       STORE 1             ; Store i in M<1>
6       ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
7       ; F(n) will be stored in M<3>
8       LOADI ACC 1
9       STORE 3
10      LOADI IN2 3         ; IN2 will save the offset for the used variables
11
12  ; LOOP
13  JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
14      LOAD IN1 2      ; LOAD F(n-1) into IN1
15      LOAD ACC 3      ; LOAD F(n) into IN1
16      STORE 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
17      ADD IN1 3       ; calculate F(n+1)
18      MOVE IN1 ACC    ;
19      STORE 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
20      ADDI ACC 1      ; increase i
21      STORE 1         ; STORE new value of i in M<1>
22      ; Check if i < n for JUMP condition
23      LOAD ACC 0
24      SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
25  JUMP -11            ; JUMP back to the start of the loop
26  NOP
```

Fig. 10: Screenshot of running ReTI-Debug session

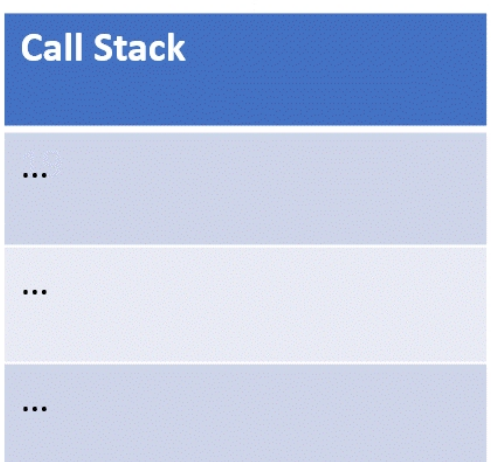


Fig. 3: Visual representation of the call stack

Conclusion
Goals Reached

Conclusion
Goals Reached

- Quiz more accessible both as website and in the extension

Conclusion
Goals Reached

- Quiz more accessible both as website and in the extension
- More Feedback in Quiz

Conclusion
Goals Reached

- Quiz more accessible both as website and in the extension
- More Feedback in Quiz
- All features (Assembler, Quiz, Emulator, RanReTi, Disassembler) accessible through extension

Conclusion
Goals Reached

- Quiz more accessible both as website and in the extension
- More Feedback in Quiz
- All features (Assembler, Quiz, Emulator, RanReTi, Disassembler) accessible through extension
- Debugger and Language Server offer
  - Syntax Highlighting
  - Realtime Compilation
  - Code Completion Suggestions
  - Executing Code and Inspecting/Manipulating ReTI State

Conclusion
Goals Reached

- Quiz more accessible both as website and in the extension
- More Feedback in Quiz
- All features (Assembler, Quiz, Emulator, RanReTi, Disassembler) accessible through extension
- Debugger and Language Server offer
  - Syntax Highlighting
  - Realtime Compilation
  - Code Completion Suggestions
  - Executing Code and Inspecting/Manipulating ReTI State
- Extension provides familiar programming workflow

Conclusion

Limitations & Future Work

- **Improve Memory View:**
  - Using “watch expression” is cumbersome
- **Expand to other versions of ReTI**
- **Add datapaths visualization:**

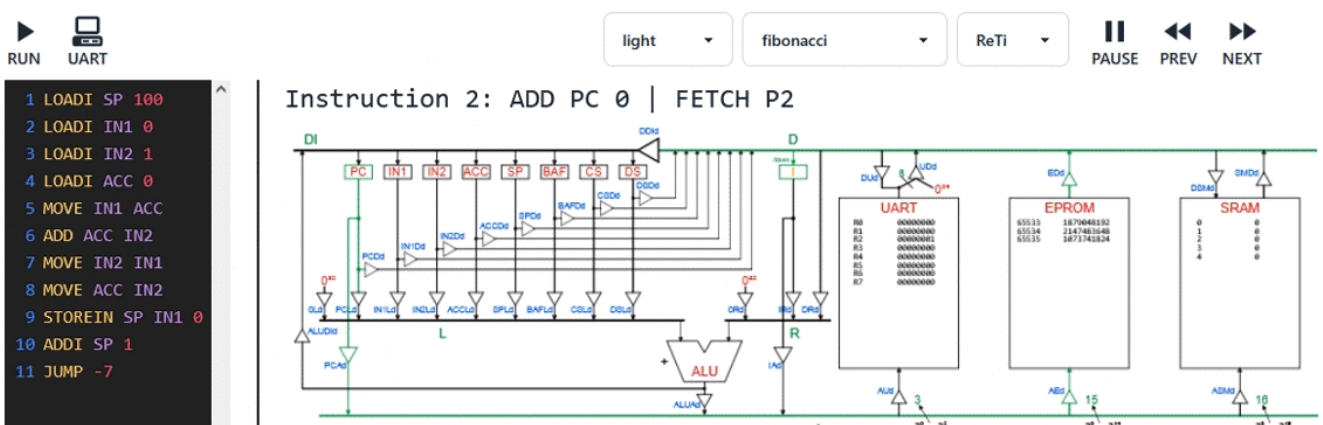


Fig. 11: Screenshot of ReTI-Emulator with Datapath-Visualization [3]

- Same project also provides Pico-C to ReTI compiler

Extra:
Language-Server-Protocol

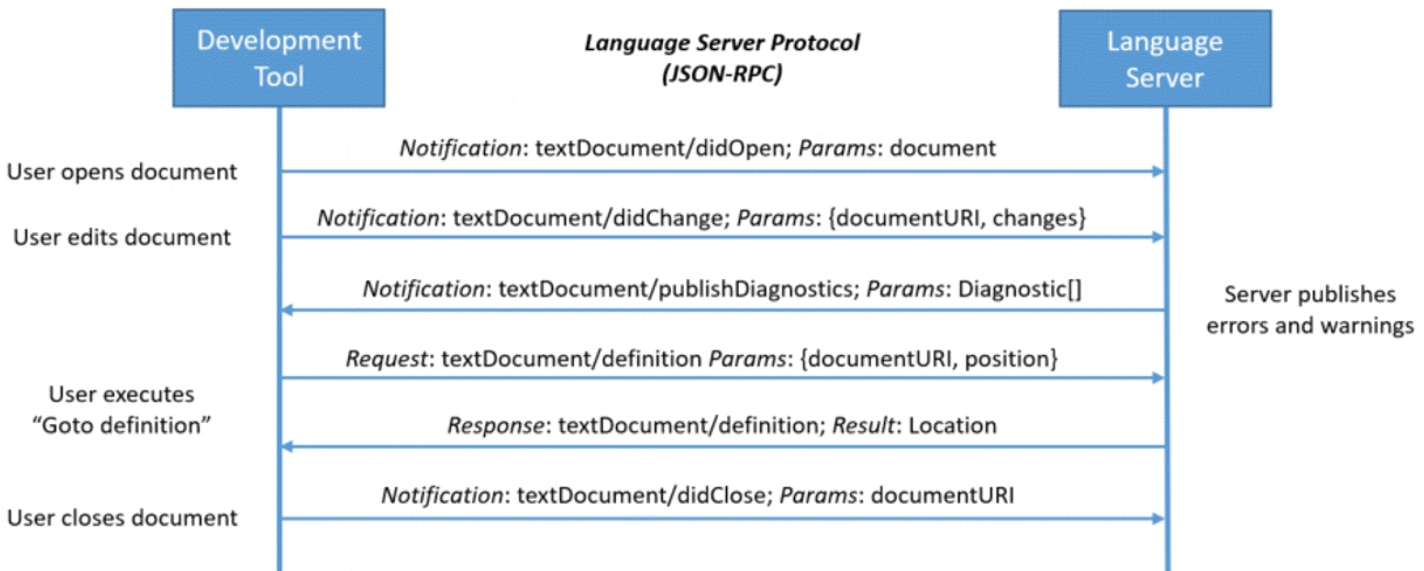


Fig. 12: Example for Communication between Tool and Server in the Language-Server-Protocol [4]

Extra:
Debug-Adapter-Protocol

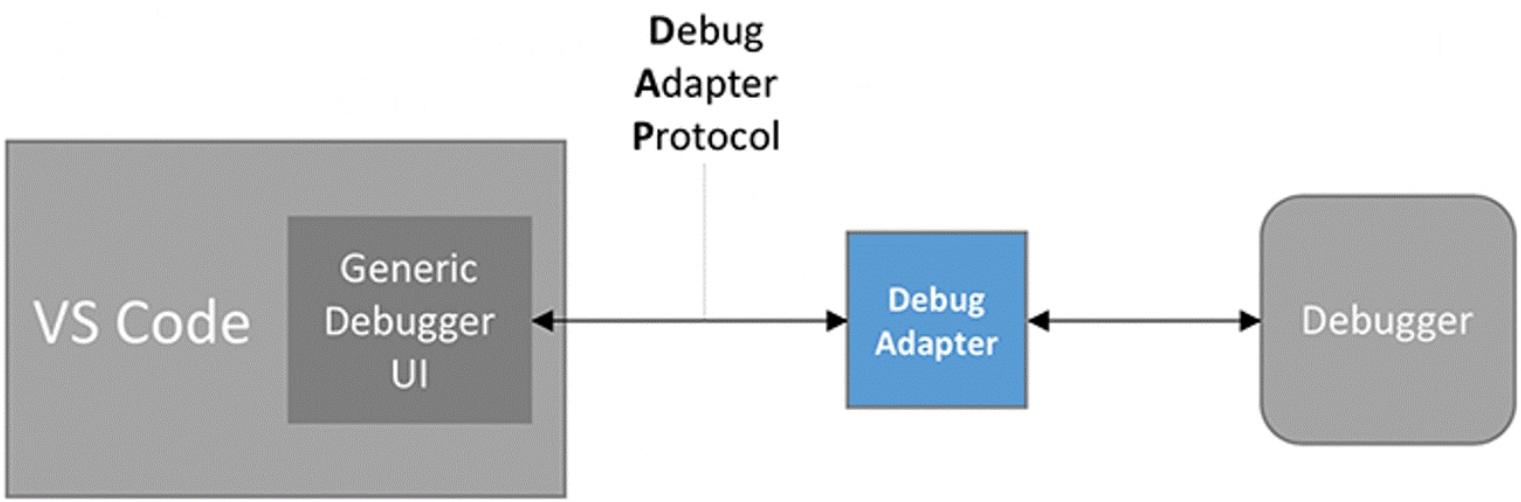


Fig. 13: VS Code Debug Architecture [5]

References

[1]: Screenshots taken of Reticode by Prof. Dr. Armin Biere, https://github.com/arminbiere/reticode
[2]: Technische Informatik – Kapitel 2 – Kodierung, Prof. Dr. Armin Biere, University of Freiburg, SS 2024
[3]: Screenshots taken of ReTI-Emulator, github.com/michel-giehl/Reti-Emulator
[4]: Microsoft *Language Server Protocol - Sequence Diagram*. Retrieved 13. July 2025, from https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/
[5]:Microsoft. *VS Code Debug Architecture*. Retrieved 13. July 2025, from https://code.visualstudio.com/api/extension-guides/debugger-extension