

ReTI-Tools: A VSCode Extension for ReTI Assembly Programming

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

Presenter: Malte Pullich

Chair of Computer Architecture

Examiner: Prof. Dr. Armin Biere

Adviser: Dr. Mathias Fleury, Tobias Faller

Freiburg, 12.11.2025

Agenda

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

Motivation

Registers	SRAM Codesegment: PC (2147483652)	SRAM Datasegment: DS (2147483670)	SRAM Stack: SP (2147549183)
PC: 2147483652 (-2147483644)	000000: ADDI PC 5<- CS	000005: STORE ACC 3	655001: -631242752
IN1: 0 (0)	000001: ADDI PC 2	000006: LOADI IN2 3	655002: -631242752
IN2: 0 (0)	000002: LOADI ACC 2	000007: JUMP<= 12	655003: -631242752
ACC: 2 (2)	000003: STORE ACC 1	000008: LOAD IN1 2	655004: -631242752
SP: 2147549183 (-2147418113)	000004: LOADI ACC 1<- PC	000009: LOAD ACC 3	655005: -631242752
BAF: 2147549183 (-2147418113)	000005: STORE ACC 3	000010: STORE ACC 2	655006: -631242752
CS: 2147483648 (-2147483648)	000006: LOADI IN2 3	000011: ADD IN1 3	655007: -631242752
DS: 2147483670 (-2147483626)	000007: JUMP<= 12	000012: MOVE IN1 ACC	655008: -631242752
	000008: LOAD IN1 2	000013: STORE ACC 3	655009: -631242752
	000009: LOAD ACC 3	000014: ADDI ACC 1	655010: -631242752
	000010: STORE ACC 2	000015: STORE ACC 1	655011: -631242752
	000011: ADD IN1 3	000016: LOAD ACC 0	655012: -631242752
	000012: MOVE IN1 ACC	000017: SUB ACC 1	655013: -631242752
	000013: STORE ACC 3	000018: JUMP -11	655014: -631242752
	000014: ADDI ACC 1	000019: NOP	655015: -631242752
	000015: STORE ACC 1	000020: JUMP 0	655016: -631242752
	000016: LOAD ACC 0	000021: ADD ACC IN1	655017: -631242752
	000017: SUB ACC 1	000022: 281542656<- DS	655018: -631242752
	000018: JUMP -11	000023: 281542656	655019: -631242752
	000019: NOP	000024: 281542656	655020: -631242752
	000020: JUMP 0	000025: 281542656	655021: -631242752
	000021: ADD ACC IN1	000026: 281542656	655022: -631242752
	000022: 281542656<- DS	000027: 281542656	655023: -631242752
	000023: 281542656	000028: 281542656	655024: -631242752
	000024: 281542656	000029: 281542656	655025: -631242752
	000025: 281542656	000030: 281542656	655026: -631242752
	000026: 281542656	000031: 281542656	655027: -631242752
	000027: 281542656	000032: 281542656	655028: -631242752
	000028: 281542656	000033: 281542656	655029: -631242752
	000029: 281542656	000034: 281542656	655030: -631242752
	000030: 281542656	000035: 281542656	655031: -631242752
	000031: 281542656	000036: 281542656	655032: -631242752
	000032: 281542656	000037: 281542656	655033: -631242752
	000033: 281542656	000038: 281542656	655034: -631242752
	000034: 281542656	000039: 281542656	655035: -631242752<- SP BAF

(n)ext instruction, (c)ontinue to breakpoint, (r)estart, (s)tep into isr, (f)inalize isr, (t)rigger isr, (a)ssign watchobject reg or addr,

Fig.1 : Screenshot of the Emulator by Jürgen Mattheis [1]

Emulator by Jürgen Mattheis:

- OS-dependent (only Linux support)
- Workflow requires switching between editor and debugger

Motivation

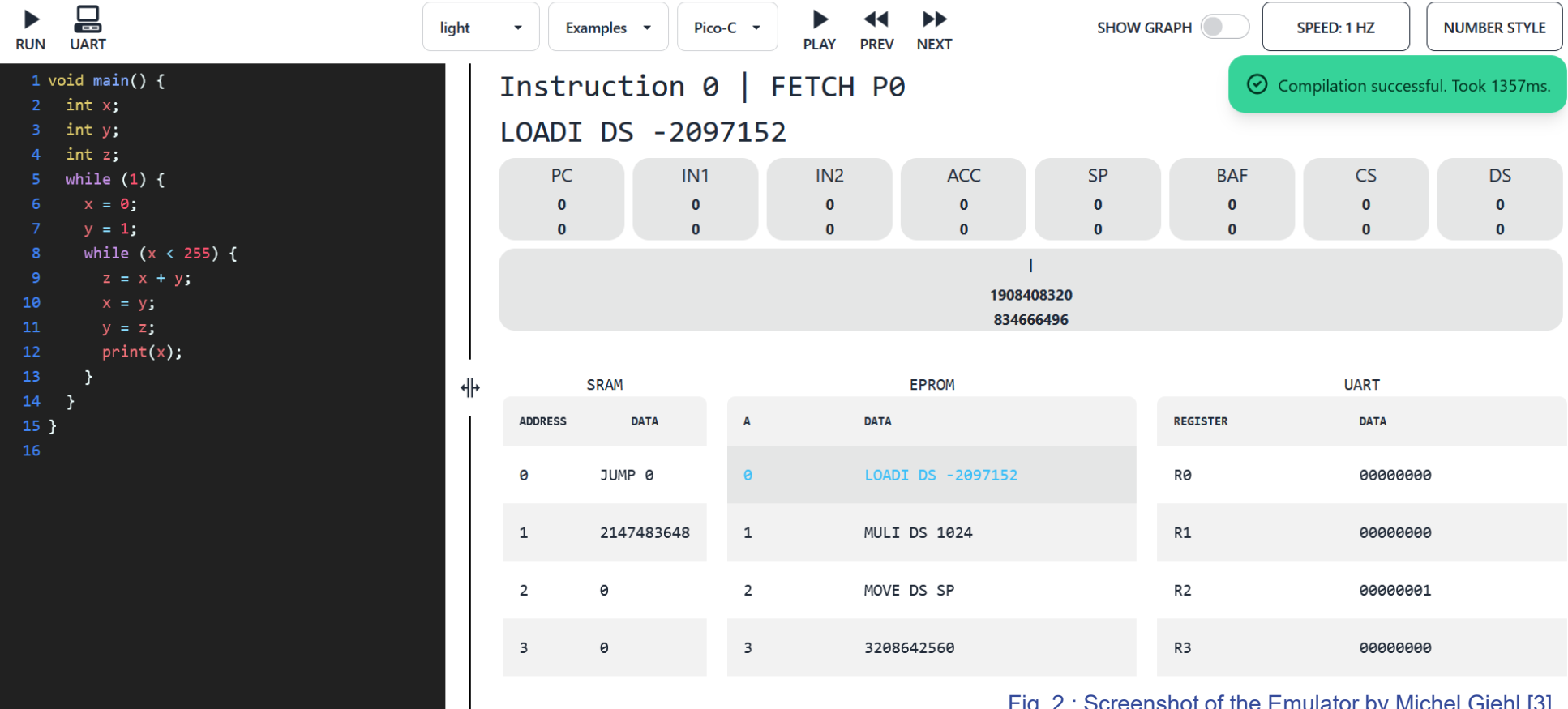


Fig. 2 : Screenshot of the Emulator by Michel Giehl [3]

Emulator by Michel Giehl:

- Web app, as of Monday 10th November, only accessible through Uni network
- Doesn't support interrupts

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: 3 Excerpt of a ReTI program that calculates Fibonacci numbers

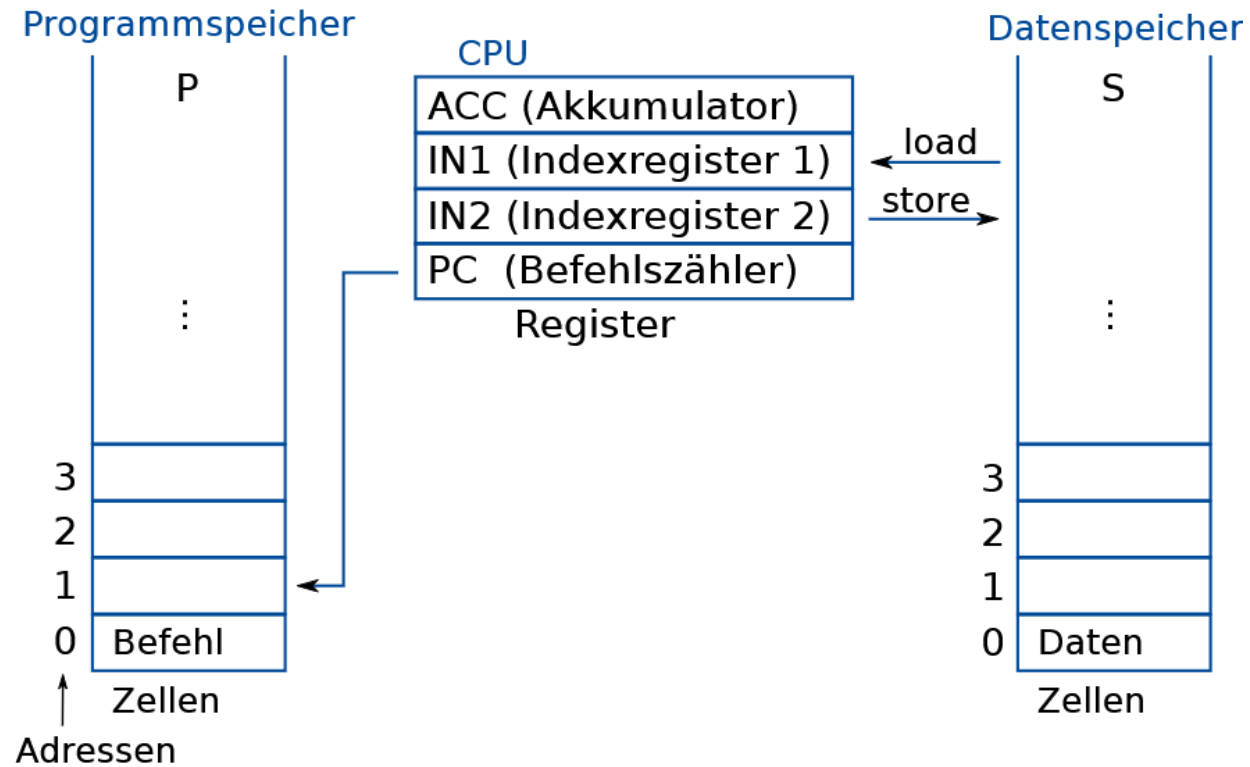


Fig 4: Abstract ReTI architecture [2]

Differences Between the ReTI Variants

ReTI-I (Technical Informatics):

- **Memory is single SRAM**
- **1 new internal registers:**
 - **I**, Instruction Register
- **No Interrupts**

ReTI-II (Operating Systems):

- **Memory split into SRAM, EPROM, UART**
- **4 new user-visible registers:**
 - **SP**, Stack Pointer
 - **BAF**, Begin Active Frame
 - **CS**, (Begin of) Code Segment
 - **DS**, (Begin of) Data Segment
- **Interrupts:**
 - Interrupt controller
 - New register **IVN**
- **New instruction encoding (3 bits for registers)**
- **New instructions (MUL, DIV, MOD)**

Approach

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

Extend existing VS Code extension:

- Most used IDE
- OS-independent
- Same tool for both lectures

Identified key features to extend:

- Emulator
- Debugger
- Language Server

Additionally:

- Provide a memory view

Approach

Extension's Architecture

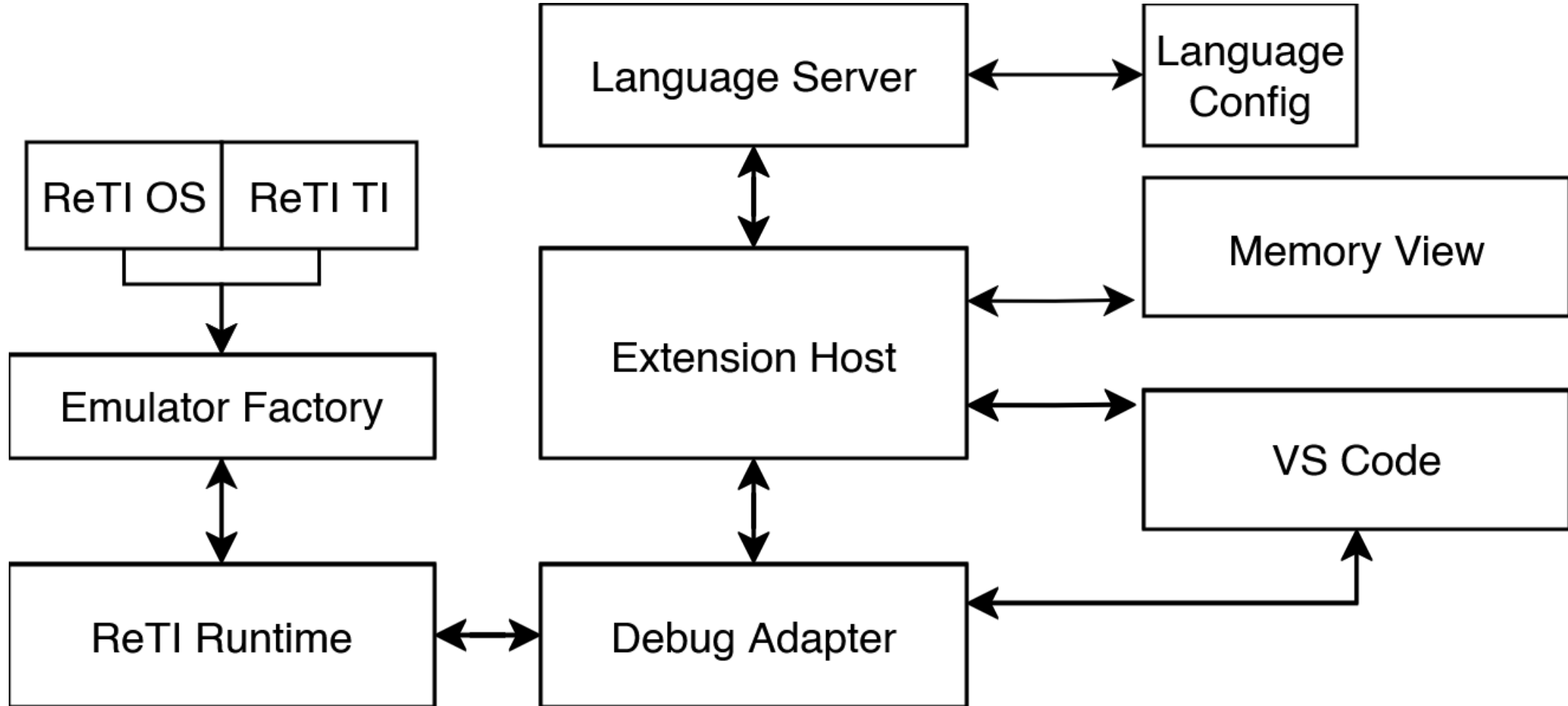


Fig. 5: Diagram illustrating the interaction between the different components of the extension.

Results

Support for Both Architectures

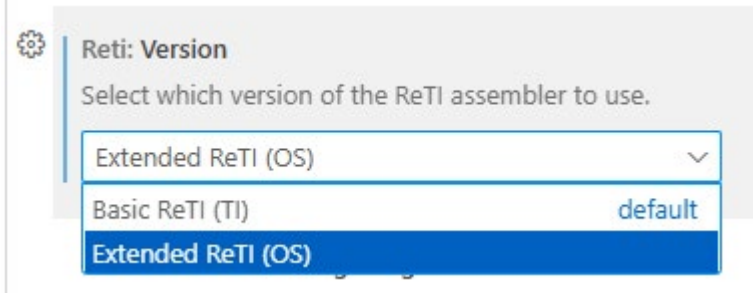


Fig 6: Screenshot of the new setting in VS Code

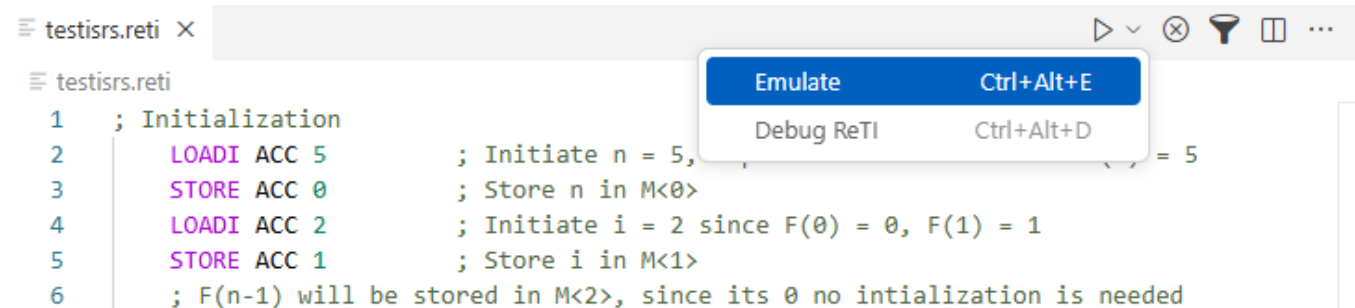
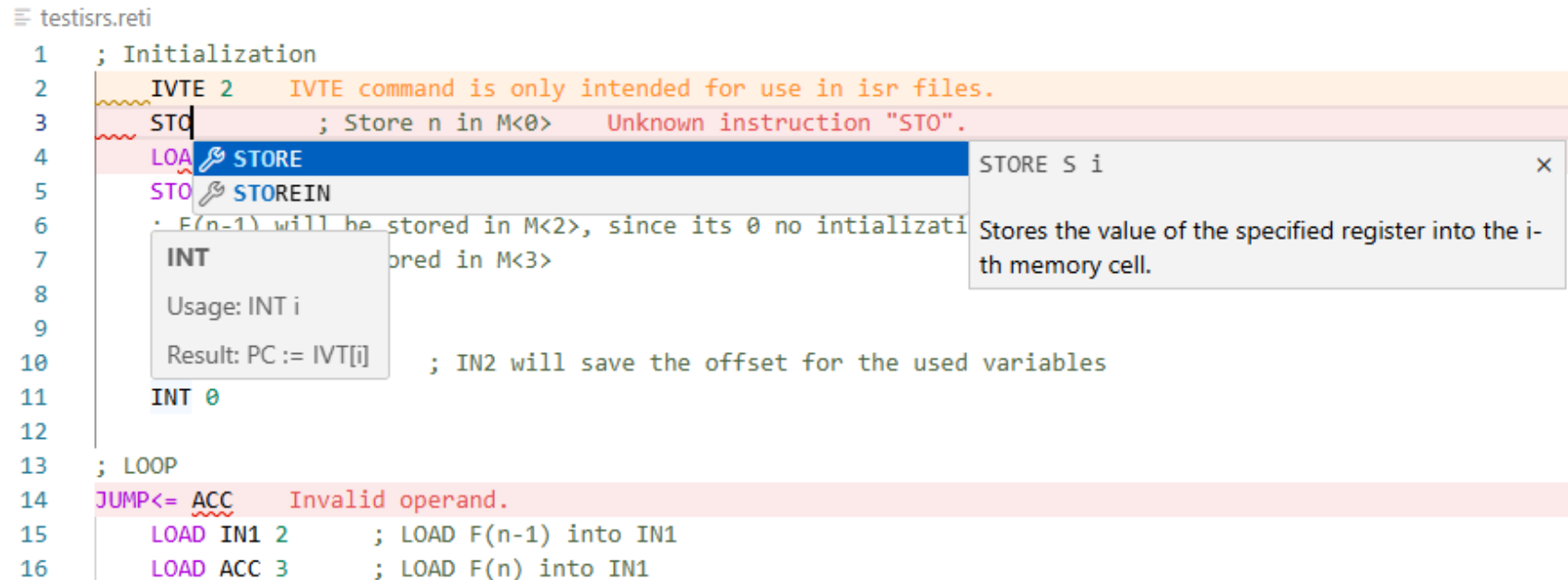


Fig 7: Example Usage of the ReTI Emulation in VS Code

- Added setting to specify desired version
- Affects all features (emulator, debugger, language server) except quiz
- Emulator now callable in .reti files for ReTI-II (OS)

Results

Language Server



```
testisrs.reti
1 ; Initialization
2 IVTE 2      IVTE command is only intended for use in isr files.
3 STQ        ; Store n in M<0>      Unknown instruction "STQ".
4 LOA STORE
5 STO STOREIN
6 ; F(n-1) will be stored in M<2>, since its 0 no initializati
7 ; F(n-1) will be stored in M<3>
8 INT
9 Usage: INT i
10 Result: PC := IVT[i] ; IN2 will save the offset for the used variables
11 INT 0
12
13 ; LOOP
14 JUMP<= ACC Invalid operand.
15 LOAD IN1 2 ; LOAD F(n-1) into IN1
16 LOAD ACC 3 ; LOAD F(n) into IN1
```

Fig 8: Screenshot from VS Code highlighting LSP Features

Features:

- Syntax Highlighting
- Tooltips (syntax and documentation)
- Realtime compilation and checking
- Autocomplete suggestions

Results

Debugger and Memory View

The screenshot displays the ReTI-Debug interface. On the left, the 'MEMORY VIEW' panel shows a table with 'Address' and 'Value' columns. The 'VARIABLES' panel below it lists registers: PC = 2, IN1 = 0, IN2 = 0, ACC = 5, SP = 2147483703, BAF = 2147483650, CS = 2147483651, DS = 2147483672, and I = 2248146944. The main assembly window shows the 'fibonacci_os.reti' file with the following code:

```
1  ; Initialization
2  LOADI ACC 5      ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3  STORE ACC 0      ; Store n in M<0>
4  LOADI ACC 2      ; Initiate i = 2 since F(0) = 0, F(1) = 1
5  STORE ACC 1      ; Store i in M<1>
6  ; F(n-1) will be stored in M<2>, since its 0 no initialization is needed
7  ; F(n) will be stored in M<3>
8  LOADI ACC 1
9  STORE ACC 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 ACC 2      ; store F(n) in M<2> since in the next iteration it is F(n-
17 ADD IN1 3        ; calculate F(n+1)
18 MOVE IN1 ACC     ;
19 STORE ACC 3      ; M<3> now holds F(n+1) which in the next iteration will be
20 ADDI ACC 1       ; increase i
21 STORE ACC 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
27 JUMP 0
```

Fig 9: Screenshot of running ReTI-Debug session

Features both architectures:

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

Features for the ReTI-II (OS):

- Switching between main program and interrupt service routine file
- Updated Stepping Logic to support interrupts

Results

Debugger, Stepping

```
testisrs.reti
1  ∨ ; Initialization
2      LOADI ACC 5      ; Initiate n = 5, Expected result in M<3> = F(5) = 5
3      STORE ACC 0      ; Store n in M<0>
4      LOADI ACC 2      ; Initiate i = 2 since F(0) = 0, F(1) = 1
5      STORE ACC 1      ; Store i in M<1>
6      ; F(n-1) will be stored in M<2>, since its 0 no initialization is needed
7      ; F(n) will be stored in M<3>
8      LOADI ACC 1
9      STORE ACC 3
10     LOADI IN2 3      ; IN2 will save the offset for the used variables
11     INT 0
12
13     ; LOOP
14     ∨ JUMP<= 12      ; Skip over the loop if n - i <= 0, meaning i > 0
15         LOAD IN1 2      ; LOAD F(n-1) into IN1
16         LOAD ACC 3      ; LOAD F(n) into IN1
17         STORE ACC 2      ; store F(n) in M<2> since in the next iteration it is F(n-1)
18         ADD IN1 3      ; calculate F(n+1)
19         MOVE IN1 ACC      ;
20         NOP
21         STORE ACC 3      ; M<3> now holds F(n+1) which in the next iteration will be F(n)
22         ADDI ACC 1      ; increase i
23         STORE ACC 1      ; STORE new value of i in M<1>
24         ; Check if i < n for JUMP condition
25         LOAD ACC 0
26         INT 1
27         SUB ACC 1      ; ACC = n - M<1> = n - i, will be positive as long as n > i
28     JUMP -11      ; JUMP back to the start of the loop
29     NOP
30     JUMP 0
31
```

```
isrs.reti
1      ; Example routine that restores registers
2      IVTE 0
3      SUBI SP 7
4      STOREIN SP IN1 7
5      STOREIN SP IN2 6
6      STOREIN SP ACC 5
7
8      STOREIN SP CS 2
9      STOREIN SP DS 1
10
11     LOADI IN1 2
12     LOADI IN2 5
13     SUB IN2 IN1
14     LOADI ACC 1
15
16     LOADIN SP DS 1
17     LOADIN SP CS 2
18     LOADIN SP ACC 5
19     LOADIN SP IN2 6
20     LOADIN SP IN1 7
21     ADDI SP 7
22     RTI
23     ; Example routine that keeps register values.
24     IVTE 1
25     LOADI ACC 2
26     MOVE ACC IN1
27     LOADI IN2 5
28     SUB IN2 IN1
29     LOADI ACC 1
30     RTI
```

Fig. 10: Screenshots of running ReTI-Debug session

Conclusion

Goals Reached

- All features (emulator, debugger, language server) extended to cover both versions ✓
- Memory view allows easier monitoring and manipulation of memory ✓
- Debugger and Language Server offer
 - Syntax Highlighting ✓
 - Realtime Compilation ✓
 - Code Completion Suggestions ✓
 - Executing Code and Inspecting/Manipulating ReTI State ✓
- Extension provides familiar programming workflow and OS-independent use ✓

Conclusion

Limitations & Future Work

- **Emulate UART interface**
- **Make debug adapter independent from VS Code (embedding in other editors/tools)**
- **Integrate Pico-C compiler**
- **Add datapaths visualization [3]:**

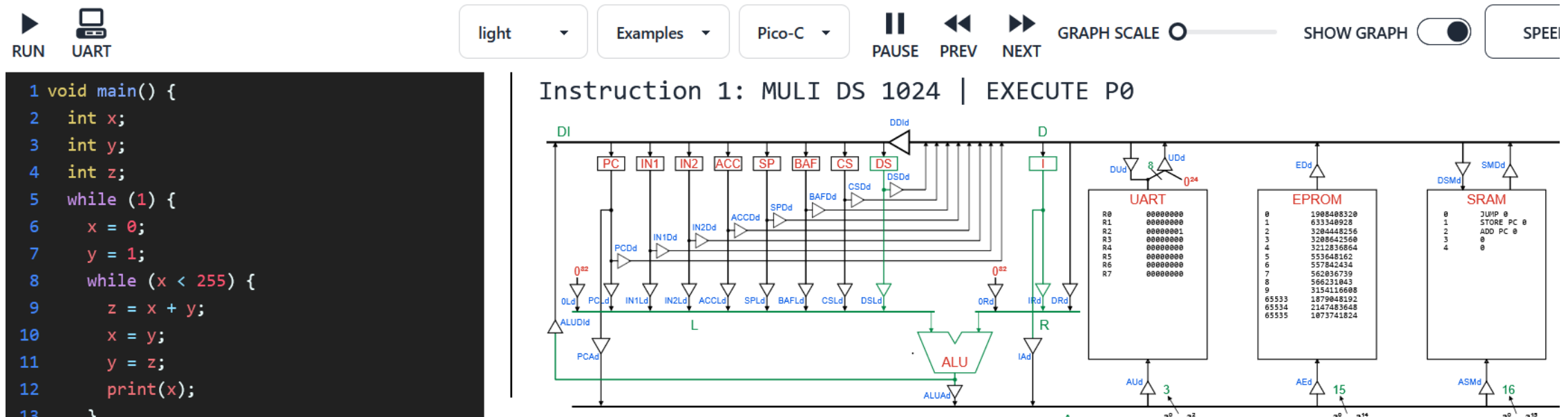


Fig. 11: Screenshot of ReTI-Emulator with Datapath-Visualization [3]

Extra:

Language-Server-Protocol

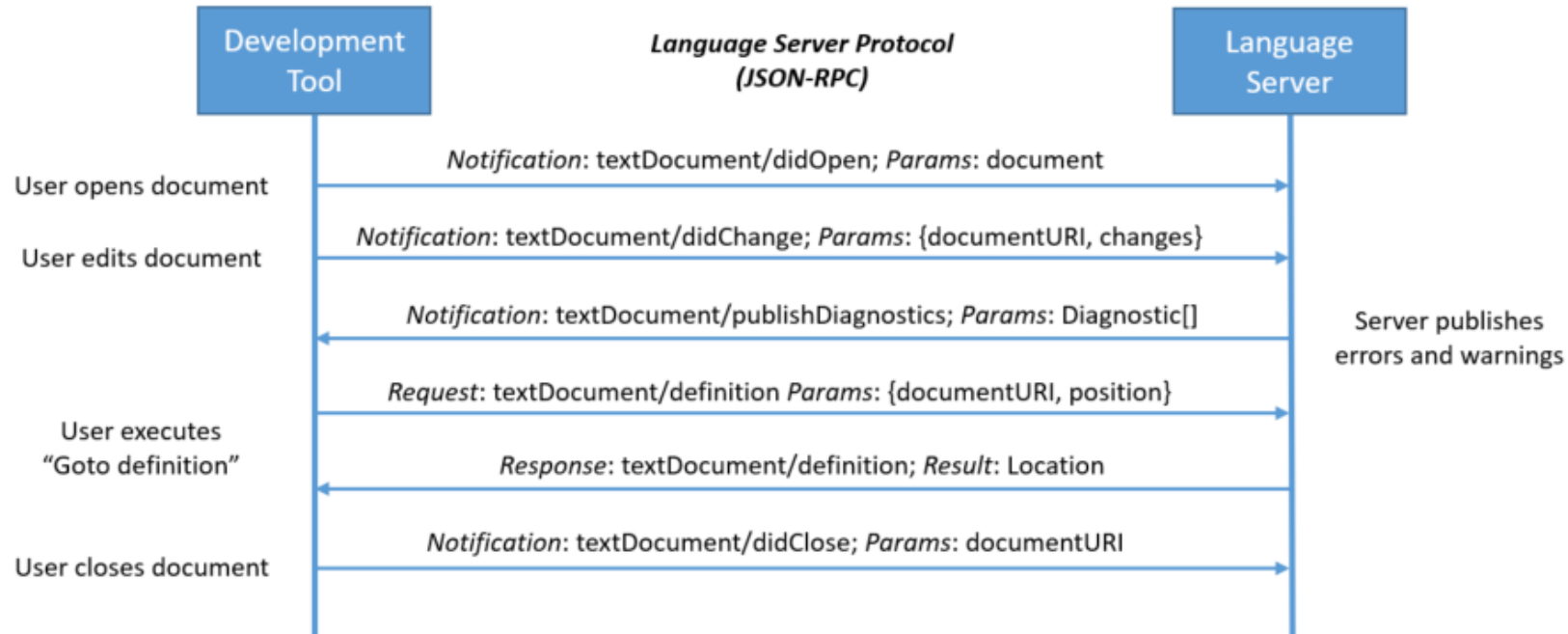


Fig. 12: Example for Communication between Tool and Server in the Language-Server-Protocol [4]

Extra: Debug-Adapter-Protocol

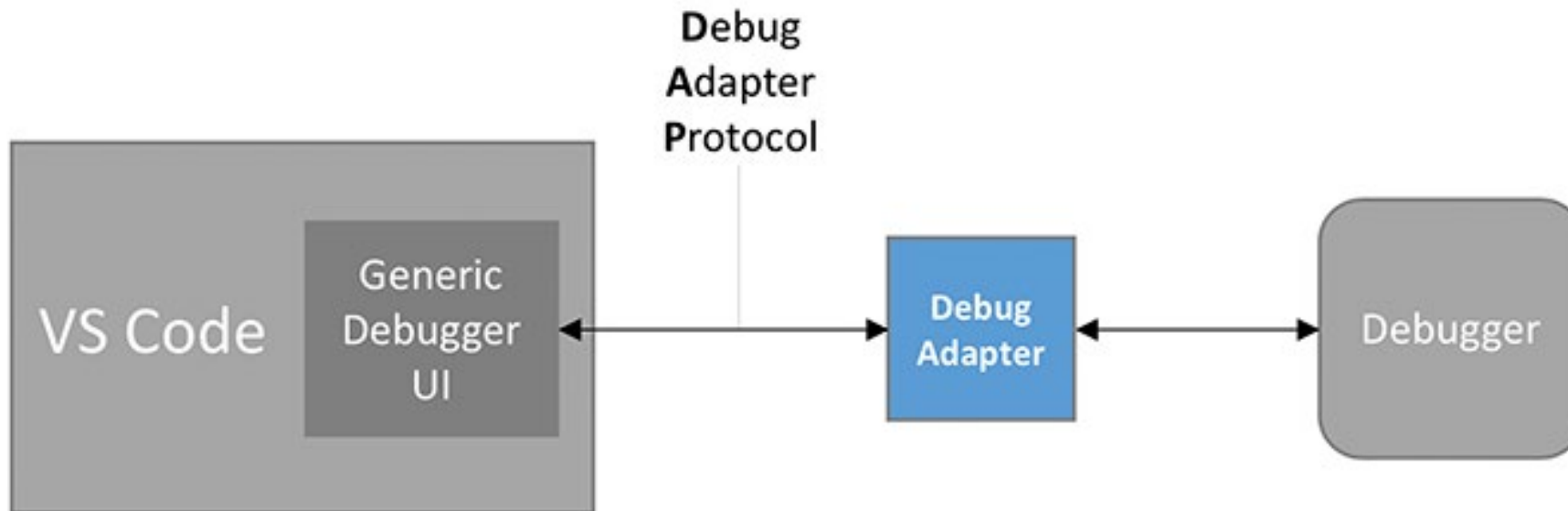


Fig. 13: VS Code Debug Architecture [5]

References

- [1]: RETI-Emulator by Jürgen Mattheis, <https://github.com/matthejue/RETI-Emulator>
- [2]: Technische Informatik – Kapitel 2 – Kodierung, Prof. Dr. Armin Biere, University of Freiburg, SS 2024
- [3]: RETI-Emulator by Michel Giehl, 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>