universität freiburg

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

ReTi-Architecture

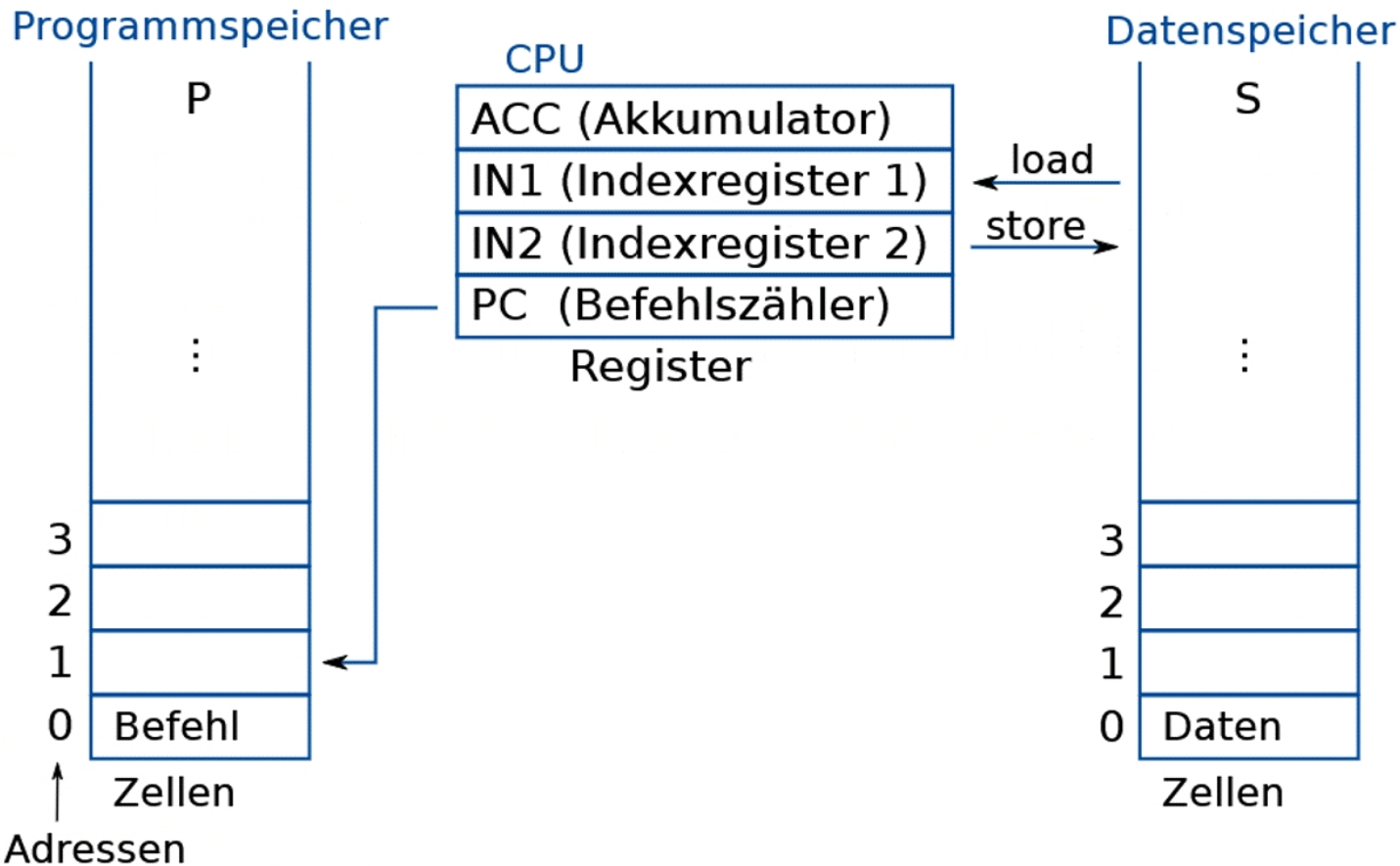

Fig 1: Abstract ReTI architecture [2]

Differences Between the ReTI Variants

Differences Between the ReTI Variants

ReTI-I (Technical Informatics):

ReTI-II (Operating Systems):

Differences Between the ReTI Variants

ReTI-I (Technical Informatics):

- **Memory is single SRAM**

ReTI-II (Operating Systems):

- **Memory split into SRAM, EPROM, UART**

Differences Between the ReTI Variants

ReTI-I (Technical Informatics):

- **Memory is single SRAM**
- **1 new internal registers:**
  - **I,** Instruction Register

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

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**

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)**

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)**

Motivation

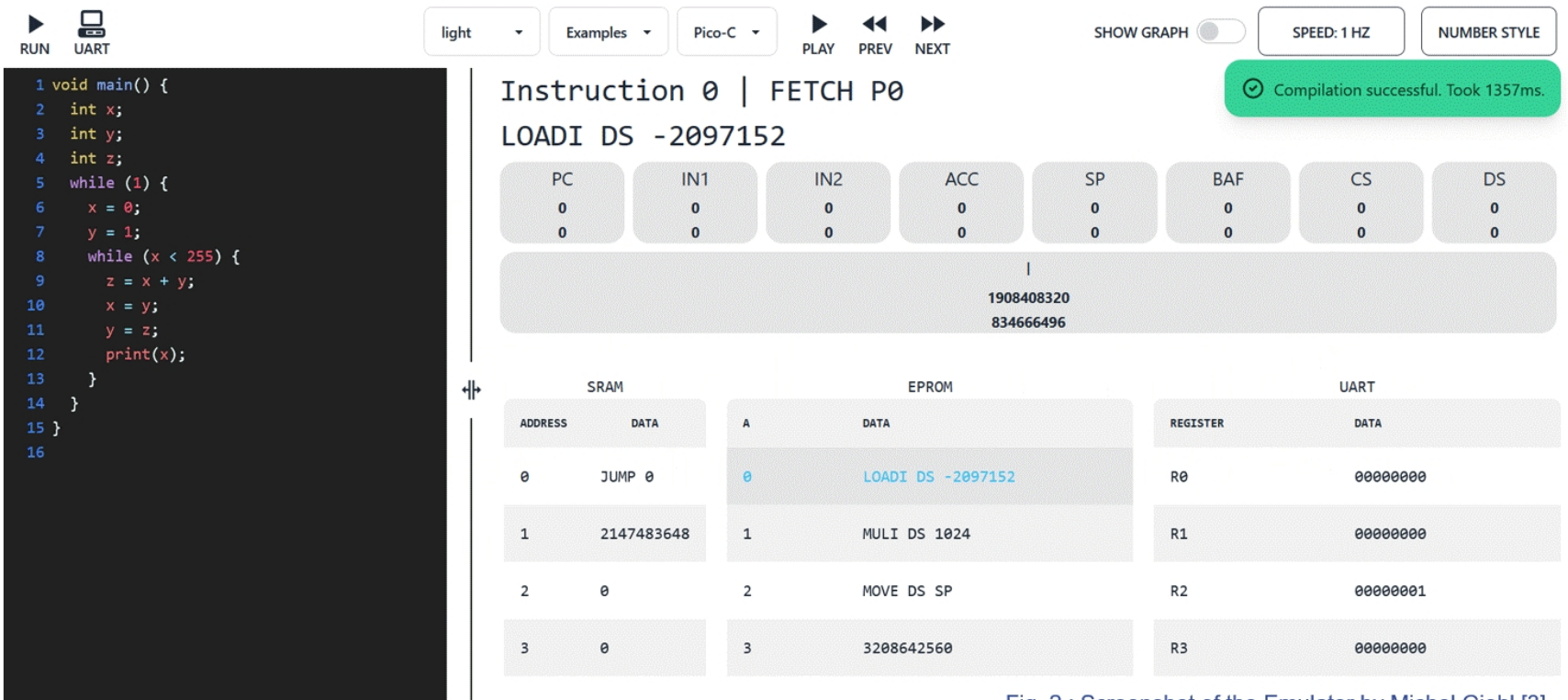

Fig. 2 : Screenshot of the Emulator by Michel Giehl [3]

Motivation

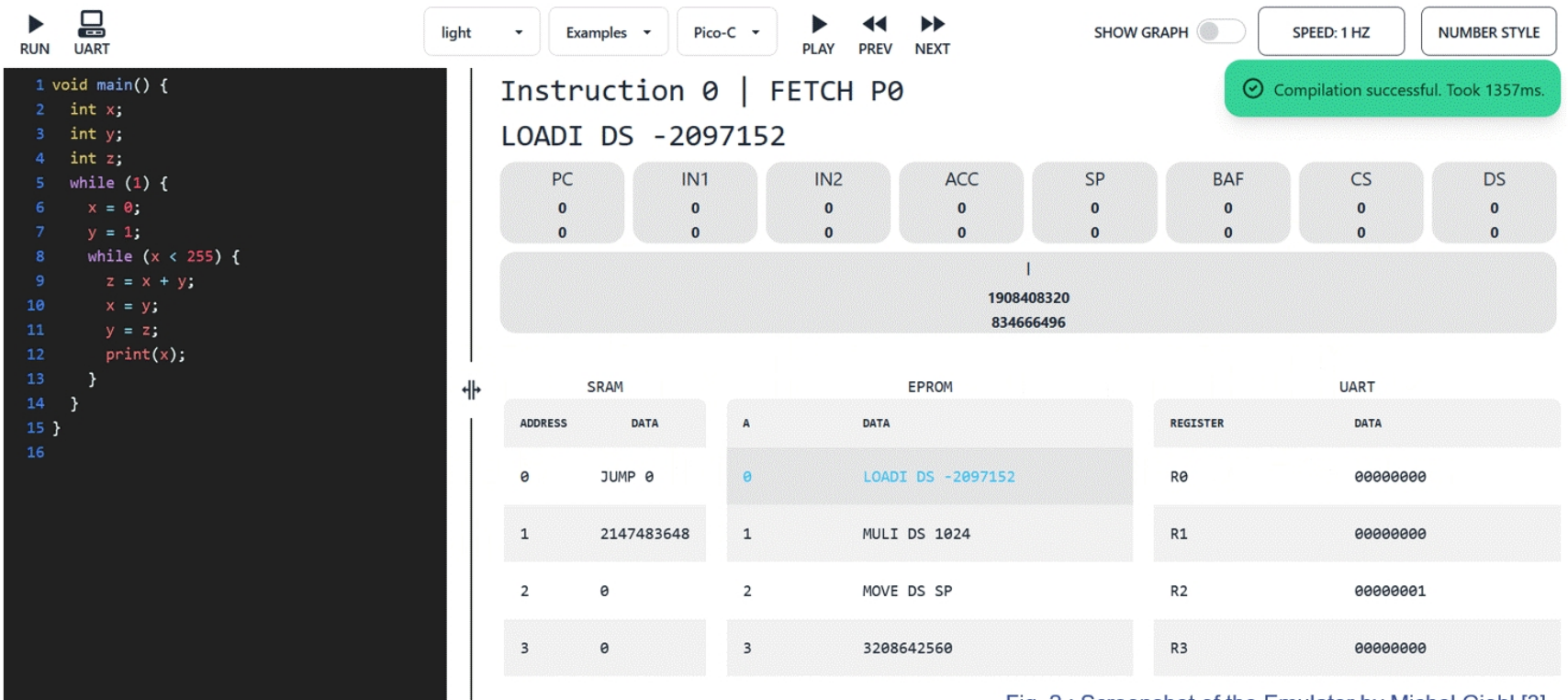

Fig. 2 : Screenshot of the Emulator by Michel Giehl [3]

Emulator by Michel Giehl:

Motivation

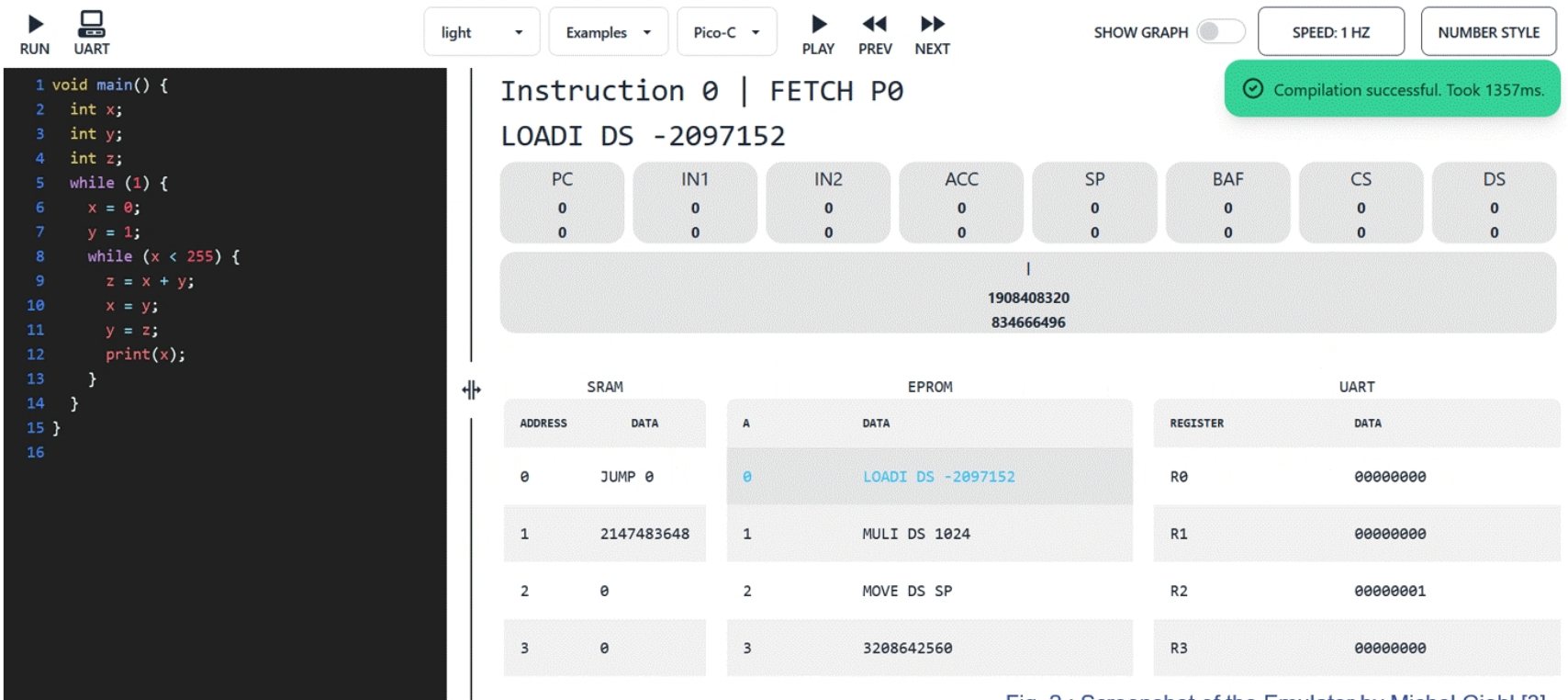

Fig. 2 : Screenshot of the Emulator by Michel Giehl [3]

Emulator by Michel Giehl:

- Web app, as of Monday 10th November, only accessible through Uni network

Motivation

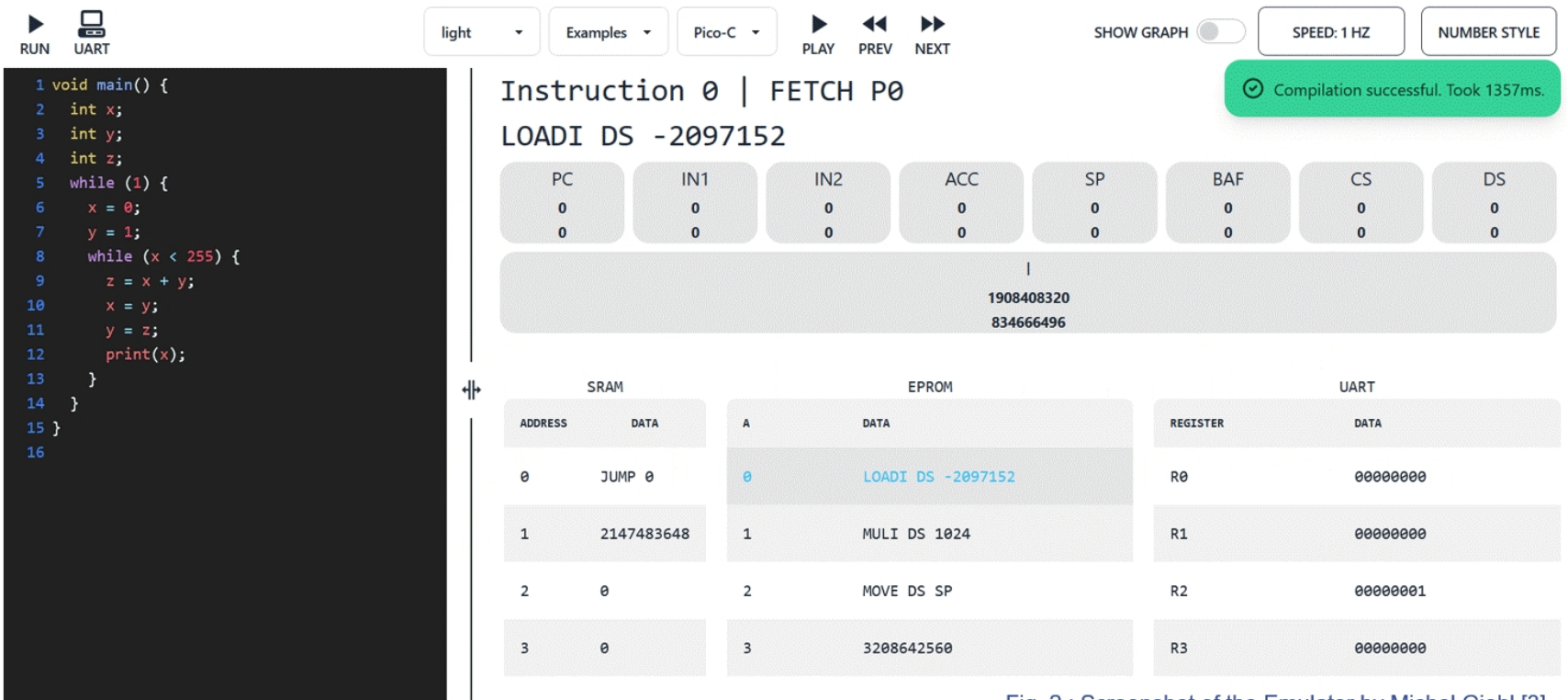

Fig. 2 : Screenshot of the Emulator by Michel Giehl [3]

Emulator by Michel Giehl:

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

Motivation

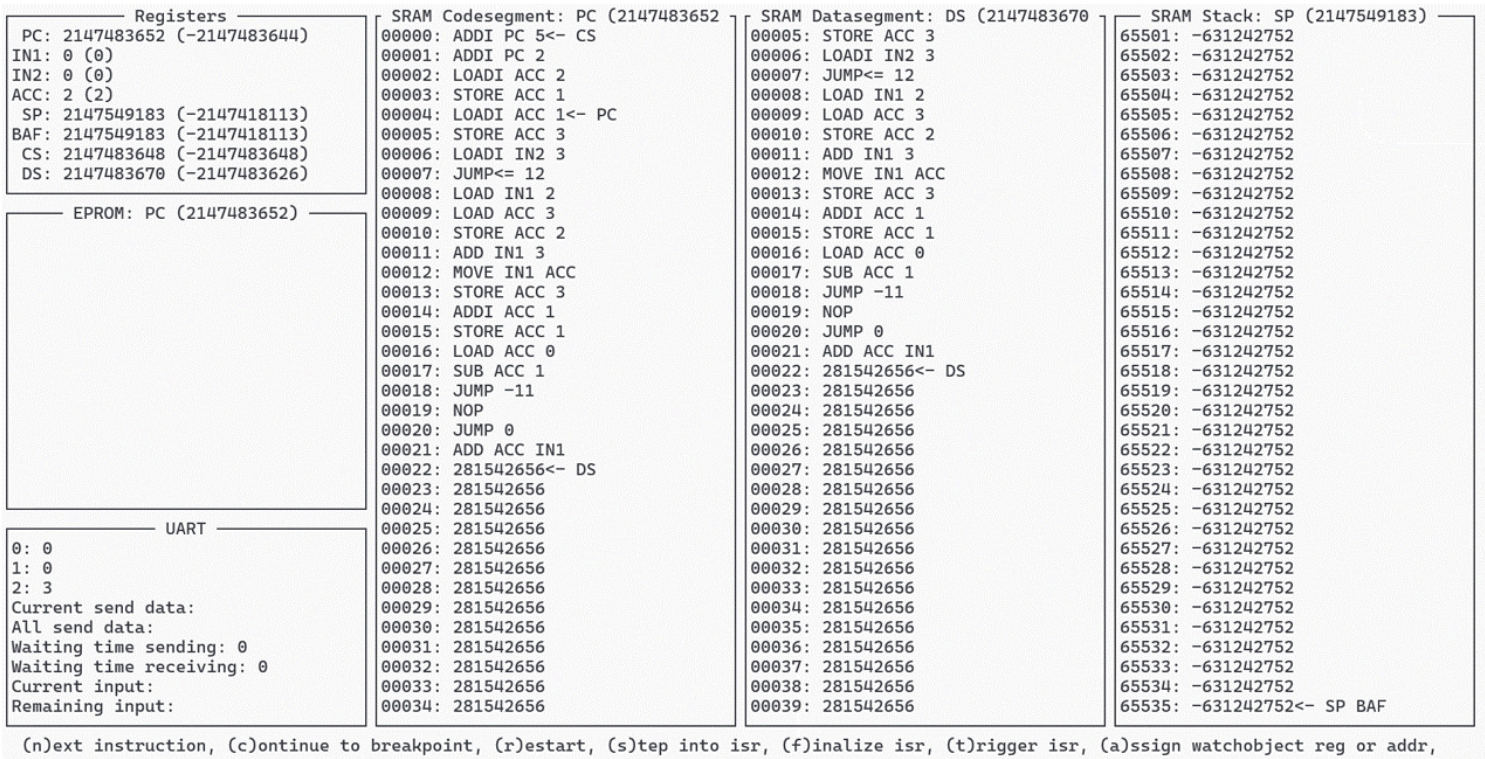

Fig. 3 : Screenshot of the Emulator by Jürgen Mattheis [1]

Motivation

```
Registers
 PC: 2147483652 (-2147483644)
IN1: 0 (0)
IN2: 0 (0)
ACC: 2 (2)
 SP: 2147549183 (-2147418113)
BAF: 2147549183 (-2147418113)
 CS: 2147483648 (-2147483648)
 DS: 2147483670 (-2147483626)

EPROM: PC (2147483652)

UART
0: 0
1: 0
2: 3
Current send data:
All send data:
Waiting time sending: 0
Waiting time receiving: 0
Current input:
Remaining input:

SRAM Codesegment: PC (2147483652
00000: ADDI PC 5<- CS
00001: ADDI PC 2
00002: LOADI ACC 2
00003: STORE ACC 1
00004: LOADI ACC 1<- PC
00005: STORE ACC 3
00006: LOADI IN2 3
00007: JUMP<= 12
00008: LOAD IN1 2
00009: LOAD ACC 3
00010: STORE ACC 2
00011: ADD IN1 3
00012: MOVE IN1 ACC
00013: STORE ACC 3
00014: ADDI ACC 1
00015: STORE ACC 1
00016: LOAD ACC 0
00017: SUB ACC 1
00018: JUMP -11
00019: NOP
00020: JUMP 0
00021: ADD ACC IN1
00022: 281542656<- DS
00023: 281542656
00024: 281542656
00025: 281542656
00026: 281542656
00027: 281542656
00028: 281542656
00029: 281542656
00030: 281542656
00031: 281542656
00032: 281542656
00033: 281542656
00034: 281542656

SRAM Datasegment: DS (2147483670
00005: STORE ACC 3
00006: LOADI IN2 3
00007: JUMP<= 12
00008: LOAD IN1 2
00009: LOAD ACC 3
00010: STORE ACC 2
00011: ADD IN1 3
00012: MOVE IN1 ACC
00013: STORE ACC 3
00014: ADDI ACC 1
00015: STORE ACC 1
00016: LOAD ACC 0
00017: SUB ACC 1
00018: JUMP -11
00019: NOP
00020: JUMP 0
00021: ADD ACC IN1
00022: 281542656<- DS
00023: 281542656
00024: 281542656
00025: 281542656
00026: 281542656
00027: 281542656
00028: 281542656
00029: 281542656
00030: 281542656
00031: 281542656
00032: 281542656
00033: 281542656
00034: 281542656
00035: 281542656
00036: 281542656
00037: 281542656
00038: 281542656
00039: 281542656

SRAM Stack: SP (2147549183)
65501: -631242752
65502: -631242752
65503: -631242752
65504: -631242752
65505: -631242752
65506: -631242752
65507: -631242752
65508: -631242752
65509: -631242752
65510: -631242752
65511: -631242752
65512: -631242752
65513: -631242752
65514: -631242752
65515: -631242752
65516: -631242752
65517: -631242752
65518: -631242752
65519: -631242752
65520: -631242752
65521: -631242752
65522: -631242752
65523: -631242752
65524: -631242752
65525: -631242752
65526: -631242752
65527: -631242752
65528: -631242752
65529: -631242752
65530: -631242752
65531: -631242752
65532: -631242752
65533: -631242752
65534: -631242752
65535: -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. 3 : Screenshot of the Emulator by Jürgen Mattheis [1]

Emulator by Jürgen Mattheis:

Motivation

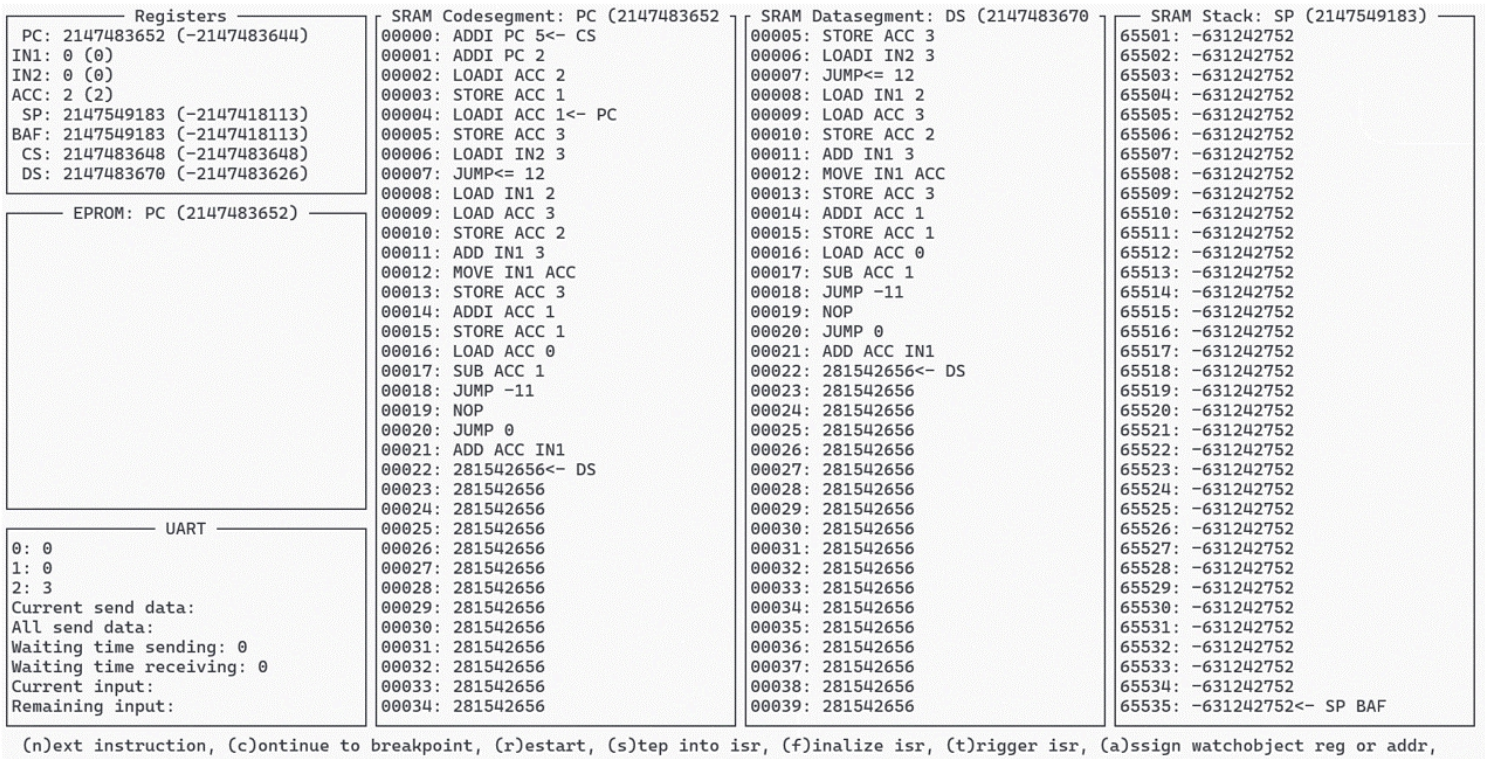

Fig. 3 : Screenshot of the Emulator by Jürgen Mattheis [1]

Emulator by Jürgen Mattheis:

- OS-dependent (only Linux support)

Motivation

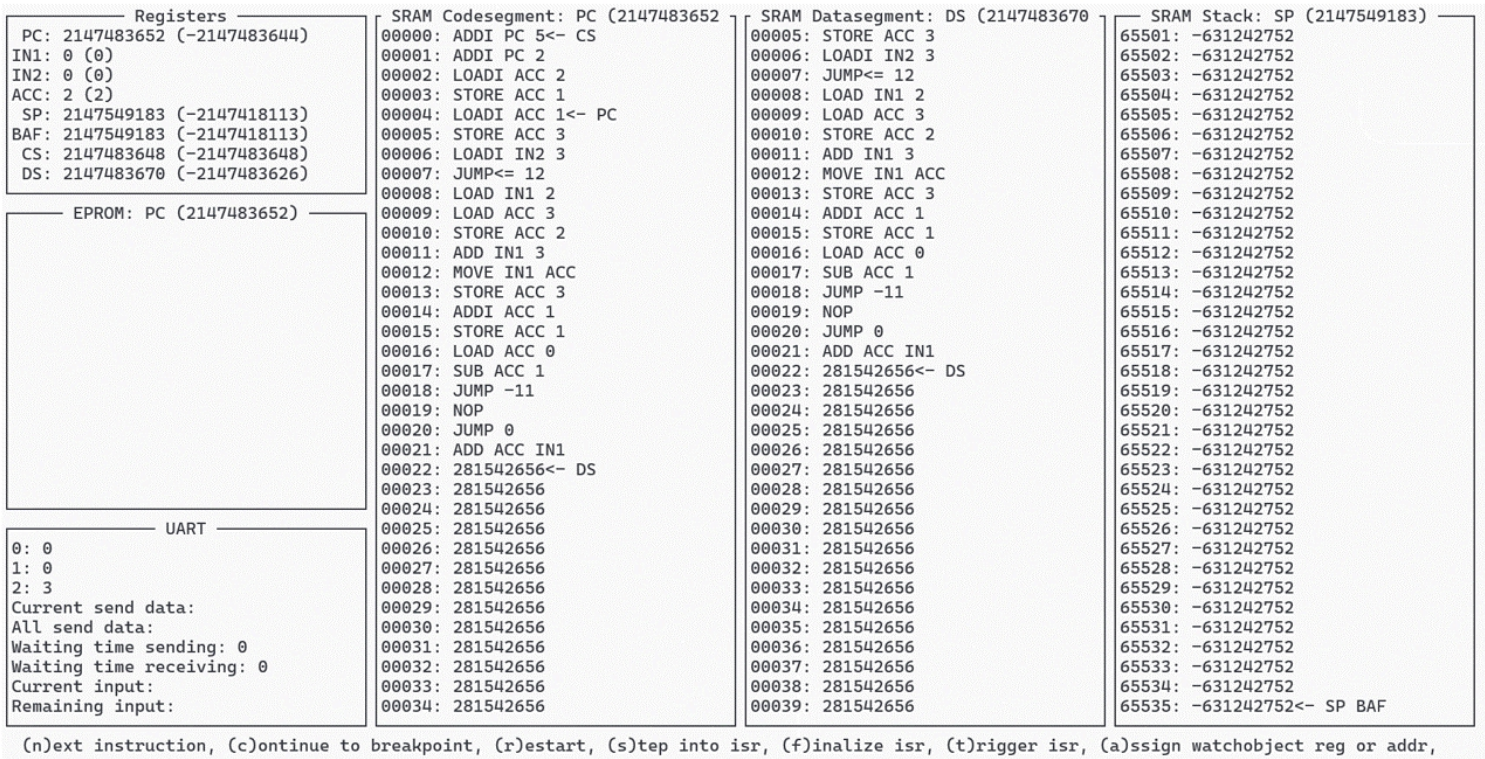

Fig. 3 : Screenshot of the Emulator by Jürgen Mattheis [1]

Emulator by Jürgen Mattheis:

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

Approach

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

Approach

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

Extend existing VS Code extension [6]:

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)
- Make it a single tool supporting both lectures

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)
- Make it a single tool supporting both lectures

Identified key features to extend:

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)
- Make it a single tool supporting both lectures

Identified key features to extend:

- Emulator (implement ReTI-II alongside ReTI-I)

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)
- Make it a single tool supporting both lectures

Identified key features to extend:

- Emulator (implement ReTI-II alongside ReTI-I)
- Debugger (implement ReTI-II alongside ReTI-I)

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)
- Make it a single tool supporting both lectures

Identified key features to extend:

- Emulator (implement ReTI-II alongside ReTI-I)
- Debugger (implement ReTI-II alongside ReTI-I)
- Language Server (implement ReTI-II alongside ReTI-I)

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)
- Make it a single tool supporting both lectures

Identified key features to extend:

- Emulator (implement ReTI-II alongside ReTI-I)
- Debugger (implement ReTI-II alongside ReTI-I)
- Language Server (implement ReTI-II alongside ReTI-I)

Identified additional requirement:

Approach

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

Extend existing VS Code extension [6]:

- Most popular editor/IDE
- OS-independent (available for Linux, Mac, Windows)
- Make it a single tool supporting both lectures

Identified key features to extend:

- Emulator (implement ReTI-II alongside ReTI-I)
- Debugger (implement ReTI-II alongside ReTI-I)
- Language Server (implement ReTI-II alongside ReTI-I)

Identified additional requirement:

- Provide a memory view

Approach

Extension's Architecture

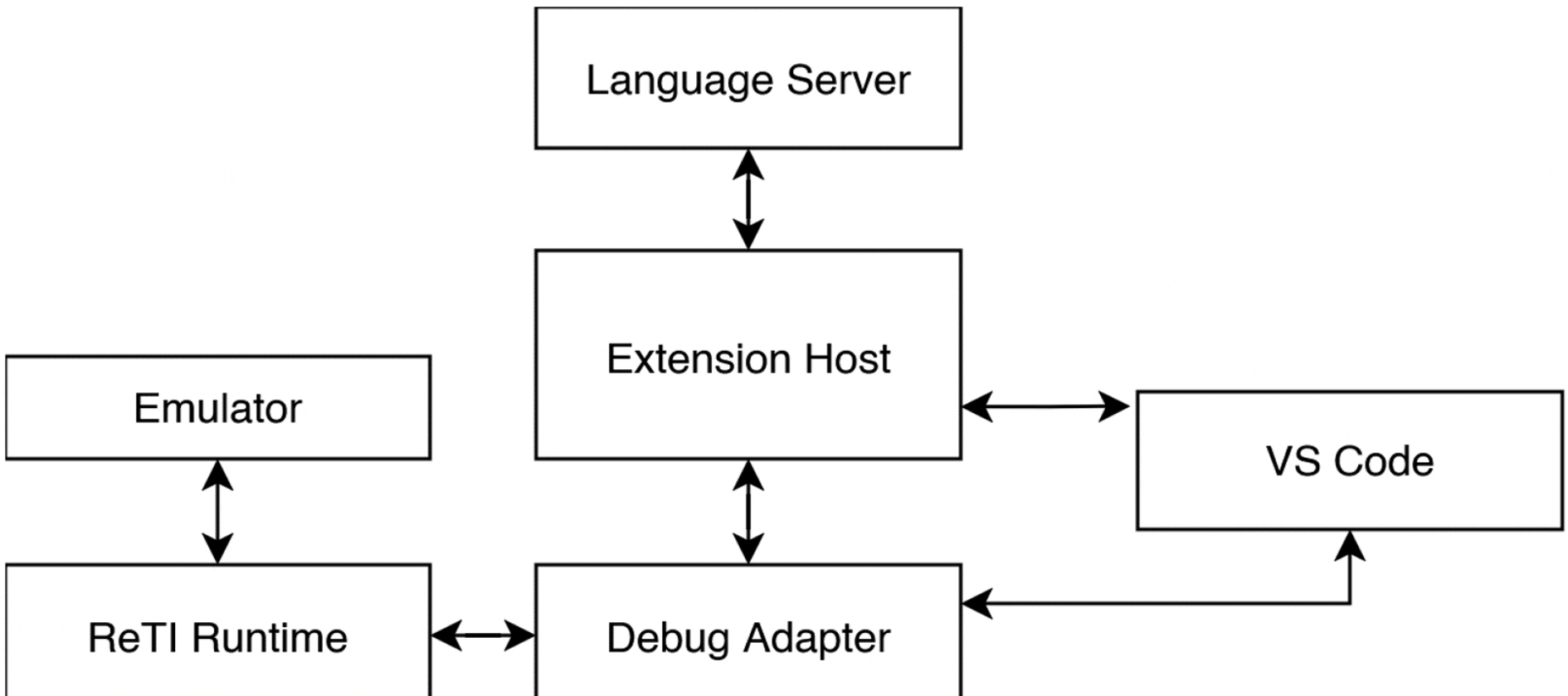

Fig. 4: Diagram illustrating the interaction between the different components of the extension.

Approach

Extension's Architecture

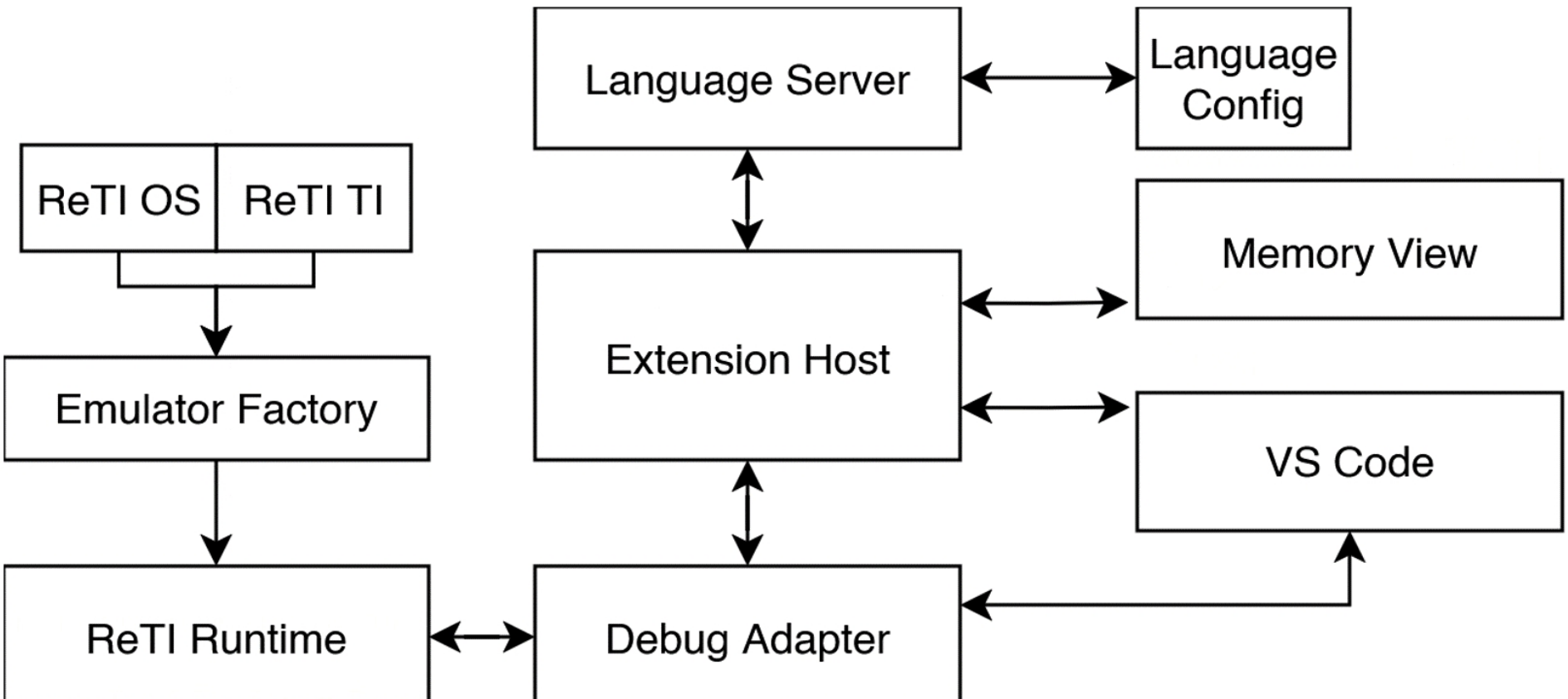

Fig. 4: Diagram illustrating the interaction between the different components of the extension.

Results

Support for Both Architectures

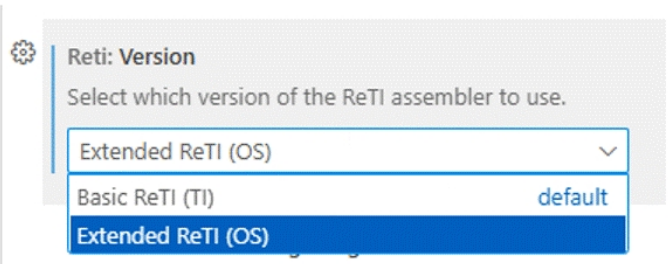

Fig 5: Screenshot of the new setting in VS Code

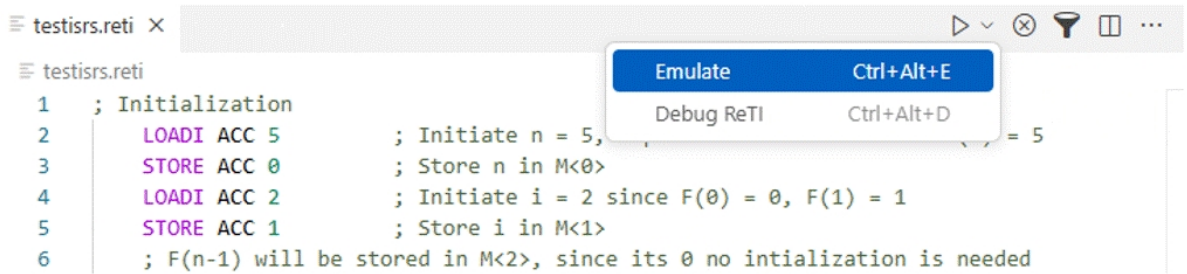

Fig 6: Example Usage of the ReTI Emulation in VS Code

Results

Support for Both Architectures

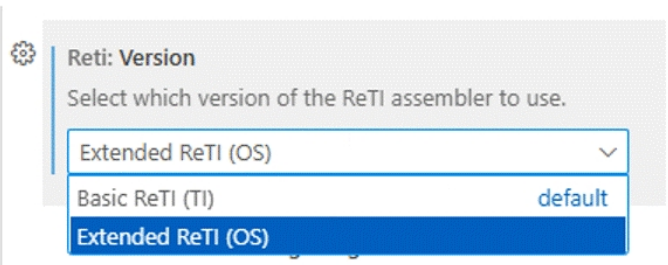

Fig 5: Screenshot of the new setting in VS Code

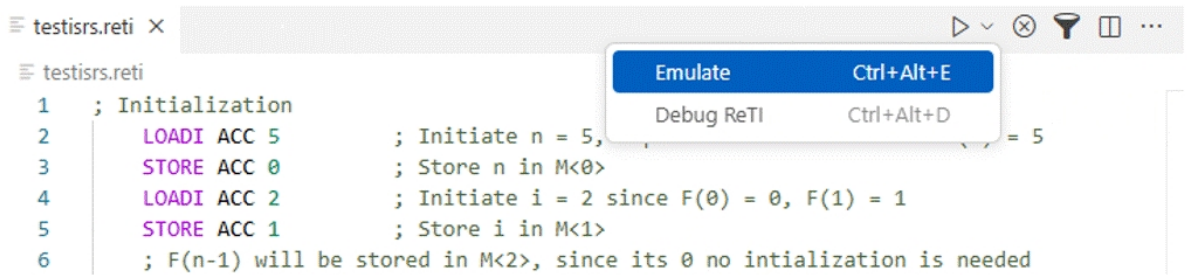

Fig 6: Example Usage of the ReTI Emulation in VS Code

- Added setting to specify desired version

Results

Support for Both Architectures

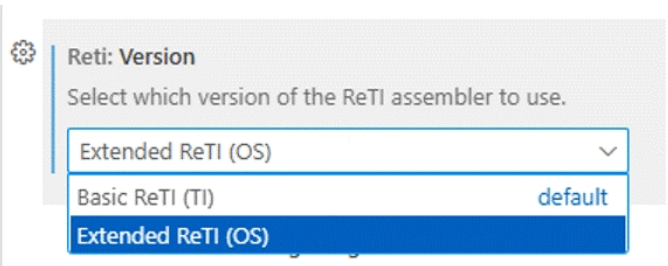

Fig 5: Screenshot of the new setting in VS Code

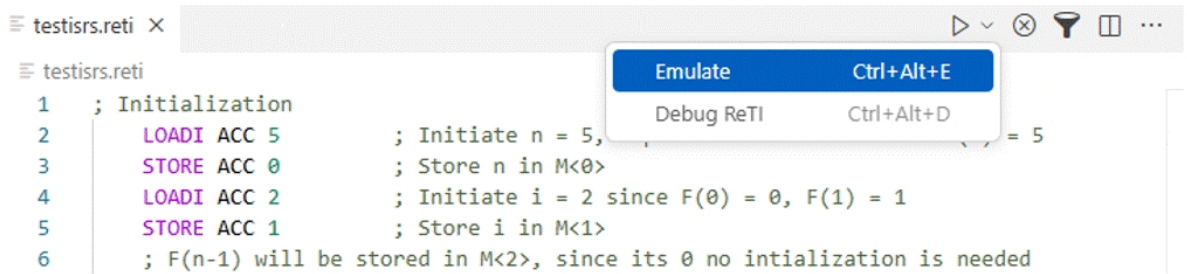

Fig 6: Example Usage of the ReTI Emulation in VS Code

- Added setting to specify desired version
- Affects all features (emulator, debugger, language server) except quiz

Results

Support for Both Architectures

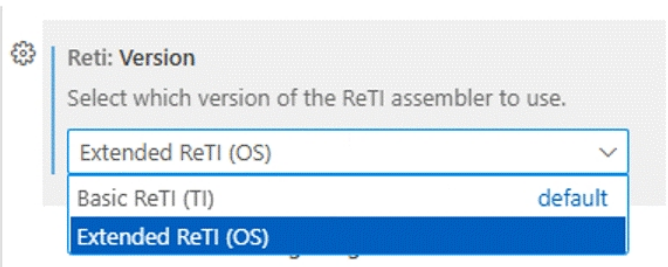

Fig 5: Screenshot of the new setting in VS Code

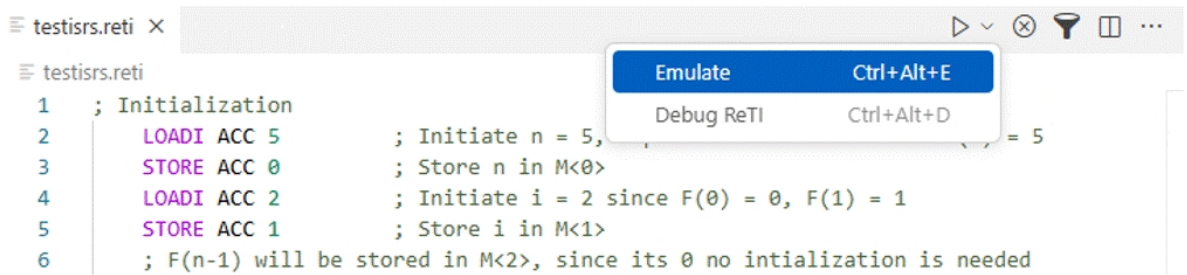

Fig 6: Example Usage of the ReTI Emulation in VS Code

- 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       STO         ; Store n in M<0>    Unknown instruction "STO".
4       LOA
5       STO
6       ; F(n-1) will be stored in M<2>, since its 0 no intializati
7                 ored in M<3>
8
9
10                    ; 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
```

STORE

STOREIN

STORE S i

Stores the value of the specified register into the i-th memory cell.

INT

Usage: INT i

Result: PC := IVT[i]

Fig 7: Screenshot from VS Code highlighting LSP Features

Results

Language Server

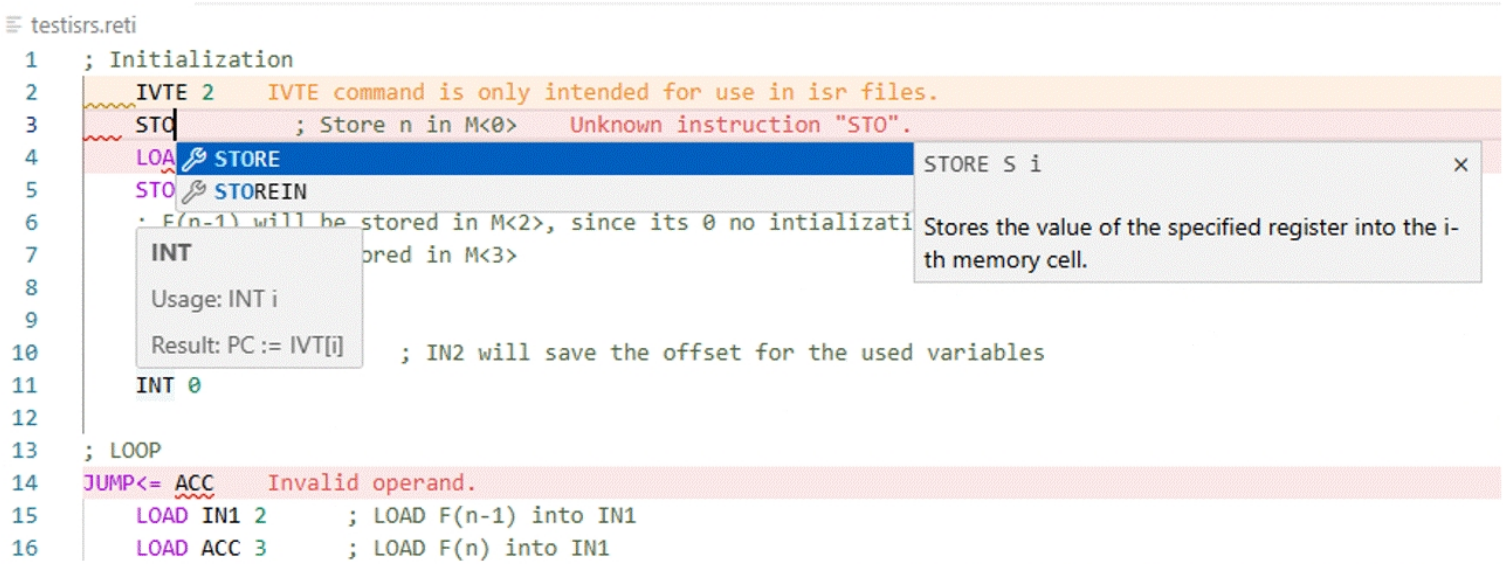

Fig 7: Screenshot from VS Code highlighting LSP Features

Features:

Results

Language Server

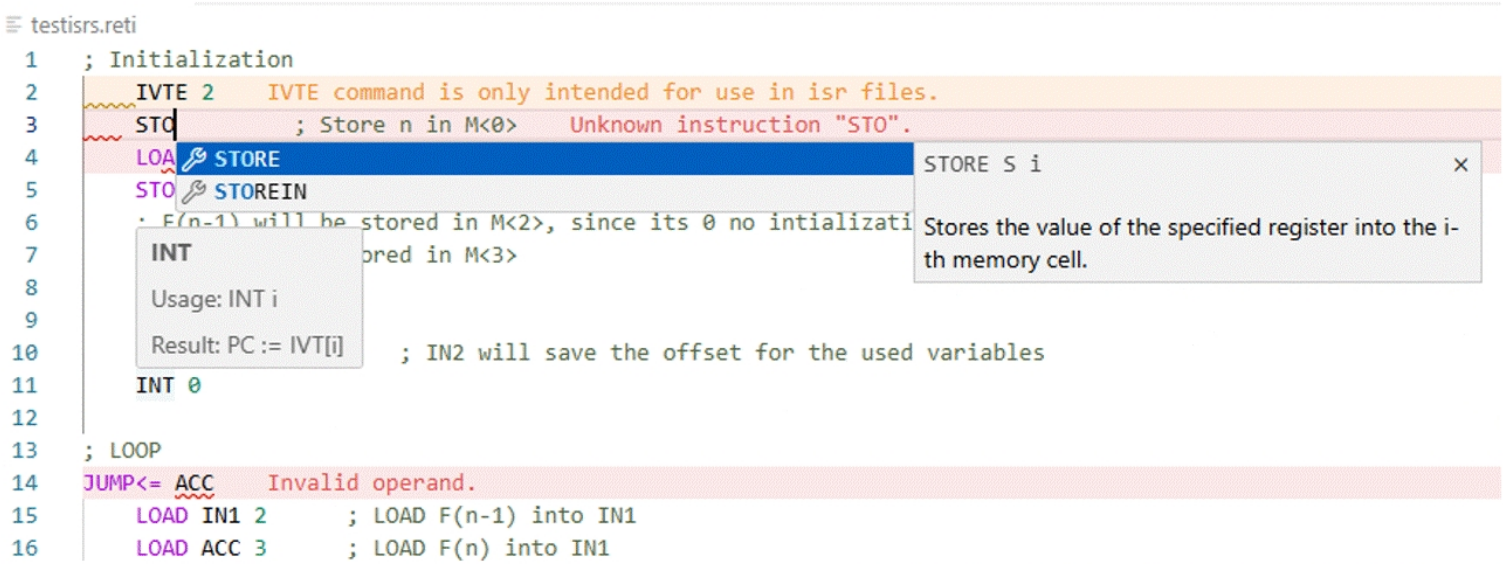

Fig 7: Screenshot from VS Code highlighting LSP Features

Features:

- Syntax Highlighting

Results

Language Server

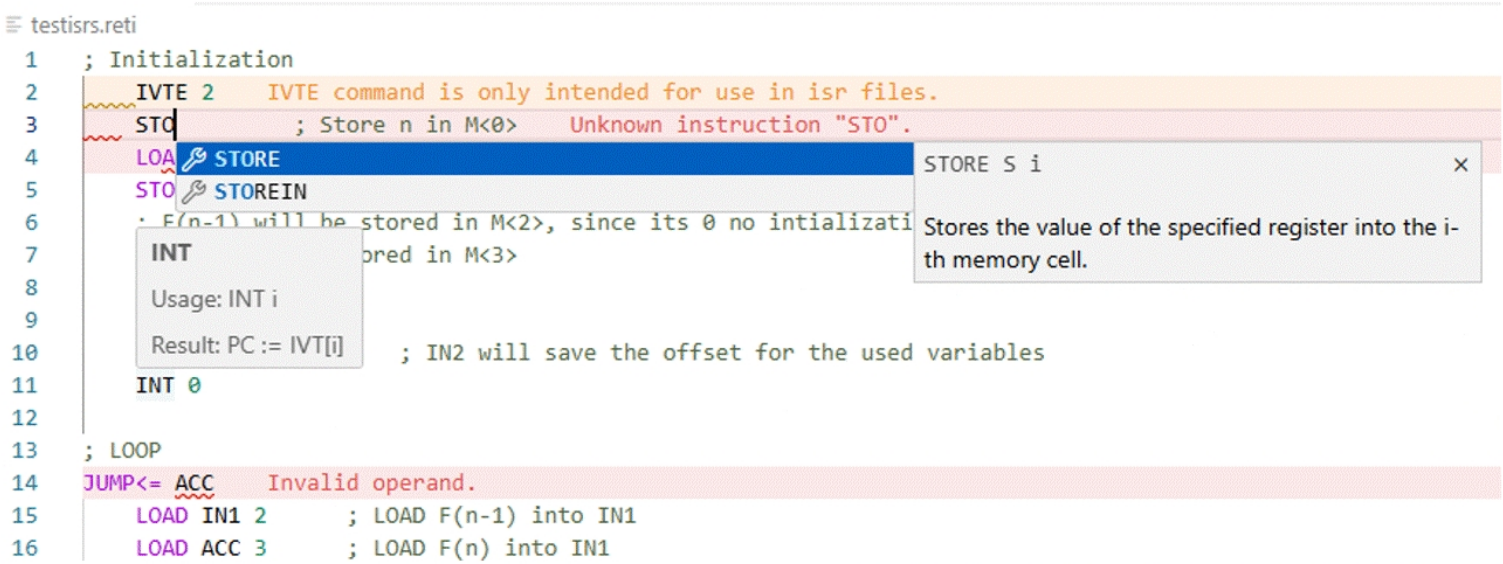

Fig 7: Screenshot from VS Code highlighting LSP Features

Features:

- Syntax Highlighting
- Autocomplete suggestions

Results

Language Server

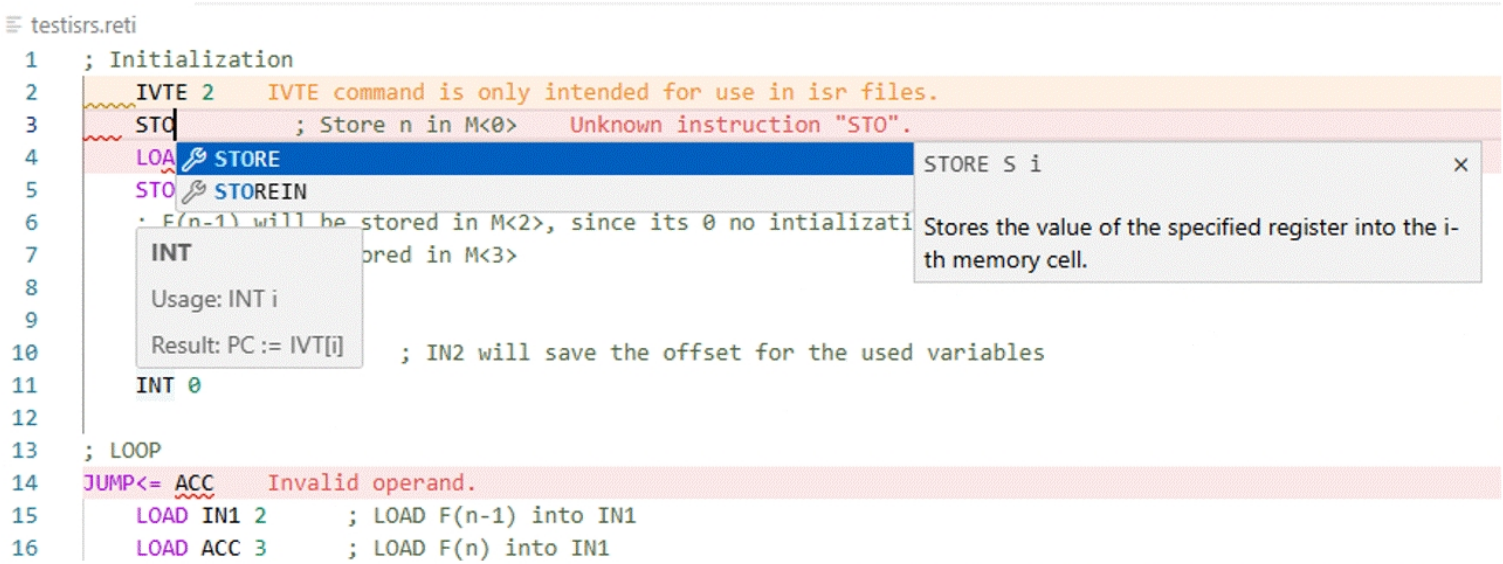

Fig 7: Screenshot from VS Code highlighting LSP Features

Features:

- Syntax Highlighting
- Autocomplete suggestions
- Tooltips (syntax and documentation)

Results

Language Server

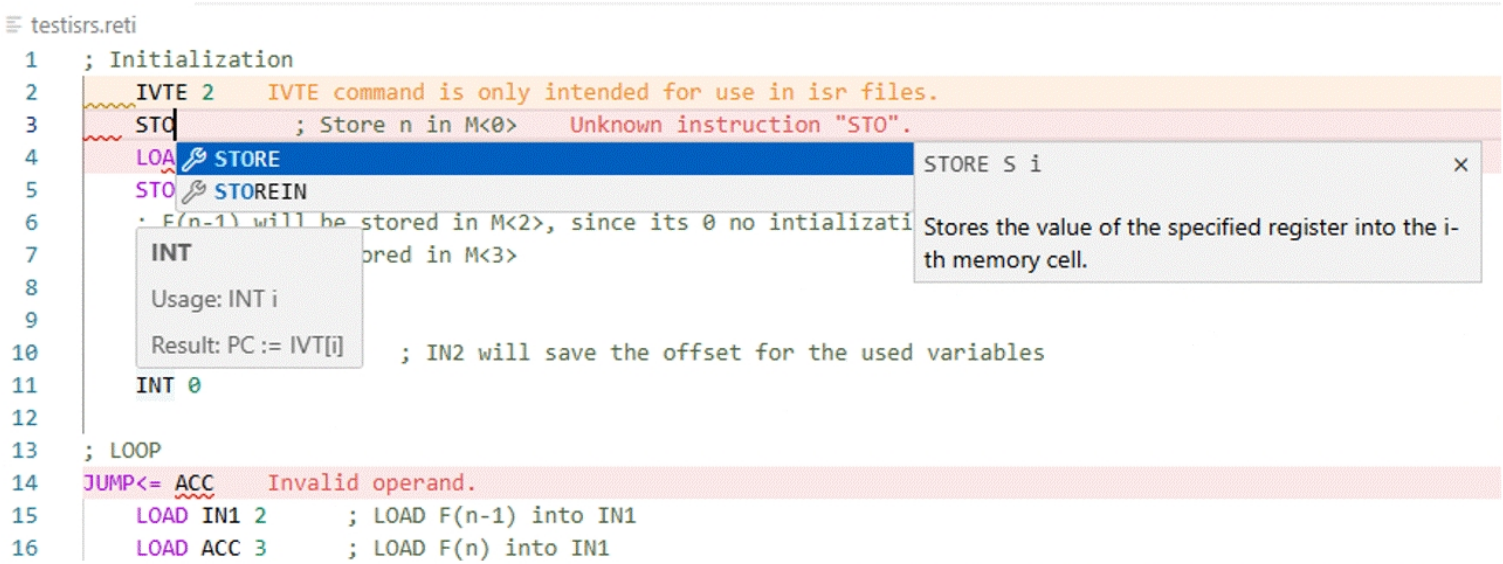

Fig 7: Screenshot from VS Code highlighting LSP Features

Features:

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

Results

Debugger and Memory View

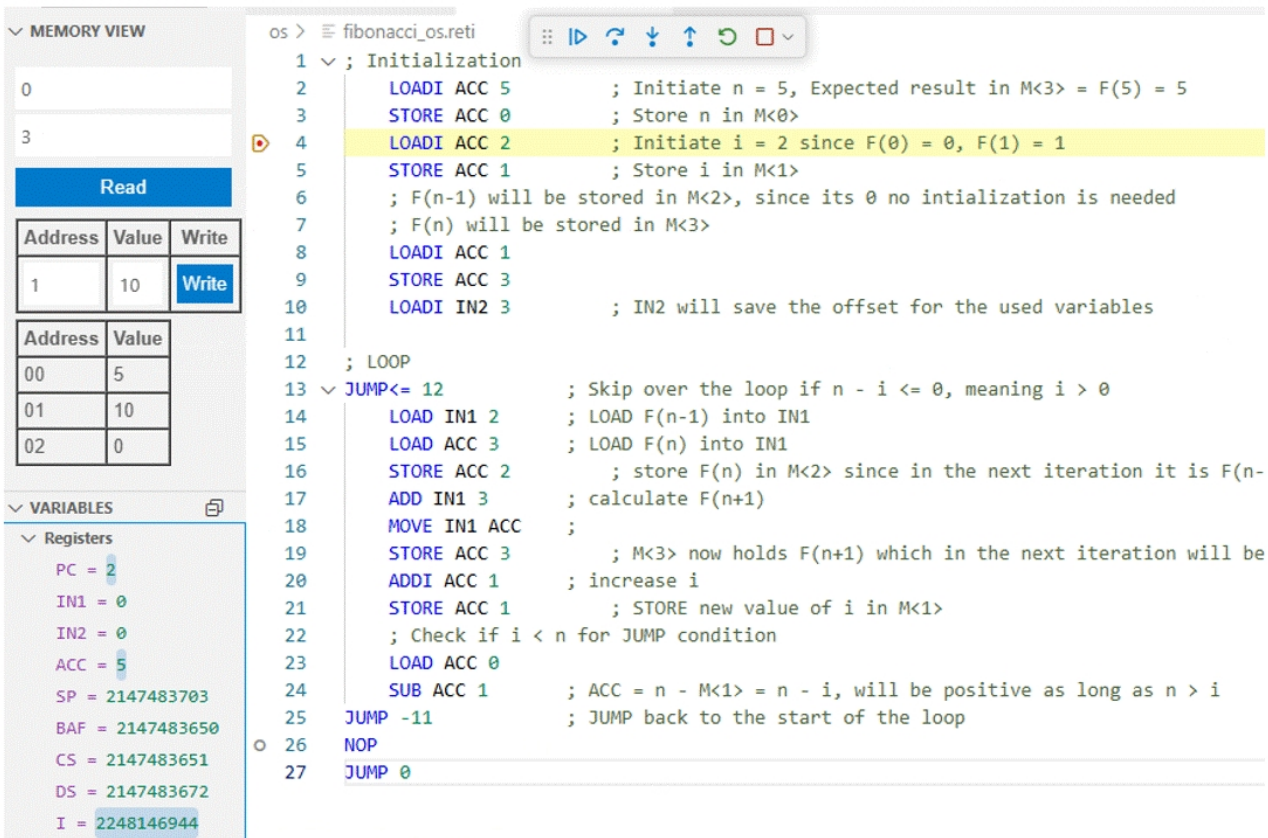

Fig 8: Screenshot of running ReTI-Debug session

Results

Debugger and Memory View

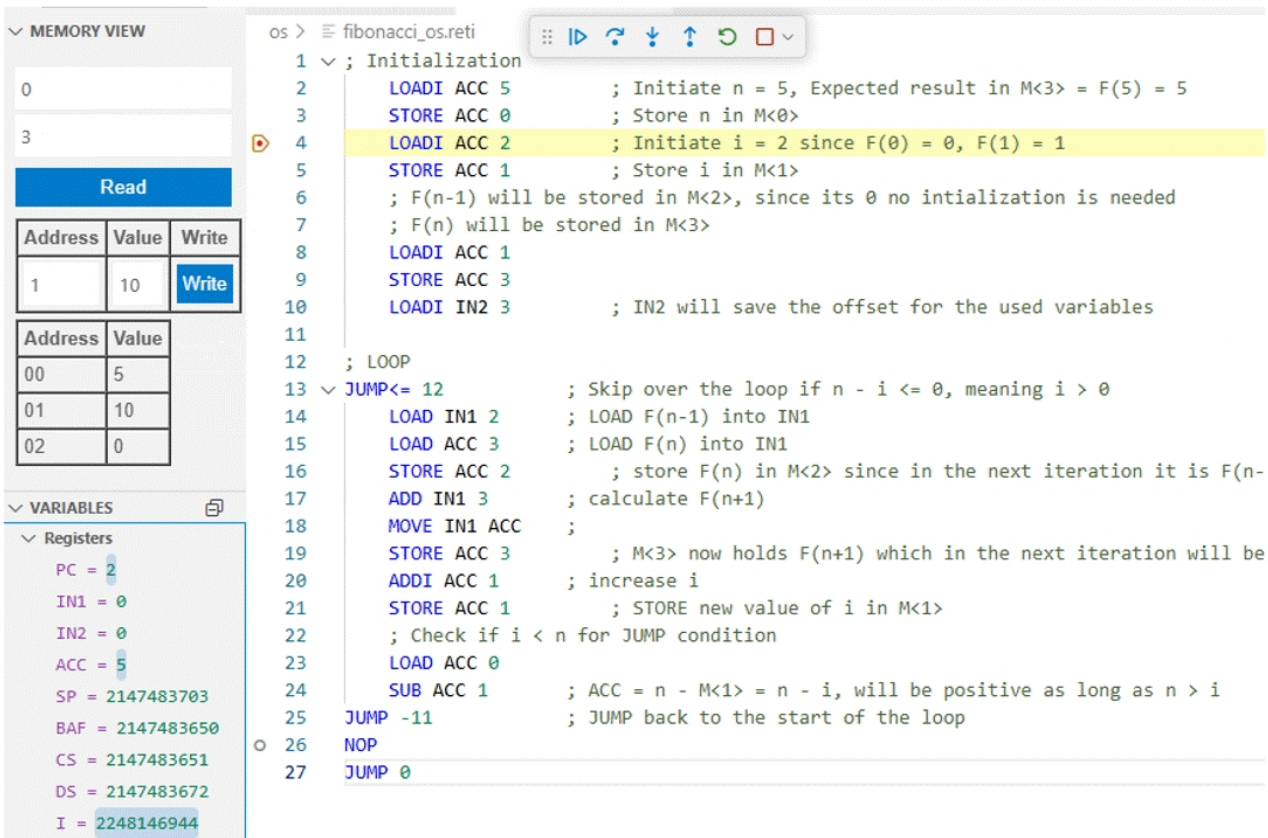

Fig 8: Screenshot of running ReTI-Debug session

Features both architectures:

Results

Debugger and Memory View

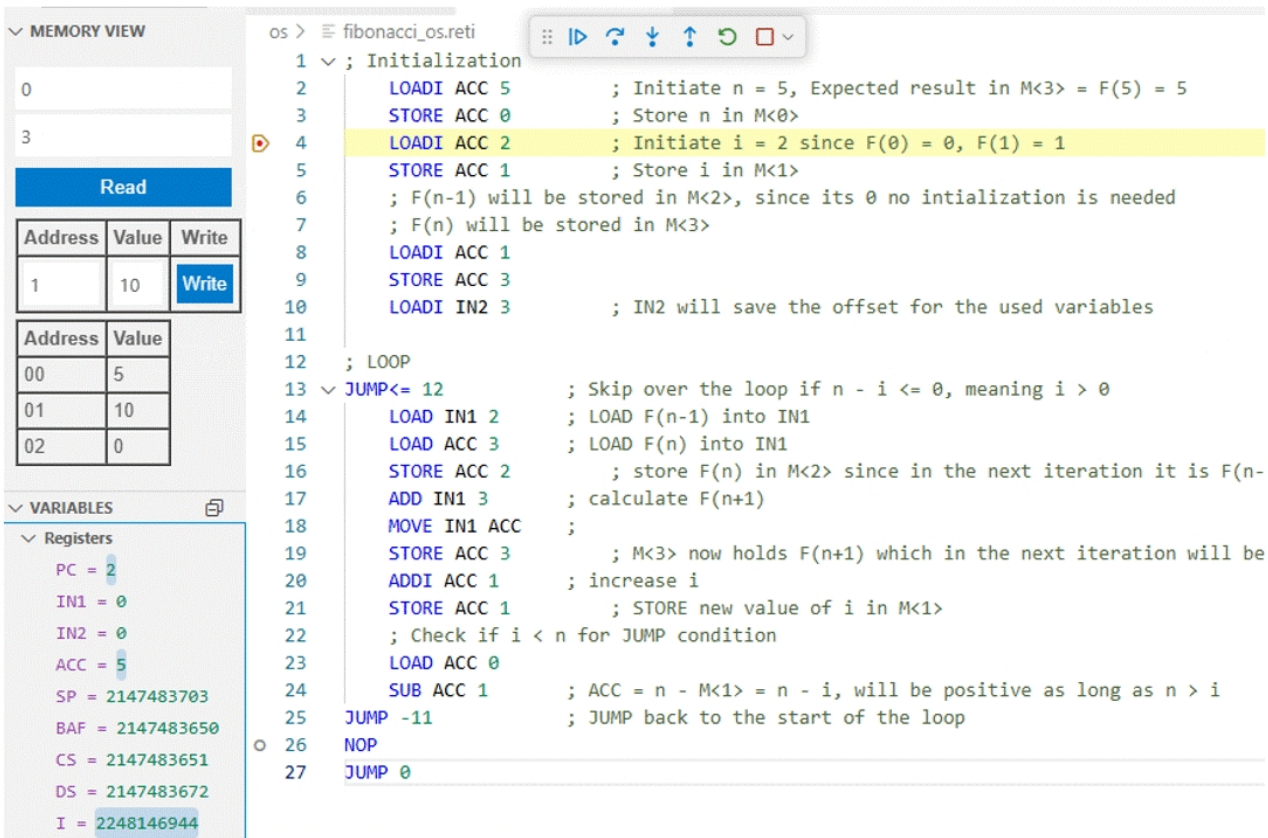

Fig 8: Screenshot of running ReTI-Debug session

Features both architectures:

- Reading and writing register values

Results

Debugger and Memory View

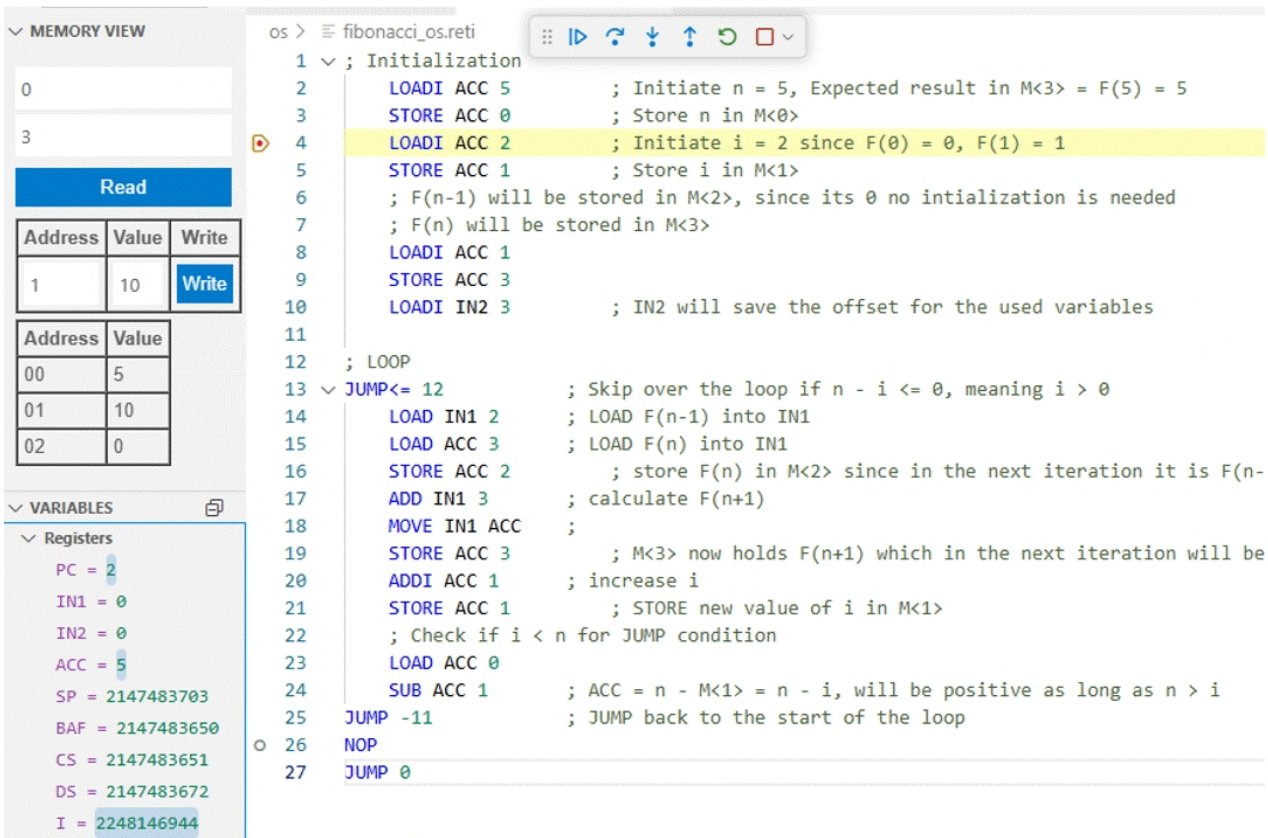

Fig 8: Screenshot of running ReTI-Debug session

Features both architectures:

- Reading and writing register values
- Reading and writing memory

Results

Debugger and Memory View

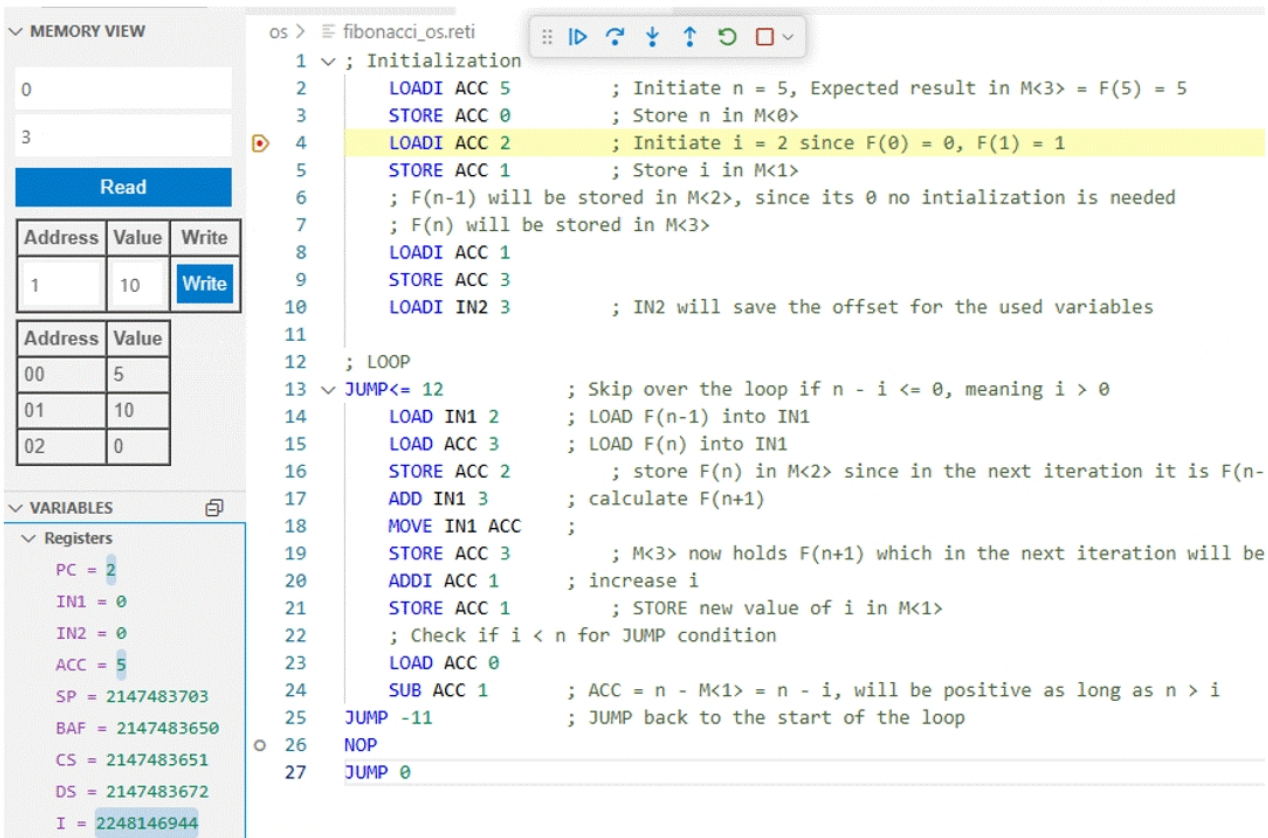

Fig 8: Screenshot of running ReTI-Debug session

Features both architectures:

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

Results

Debugger and Memory View

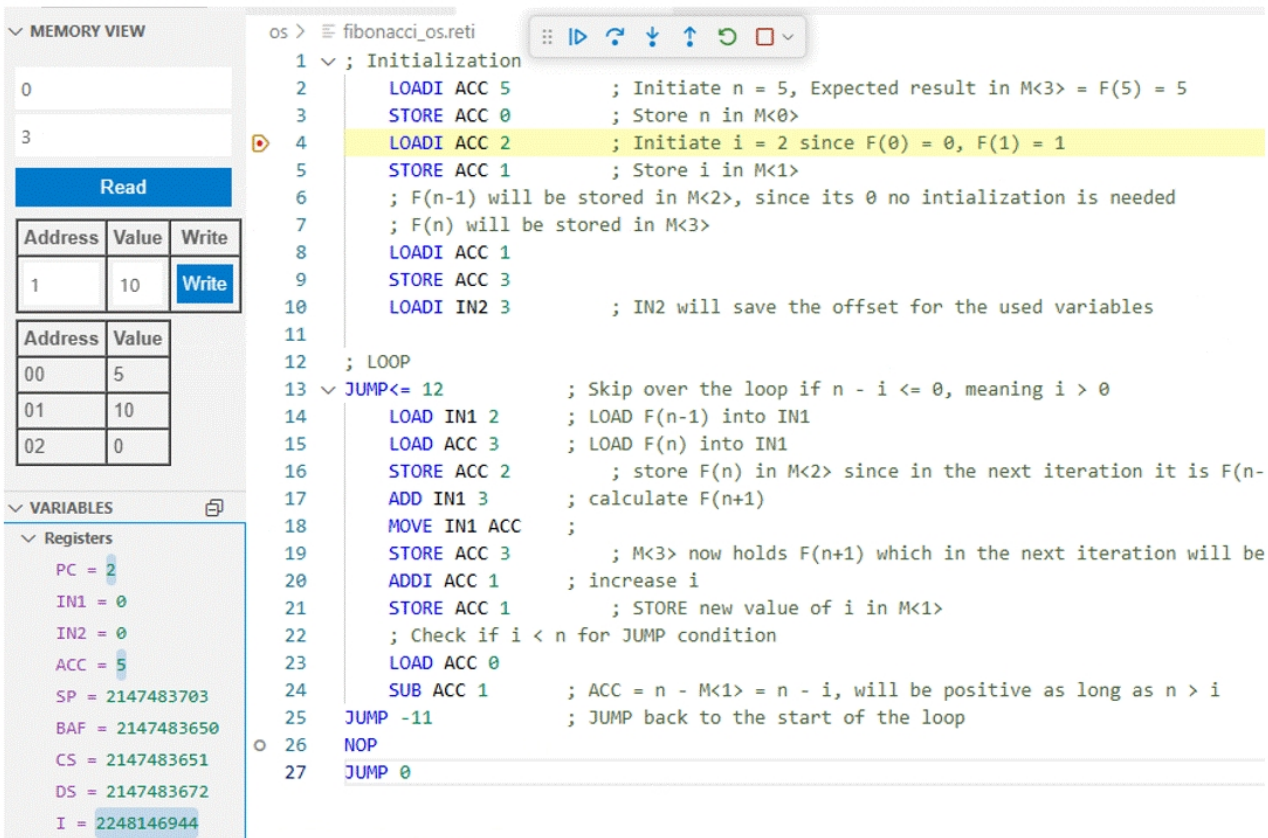

Fig 8: Screenshot of running ReTI-Debug session

Features both architectures:

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

Features for the ReTI-II (OS):

Results

Debugger and Memory View

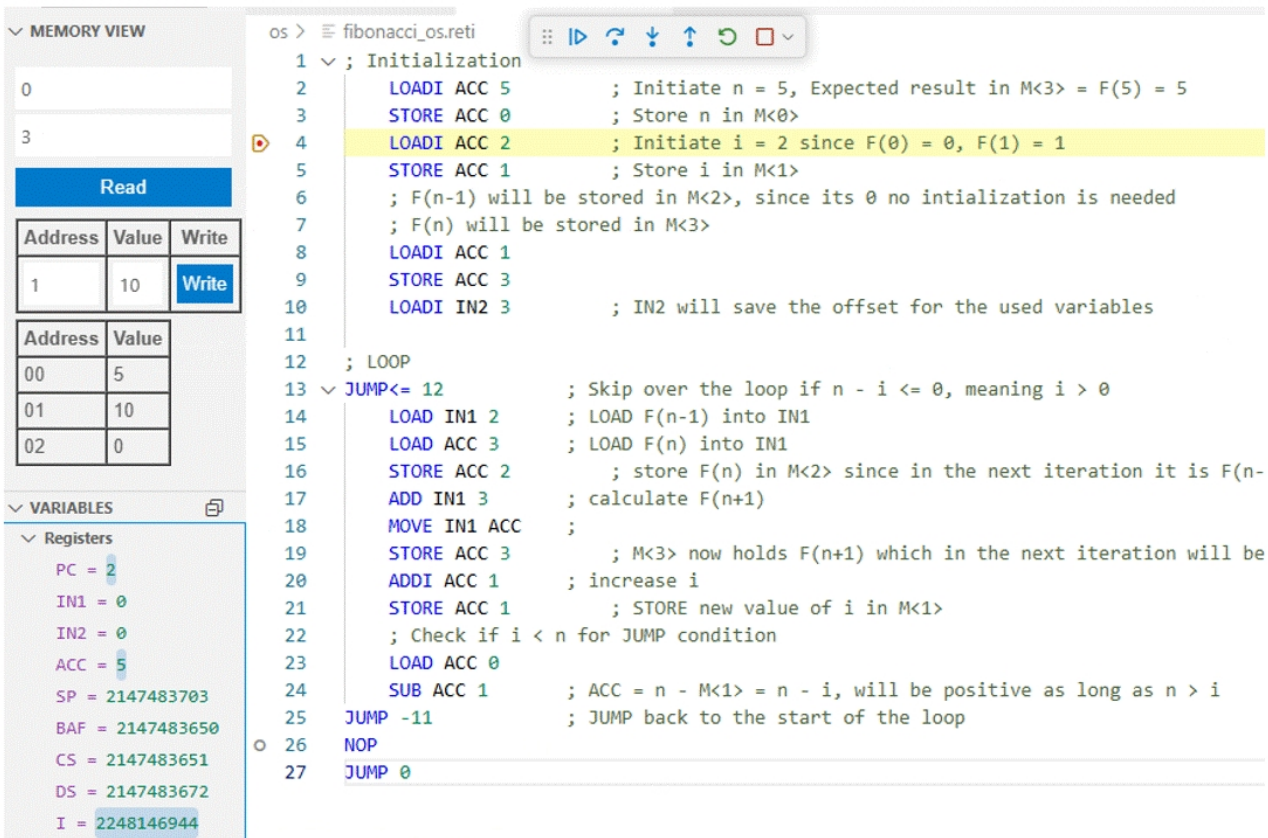

Fig 8: Screenshot of running ReTI-Debug session

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

Results

Debugger and Memory View

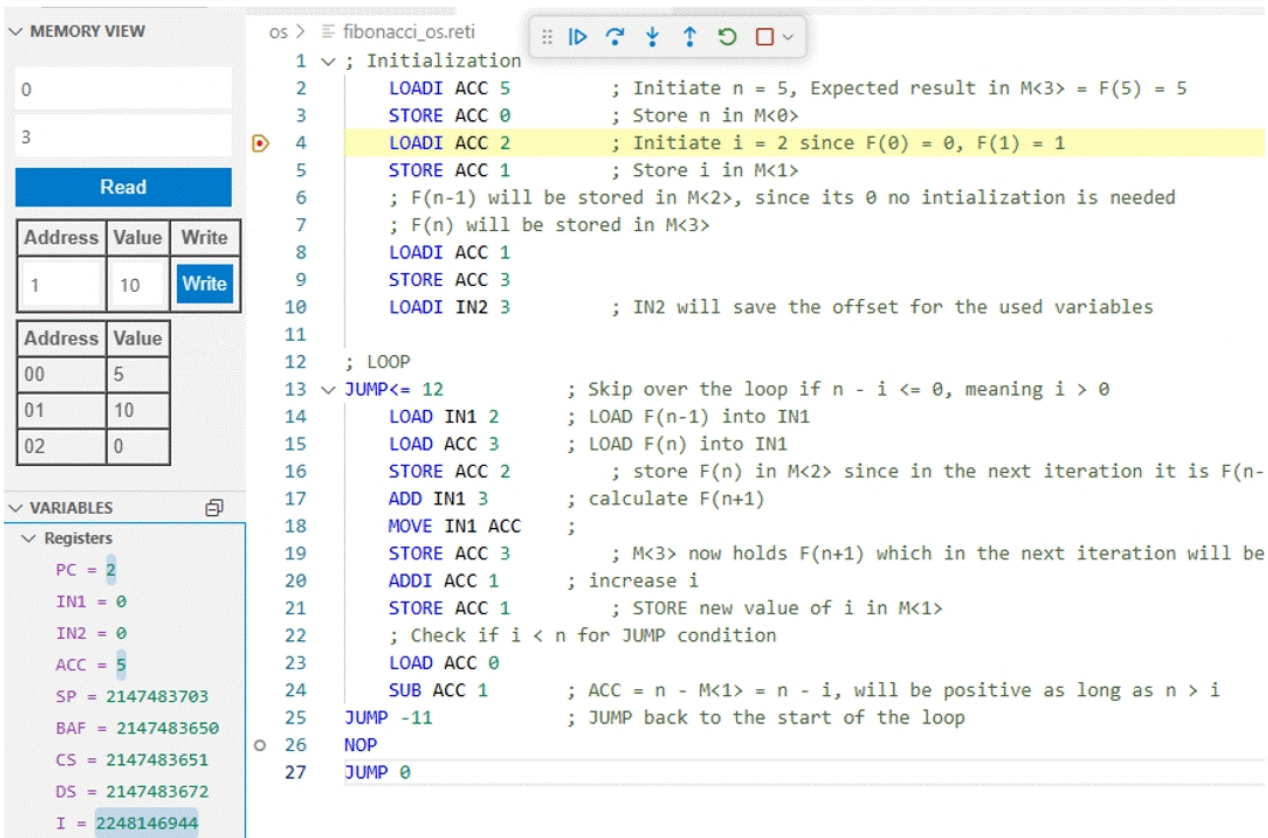

Fig 8: Screenshot of running ReTI-Debug session

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

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

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 intialization 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
```

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. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

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 intialization 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
```

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. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Results

Debugger, Stepping

testisrs.reti

```
; Initialization
    LOADI ACC 5         ; Initiate n = 5, Expected result in M<3> = F(5) = 5
    STORE ACC 0         ; Store n in M<0>
    LOADI ACC 2         ; Initiate i = 2 since F(0) = 0, F(1) = 1
    STORE ACC 1         ; Store i in M<1>
    ; F(n-1) will be stored in M<2>, since its 0 no intialization is needed
    ; F(n) will be stored in M<3>
    LOADI ACC 1
    STORE ACC 3
    LOADI IN2 3         ; IN2 will save the offset for the used variables
    INT 0

; LOOP
JUMP<= 12           ; Skip over the loop if n - i <= 0, meaning i > 0
    LOAD IN1 2      ; LOAD F(n-1) into IN1
    LOAD ACC 3      ; LOAD F(n) into IN1
    STORE ACC 2         ; store F(n) in M<2> since in the next iteration it is F(n-1)
    ADD IN1 3       ; calculate F(n+1)
    MOVE IN1 ACC    ;
    NOP
    STORE ACC 3         ; M<3> now holds F(n+1) which in the next iteration will be F(n)
    ADDI ACC 1      ; increase i
    STORE ACC 1         ; STORE new value of i in M<1>
    ; Check if i < n for JUMP condition
    LOAD ACC 0
    INT 1
    SUB ACC 1       ; ACC = n - M<1> = n - i, will be positive as long as n > i
JUMP -11            ; JUMP back to the start of the loop
NOP
JUMP 0
```

isrs.reti

```
; Example routine that restores registers
IVTE 0
SUBI SP 7
STOREIN SP IN1 7
STOREIN SP IN2 6
STOREIN SP ACC 5

STOREIN SP CS 2
STOREIN SP DS 1

LOADI IN1 2
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1

LOADIN SP DS 1
LOADIN SP CS 2
LOADIN SP ACC 5
LOADIN SP IN2 6
LOADIN SP IN1 7
ADDI SP 7
RTI
; Example routine that keeps register values.
IVTE 1
LOADI ACC 2
MOVE ACC IN1
LOADI IN2 5
SUB IN2 IN1
LOADI ACC 1
RTI
```

Fig. 9: Screenshots of running ReTI-Debug session

Conclusion
Goals Reached

Conclusion
Goals Reached

- All features (emulator, debugger, language server) extended to cover both versions ✓

Conclusion
Goals Reached

- All features (emulator, debugger, language server) extended to cover both versions ✓
- Memory view allows easier monitoring and manipulation of memory ✓

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 ✓

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

Conclusion

Limitations & Future Work

- **Emulate UART interface**

Conclusion

Limitations & Future Work

- **Emulate UART interface**
- **Make debug adapter independent from VS Code (embedding in other editors/tools)**

Conclusion

Limitations & Future Work

- **Emulate UART interface**
- **Make debug adapter independent from VS Code (embedding in other editors/tools)**
- **Integrate Pico-C compiler**

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]:**

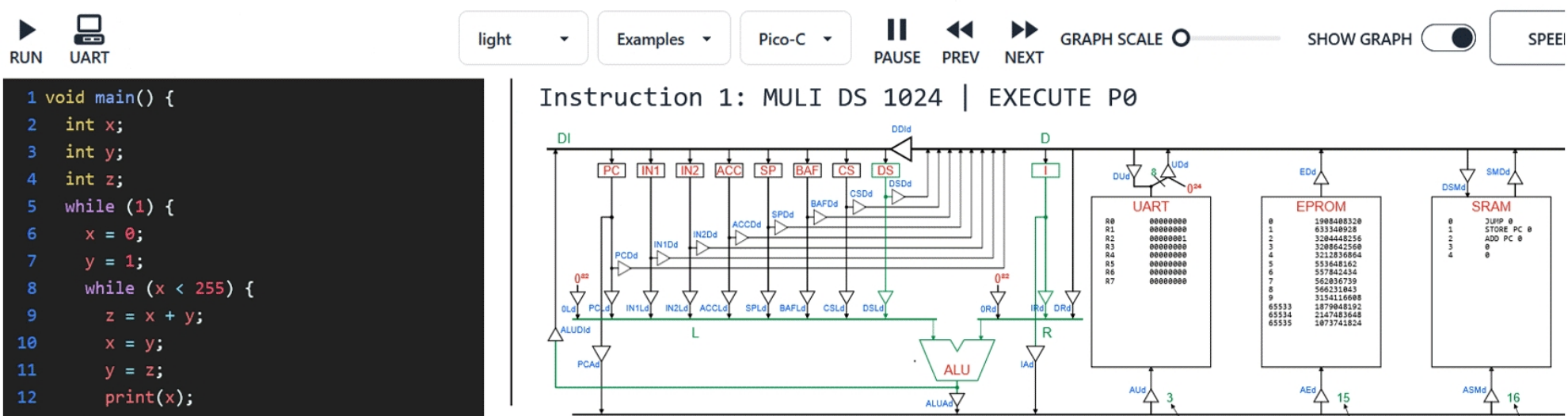

Fig. 10: Screenshot of ReTI-Emulator with Datapath-Visualization [3]

Extra:
Language-Server-Protocol

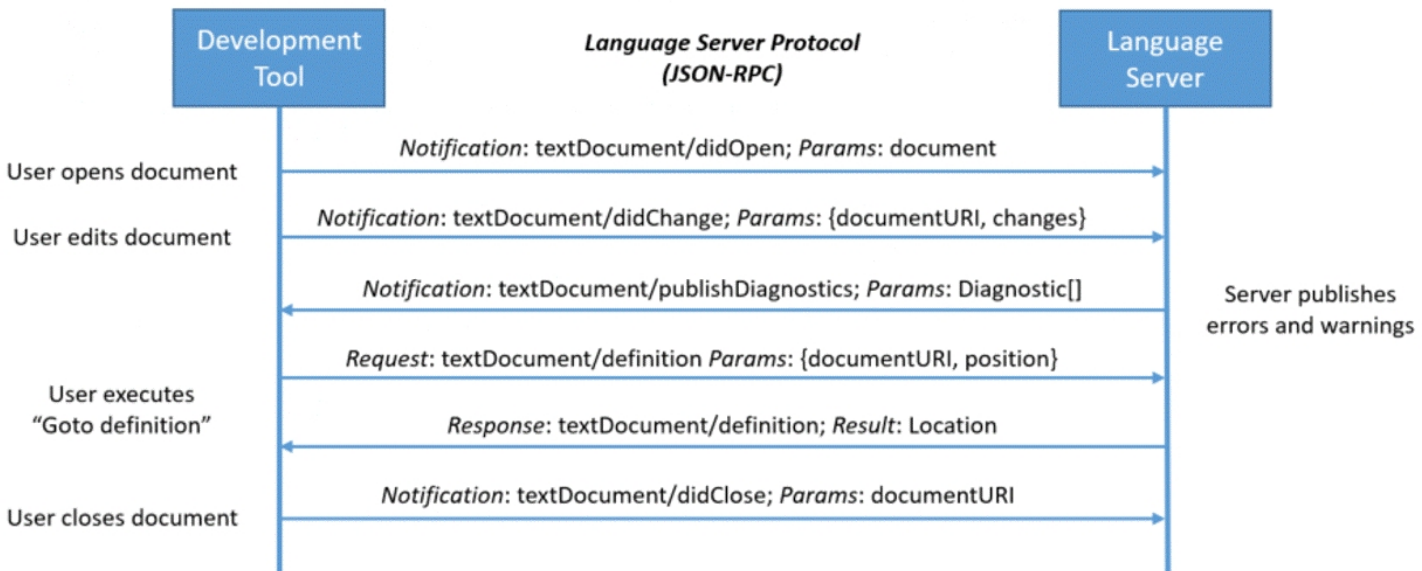

Fig. 11: Example for Communication between Tool and Server in the Language-Server-Protocol [4]

Extra:
Debug-Adapter-Protocol

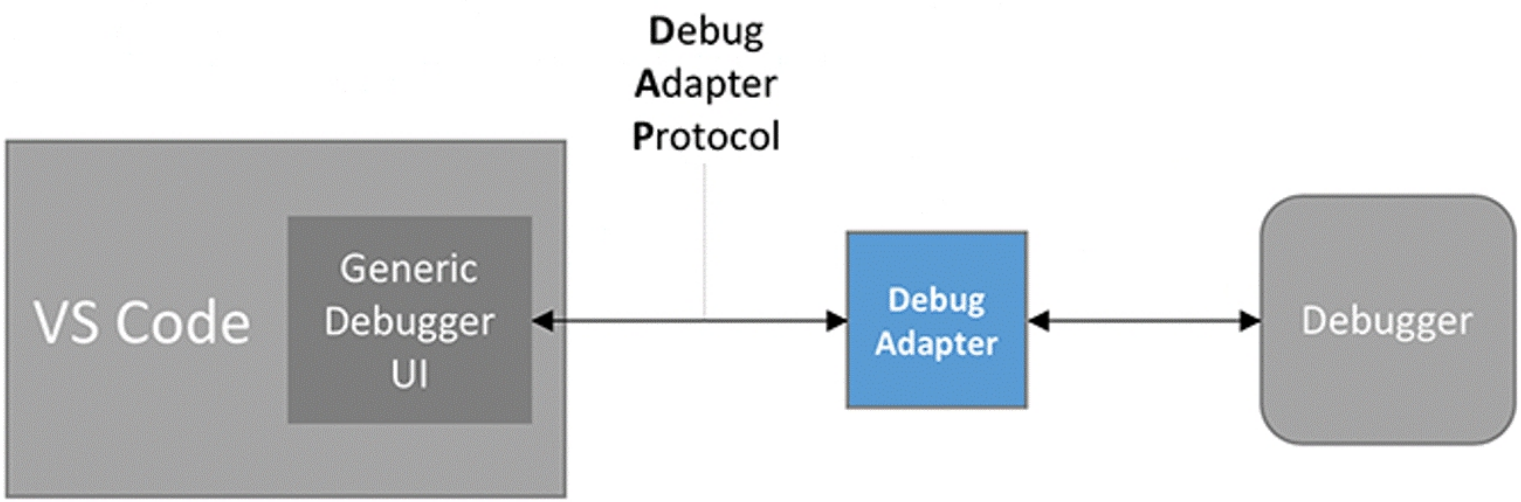

Fig. 12: VS Code Debug Architecture [5]

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
[6]: ReTI-Tools by Malte Pullich, https://github.com/mlt279/vscode-reti