

ARM Simulator, Interpreter und Debugger als Webanwendung

Initialpräsentation

Zangerl Dominik

Betreuer: Alexander Schlögl

Gliederung

- Motivation
- Implementation und Technologien
 - Typescript
 - Parser
 - Simulator und Debugger
 - Frontend/React
- Vorgehensweise und Zeitplan
- Voraussetzungen für finale Implementierung
- Referenzen

ARMv5 im ersten Semester

- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur

ARMv5 im ersten Semester

- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur

ARMv5 im ersten Semester

- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [\[1\]](#):

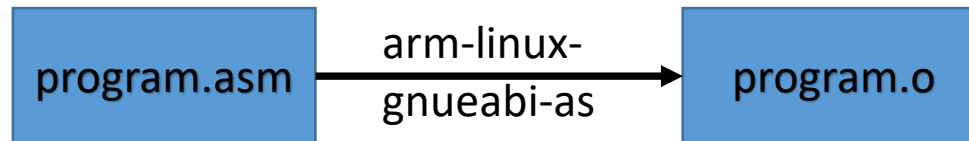
ARMv5 im ersten Semester

- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [\[1\]](#):

```
program.asm
```

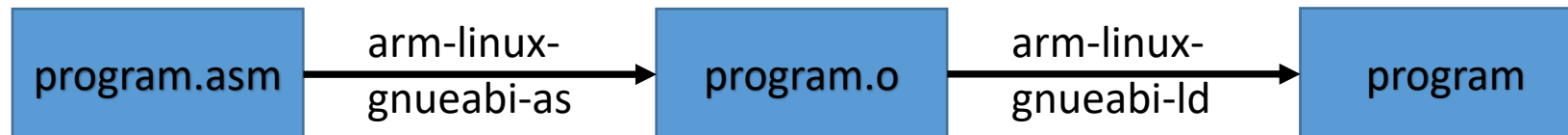
ARMv5 im ersten Semester

- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [\[1\]](#):



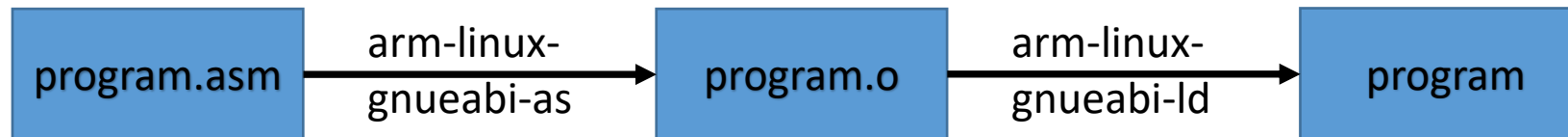
ARMv5 im ersten Semester

- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [\[1\]](#):



ARMv5 im ersten Semester

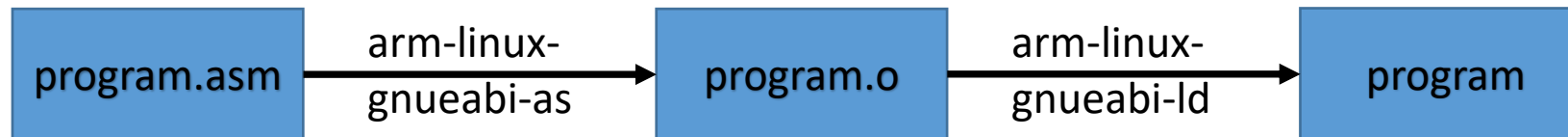
- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [\[1\]](#):



- Ausführen mit QEMU User-Space-Emulator [\[9\]](#)

ARMv5 im ersten Semester

- ARMv5 [\[2\]](#) als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [\[1\]](#):



- Ausführen mit QEMU User-Space-Emulator [\[9\]](#)
- Vereinfachung mit Skript und Ausführung über virtuelle Maschine oder WSL [\[6\]](#)

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [\[8\]](#) verwendet werden:

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [\[8\]](#) verwendet werden:

```
Register group: general
r0      0xfdbbf7af      -38013009      r1      0xfdffffe5      -33554459
r2      0x3             3              r3      0x4             4
r4      0x7             7              r5      0xffdffffd4     -2097196
r6      0x4c4d5b53      1280138067     r7      0x8410de10      -2079269360
r8      0x37feffff      939458558     r9      0xffedfffc      -1179652
r10     0xfbaad45e      -72690594     r11     0x88cad3c4      -1999973436
r12     0xfdfaafff      -33882113     sp      0x0             0x0
lr      0xffffffff      -1            pc      0x1c          0x1c <_start>
xPSR    0x1000000      16777216
```

Bild: Use GDB on an ARM assembly program [\[7\]](#)

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [\[8\]](#) verwendet werden:

```
Register group: general
r0      0xfdbbf7af      -38013009      r1      0xfdffffe5      -33554459
r2      0x3             3              r3      0x4             4
r4      0x7             7              r5      0xffdffffd4     -2097196
r6      0x4c4d5b53      1280138067     r7      0x8410de10      -2079269360
r8      0x37feffffe     939458558     r9      0xffedfffc      -1179652
r10     0xfbaad45e      -72690594     r11     0x88cad3c4      -1999973436
r12     0xfdfaafff      -33882113     sp      0x0             0x0
lr      0xffffffff      -1            pc      0x1c          0x1c <_start>
xPSR    0x1000000      16777216
```

Bild: Use GDB on an ARM assembly program [\[7\]](#)

- Arbeiten mit Debuggern im ersten Semester oft schwierig

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [\[8\]](#) verwendet werden:

```
Register group: general
r0      0xfdbbf7af      -38013009      r1      0xfdffffe5      -33554459
r2      0x3             3              r3      0x4             4
r4      0x7             7              r5      0xffdffffd4     -2097196
r6      0x4c4d5b53      1280138067     r7      0x8410de10      -2079269360
r8      0x37feffff      939458558      r9      0xffedfffc      -1179652
r10     0xfbaad45e      -72690594      r11     0x88cad3c4      -1999973436
r12     0xfdfaafff      -33882113      sp      0x0             0x0
lr      0xffffffff      -1             pc      0x1c          0x1c <_start>
xPSR    0x1000000      16777216
```

Bild: Use GDB on an ARM assembly program [\[7\]](#)

- Arbeiten mit Debuggern im ersten Semester oft schwierig
- Großer Zeitaufwand zusammen mit Aufsetzen der Toolchain

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
 - Simulierte CPU und Hauptspeicher

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
 - Simulierte CPU und Hauptspeicher
 - Assembler-Code direkt in Anwendung schreiben und ausführen → ersetzt Toolchain

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
 - Simulierte CPU und Hauptspeicher
 - Assembler-Code direkt in Anwendung schreiben und ausführen → ersetzt Toolchain
 - Dauerhafte Anzeige von Registern und Stacks

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
 - Simulierte CPU und Hauptspeicher
 - Assembler-Code direkt in Anwendung schreiben und ausführen → ersetzt Toolchain
 - Dauerhafte Anzeige von Registern und Stacks
- Debugger

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
 - Simulierte CPU und Hauptspeicher
 - Assembler-Code direkt in Anwendung schreiben und ausführen → ersetzt Toolchain
 - Dauerhafte Anzeige von Registern und Stacks
- Debugger
 - Breakpoints

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
 - Simulierte CPU und Hauptspeicher
 - Assembler-Code direkt in Anwendung schreiben und ausführen → ersetzt Toolchain
 - Dauerhafte Anzeige von Registern und Stacks
- Debugger
 - Breakpoints
 - Zeilenweise Abarbeitung

TypeScript

- TypeScript [\[5\]](#) ist eine Sprache von Microsoft, die auf JavaScript aufbaut

TypeScript

- TypeScript [\[5\]](#) ist eine Sprache von Microsoft, die auf JavaScript aufbaut
- JavaScript überprüft nicht, ob Typen korrekt zugewiesen werden

TypeScript

- TypeScript [\[5\]](#) ist eine Sprache von Microsoft, die auf JavaScript aufbaut
- JavaScript überprüft nicht, ob Typen korrekt zugewiesen werden
 - TypeScript fügt statische Typisierung und Klassen hinzu

TypeScript

- TypeScript [\[5\]](#) ist eine Sprache von Microsoft, die auf JavaScript aufbaut
- JavaScript überprüft nicht, ob Typen korrekt zugewiesen werden
 - TypeScript fügt statische Typisierung und Klassen hinzu
- Fertiger Code wird zu einem ausführbaren JavaScript Programm kompiliert

TypeScript

- TypeScript [\[5\]](#) ist eine Sprache von Microsoft, die auf JavaScript aufbaut
- JavaScript überprüft nicht, ob Typen korrekt zugewiesen werden
 - TypeScript fügt statische Typisierung und Klassen hinzu
- Fertiger Code wird zu einem ausführbaren JavaScript Programm kompiliert
- Backend:

TypeScript

- TypeScript [\[5\]](#) ist eine Sprache von Microsoft, die auf JavaScript aufbaut
- JavaScript überprüft nicht, ob Typen korrekt zugewiesen werden
 - TypeScript fügt statische Typisierung und Klassen hinzu
- Fertiger Code wird zu einem ausführbaren JavaScript Programm kompiliert
- Backend:
 - Simulierte CPU
 - Parser
 - Debugger

Parser

- Erzeugen eines Parsers auf einer vordefinierten Grammatik mit tsPEG [\[3\]](#)

Parser

- Erzeugen eines Parsers auf einer vordefinierten Grammatik mit tsPEG [\[3\]](#)

```
start := instruction | data
instruction := instruction1 | instruction2

instruction1 := inst='MOV' '[ \t]+' reg='r[0-9]+' ', #' immediate='[0-9]+'
instruction2 := inst='CMP' '[ \t]+' reg1='r[0-9]+' ', ' reg2='r[0-9]+'

data := '.data\n' label='.[a-zA-Z]+' '[ \t]+' '\n' data='[a-zA-Z0-9\n]*' '\n'
```

- Beispielgrammatik, die 2 Instruktionen und einem Datenbereich erkennt

Parser

- Erzeugen eines Parsers auf einer vordefinierten Grammatik mit tsPEG [\[3\]](#)

```
start := instruction | data
instruction := instruction1 | instruction2

instruction1 := inst='MOV' '[ \t]+' reg='r[0-9]+' ', #' immediate='[0-9]+'
instruction2 := inst='CMP' '[ \t]+' reg1='r[0-9]+' ', ' reg2='r[0-9]+'

data := '.data\n' label='.[a-zA-Z]+' '[ \t]+' '\n' data='[a-zA-Z0-9\n]*' '\n'
```

- Beispielgrammatik, die 2 Instruktionen und einem Datenbereich erkennt
- Speichern der wichtigen Werte mit **inst**='MOV' oder **reg**='r[0-9]+'

Parser

- Erzeugen eines Parsers auf einer vordefinierten Grammatik mit tsPEG [\[3\]](#)

```
start := instruction | data
instruction := instruction1 | instruction2

instruction1 := inst='MOV' '[ \t]+' reg='r[0-9]+' ', #' immediate='[0-9]+'
instruction2 := inst='CMP' '[ \t]+' reg1='r[0-9]+' ', ' reg2='r[0-9]+'

data := '.data\n' label='.[a-zA-Z]+' '[ \t]+' '\n' data='[a-zA-Z0-9\n]*' '\n'
```

- Beispielgrammatik, die 2 Instruktionen und einem Datenbereich erkennt
- Speichern der wichtigen Werte mit **inst='MOV'** oder **reg='r[0-9]+'**
- Weitergabe an CPU, die Instruktionen ausführt

Debugger

- Anzeige von Registern, Stack und Teilen des Hauptspeichers

Debugger

- Anzeige von Registern, Stack und Teilen des Hauptspeichers
- Zeilenweise Abarbeitung und Setzen von Breakpoints

Debugger

- Anzeige von Registern, Stack und Teilen des Hauptspeichers
- Zeilenweise Abarbeitung und Setzen von Breakpoints
- Funktionen des Debuggers:
 - *Step Into* – Nächste Zeile + Springen in eine mögl. Subroutine

Debugger

- Anzeige von Registern, Stack und Teilen des Hauptspeichers
- Zeilenweise Abarbeitung und Setzen von Breakpoints
- Funktionen des Debuggers:
 - *Step Into* – Nächste Zeile + Springen in eine mögl. Subroutine
 - *Step Over* – Nächste Zeile + Ausführen einer mögl. Subroutine

Debugger

- Anzeige von Registern, Stack und Teilen des Hauptspeichers
- Zeilenweise Abarbeitung und Setzen von Breakpoints
- Funktionen des Debuggers:
 - *Step Into* – Nächste Zeile + Springen in eine mögl. Subroutine
 - *Step Over* – Nächste Zeile + Ausführen einer mögl. Subroutine
 - *Continue* – Ausführen bis zum nächsten Breakpoint

Debugger

- Anzeige von Registern, Stack und Teilen des Hauptspeichers
- Zeilenweise Abarbeitung und Setzen von Breakpoints
- Funktionen des Debuggers:
 - *Step Into* – Nächste Zeile + Springen in eine mögl. Subroutine
 - *Step Over* – Nächste Zeile + Ausführen einer mögl. Subroutine
 - *Continue* – Ausführen bis zum nächsten Breakpoint
 - *Step Return* – Ausführen bis zum Ende der Subroutine

Debugger

- Anzeige von Registern, Stack und Teilen des Hauptspeichers
- Zeilenweise Abarbeitung und Setzen von Breakpoints
- Funktionen des Debuggers:
 - *Step Into* – Nächste Zeile + Springen in eine mögl. Subroutine
 - *Step Over* – Nächste Zeile + Ausführen einer mögl. Subroutine
 - *Continue* – Ausführen bis zum nächsten Breakpoint
 - *Step Return* – Ausführen bis zum Ende der Subroutine
 - *Stop* – Beenden der Ausführung

React

- React [\[4\]](#) ist ein Webframework von Facebook um Benutzeroberflächen in JavaScript zu erstellen

React

- React [\[4\]](#) ist ein Webframework von Facebook um Benutzeroberflächen in JavaScript zu erstellen
- Frontend der Webanwendung

Register	Stack
r0	00000000
r1	00000000
r2	00000000
r3	00000000
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
r13	00000000
r14	00000000
r15	00000000

Debugger

Step Into
Step Over
Continue

Options
Save/Load File


```

7  _start:
8
9  LDR r0, =13          // n
10 LDR r1, =7           // k
11 BL pas              // Routine für Pascal-Loop
12
13 MOV r1, r0           // Wert nach r1 kopieren für dec Ausgabe
14 BL dec              // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0           // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0           // Vergleiche k mit 0
25 MOVEQ r0, #1         // Wenn k = 0, ist der Wert...
26 BEQ rec_end          // ...an dieser Stelle 1
27 MOVL r0, #0          // Wenn k < 0, wird der Wert...
28 BLT rec_end          // ...mit 0 initialisiert
29
30 CMP r1, r0           // Vergleiche k mit n
31 MOVEQ r0, #1         // Wenn k = n, ist der Wert...
32 BEQ rec_end          // ...an dieser Stelle 1
33 MOVT r0, #0          // Wenn k > n, wird der Wert...
34 BGT rec_end          // ...mit 0 initialisiert
35
36 CMP r0, #1           // Vergleiche n mit 1
37 ...

```


Terminal:
1716

React

- React [\[4\]](#) ist ein Webframework von Facebook um Benutzeroberflächen in JavaScript zu erstellen
- Frontend der Webanwendung
- Visualisierung der einzelnen Komponenten:

The screenshot displays a debugger window with the following components:

- Register:** A table showing 16 registers (r0-r15) with their current values, all of which are 00000000.
- Stack:** An empty section for stack memory.
- Debugger:** A panel with three buttons: "Step Into" (with a blue arrow), "Step Over" (with a blue arrow), and "Continue" (with a green play button).
- Options:** A section with a "Save/Load File" button.
- Assembly Code:** A list of instructions with line numbers 7 to 37. Red dots indicate the current instruction pointer at lines 13 and 24.


```

7  _start:
8
9  LDR r0, =13           // n
10 LDR r1, =7            // k
11 BL pas               // Routine für Pascal-Loop
12
13 MOV r1, r0            // Wert nach r1 kopieren für dec Ausgabe
14 BL dec               // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0            // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0            // Vergleiche k mit 0
25 MOVEQ r0, #1          // Wenn k = 0, ist der Wert...
26 BEQ rec_end          // ...an dieser Stelle 1
27 MOVL r0, #0           // Wenn k < 0, wird der Wert...
28 BLT rec_end          // ...mit 0 initialisiert
29
30 CMP r1, r0            // Vergleiche k mit n
31 MOVEQ r0, #1          // Wenn k = n, ist der Wert...
32 BEQ rec_end          // ...an dieser Stelle 1
33 MOVT r0, #0           // Wenn k > n, wird der Wert...
34 BGT rec_end          // ...mit 0 initialisiert
35
36 CMP r0, #1            // Vergleiche n mit 1
37 ...

```
- Terminal:** A black box showing the output "1716".

React

- React [\[4\]](#) ist ein Webframework von Facebook um Benutzeroberflächen in JavaScript zu erstellen
- Frontend der Webanwendung
- Visualisierung der einzelnen Komponenten:
 - Textfeld für Benutzereingabe und Setzen von Breakpoints

The screenshot displays a debugger window with the following components:

- Register:** A table showing 16 registers (r0-r15) with their current values, all set to 00000000.
- Stack:** An empty section for stack memory.
- Debugger:** A panel with three buttons: "Step Into" (with a blue arrow), "Step Over" (with a blue arrow), and "Continue" (with a green play button).
- Options:** A section with a "Save/Load File" button.
- Assembly Code:** A list of instructions with line numbers 7 to 37. Red dots indicate the current instruction pointer at lines 13 and 24.


```

7  _start:
8
9  LDR r0, =13           // n
10 LDR r1, =7            // k
11 BL pas               // Routine für Pascal-Loop
12
13 MOV r1, r0            // Wert nach r1 kopieren für dec Ausgabe
14 BL dec               // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0            // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0            // Vergleiche k mit 0
25 MOVEQ r0, #1          // Wenn k = 0, ist der Wert...
26 BEQ rec_end          // ...an dieser Stelle 1
27 MOVL r0, #0           // Wenn k < 0, wird der Wert...
28 BLT rec_end          // ...mit 0 initialisiert
29
30 CMP r1, r0            // Vergleiche k mit n
31 MOVEQ r0, #1          // Wenn k = n, ist der Wert...
32 BEQ rec_end          // ...an dieser Stelle 1
33 MOVT r0, #0           // Wenn k > n, wird der Wert...
34 BGT rec_end          // ...mit 0 initialisiert
35
36 CMP r0, #1            // Vergleiche n mit 1
37 ...

```
- Terminal:** A black box showing the output "1716".

React

- React [\[4\]](#) ist ein Webframework von Facebook um Benutzeroberflächen in JavaScript zu erstellen
- Frontend der Webanwendung
- Visualisierung der einzelnen Komponenten:
 - Textfeld für Benutzereingabe und Setzen von Breakpoints
 - Terminal für Ausgabe von Ergebnissen und Fehlern/Warnungen

The screenshot displays a debugger window with the following components:

- Register:** A table showing 16 registers (r0 to r15) with their current values, all of which are 00000000.
- Stack:** An empty section for viewing the stack.
- Debugger:** A panel with three buttons: "Step Into" (with a blue arrow), "Step Over" (with a blue arrow), and "Continue" (with a green play button).
- Options:** A section with a "Save/Load File" button.
- Assembly Code:** A list of instructions with line numbers 7 to 37. Red dots indicate the current instruction pointer at line 13 and line 34.


```

7  _start:
8
9  LDR r0, =13           // n
10 LDR r1, =7            // k
11 BL pas               // Routine für Pascal-Loop
12
13 MOV r1, r0            // Wert nach r1 kopieren für dec Ausgabe
14 BL dec               // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0            // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0            // Vergleiche k mit 0
25 MOVEQ r0, #1          // Wenn k = 0, ist der Wert...
26 BEQ rec_end          // ...an dieser Stelle 1
27 MOVL r0, #0           // Wenn k < 0, wird der Wert...
28 BLT rec_end          // ...mit 0 initialisiert
29
30 CMP r1, r0            // Vergleiche k mit n
31 MOVEQ r0, #1          // Wenn k = n, ist der Wert...
32 BEQ rec_end          // ...an dieser Stelle 1
33 MOVG r0, #0           // Wenn k > n, wird der Wert...
34 BGT rec_end          // ...mit 0 initialisiert
35
36 CMP r0, #1            // Vergleiche n mit 1
37 ...

```
- Terminal:** A black box with a red border showing the output "1716".

React

- React [\[4\]](#) ist ein Webframework von Facebook um Benutzeroberflächen in JavaScript zu erstellen
- Frontend der Webanwendung
- Visualisierung der einzelnen Komponenten:
 - Textfeld für Benutzereingabe und Setzen von Breakpoints
 - Terminal für Ausgabe von Ergebnissen und Fehlern/Warnungen
 - Zustand des Programms, wie Inhalt der Register und des Stack

The screenshot displays a debugger window with three main sections:

- Register and Stack:** A table showing 16 registers (r0-r15) and a stack. The registers are currently empty (00000000). The stack is also empty.
- Assembly Code:** A list of instructions with line numbers 7 to 37. Red dots indicate breakpoints at lines 13, 24, and 34.


```

7  _start:
8
9  LDR r0, =13           // n
10 LDR r1, =7            // k
11 BL pas               // Routine für Pascal-Loop
12
13 MOV r1, r0            // Wert nach r1 kopieren für dec Ausgabe
14 BL dec               // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0            // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0            // Vergleiche k mit 0
25 MOVEQ r0, #1          // Wenn k = 0, ist der Wert...
26 BEQ rec_end          // ...an dieser Stelle 1
27 MOVL r0, #0           // Wenn k < 0, wird der Wert...
28 BLT rec_end          // ...mit 0 initialisiert
29
30 CMP r1, r0            // Vergleiche k mit n
31 MOVEQ r0, #1          // Wenn k = n, ist der Wert...
32 BEQ rec_end          // ...an dieser Stelle 1
33 MOVGT r0, #0          // Wenn k > n, wird der Wert...
34 BGT rec_end          // ...mit 0 initialisiert
35
36 CMP r0, #1            // Vergleiche n mit 1
37 ...
      
```
- Debugger Controls:** Buttons for "Step Into", "Step Over", and "Continue".
- Options:** A button for "Save/Load File".
- Terminal:** A black box showing the output "1716".

React

- Visualisierung der einzelnen Komponenten:
 - Textfeld für Benutzereingabe und Setzen von Breakpoints
 - Terminal für Ausgabe von Ergebnissen und Fehlern/Warnungen
 - Zustand des Programms, wie Inhalt der Register und des Stack

The screenshot displays a debugger interface with the following components:

- Register Window:** Shows 16 registers (r0-r15) with their current values, all set to 00000000.
- Stack Window:** Currently empty.
- Debugger Controls:** Includes buttons for 'Step Into', 'Step Over', and 'Continue'.
- Options:** A button labeled 'Save/Load File'.
- Assembly Code Window:** Displays ARM assembly code with line numbers 7 to 37. Red dots indicate breakpoints at lines 13, 24, and 34.


```

7  _start:
8
9  LDR r0, =13           // n
10 LDR r1, =7            // k
11 BL pas               // Routine für Pascal-Loop
12
13 MOV r1, r0            // Wert nach r1 kopieren für dec Ausgabe
14 BL dec               // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0           // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0           // Vergleiche k mit 0
25 MOVEQ r0, #1         // Wenn k = 0, ist der Wert...
26 BEQ rec_end          // ...an dieser Stelle 1
27 MOVL r0, #0          // Wenn k < 0, wird der Wert...
28 BLT rec_end          // ...mit 0 initialisiert
29
30 CMP r1, r0           // Vergleiche k mit n
31 MOVEQ r0, #1         // Wenn k = n, ist der Wert...
32 BEQ rec_end          // ...an dieser Stelle 1
33 MOVT r0, #0          // Wenn k > n, wird der Wert...
34 BGT rec_end          // ...mit 0 initialisiert
35
36 CMP r0, #1           // Vergleiche n mit 1
37 ...
      
```
- Terminal Window:** Shows the output '1716'.

React

- Visualisierung der einzelnen Komponenten:
 - Textfeld für Benutzereingabe und Setzen von Breakpoints
 - Terminal für Ausgabe von Ergebnissen und Fehlern/Warnungen
 - Zustand des Programms, wie Inhalt der Register und des Stack
 - Funktionen des Debuggers

The screenshot displays a debugger interface with the following components:

- Register Window:** Shows 16 registers (r0-r15) with their current values, all set to 00000000.
- Stack Window:** Currently empty.
- Assembly Code Window:** Displays assembly instructions with line numbers 7 to 37. Red dots indicate breakpoints at lines 13, 24, and 34.


```

7  _start:
8
9  LDR r0, =13           // n
10 LDR r1, =7            // k
11 BL pas               // Routine für Pascal-Loop
12
13 MOV r1, r0            // Wert nach r1 kopieren für dec Ausgabe
14 BL dec               // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0            // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0            // Vergleiche k mit 0
25 MOVEQ r0, #1          // Wenn k = 0, ist der Wert...
26 BEQ rec_end           // ...an dieser Stelle 1
27 MOVL r0, #0           // Wenn k < 0, wird der Wert...
28 BLT rec_end           // ...mit 0 initialisiert
29
30 CMP r1, r0            // Vergleiche k mit n
31 MOVEQ r0, #1          // Wenn k = n, ist der Wert...
32 BEQ rec_end           // ...an dieser Stelle 1
33 MOVGT r0, #0          // Wenn k > n, wird der Wert...
34 BGT rec_end           // ...mit 0 initialisiert
35
36 CMP r0, #1            // Vergleiche n mit 1
37 ...
      
```
- Debugger Controls:** A red box highlights the 'Step Into', 'Step Over', and 'Continue' buttons.
- Options:** A button labeled 'Options Save/Load File' is located below the debugger controls.
- Terminal:** A black terminal window at the bottom right shows the output '1716'.

React

- Visualisierung der einzelnen Komponenten:
 - Textfeld für Benutzereingabe und Setzen von Breakpoints
 - Terminal für Ausgabe von Ergebnissen und Fehlern/Warnungen
 - Zustand des Programms, wie Inhalt der Register und des Stack
 - Funktionen des Debuggers
 - Weitere Optionen

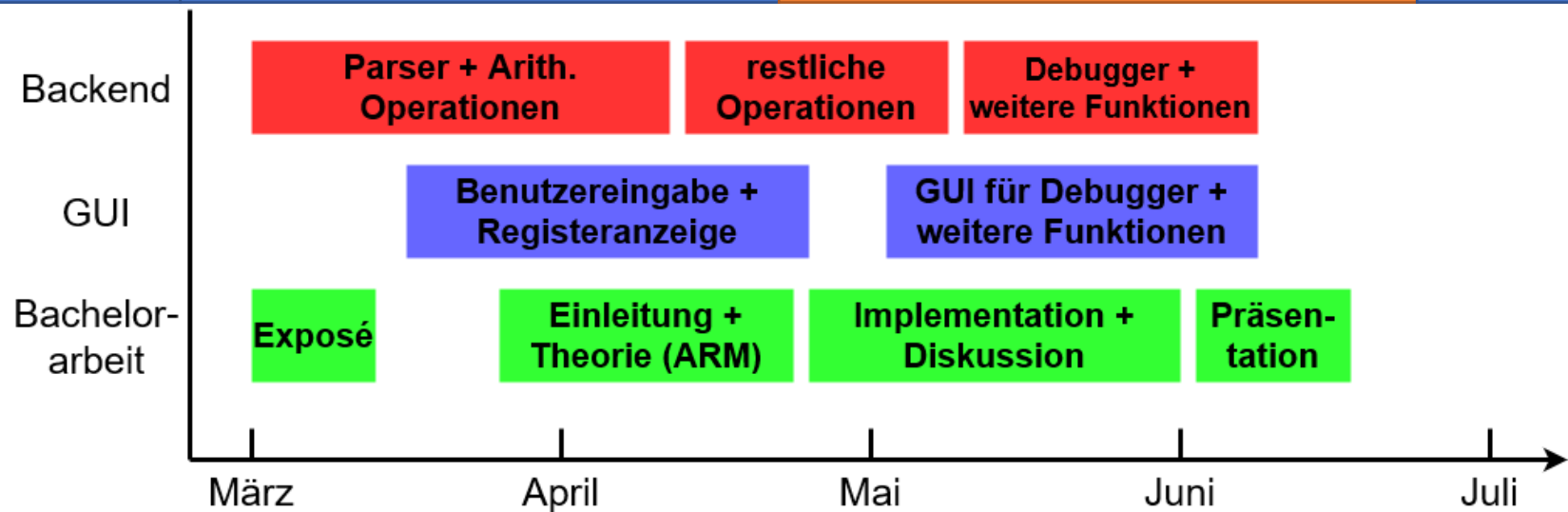
The screenshot displays a debugger window with the following components:

- Register:** A table showing 16 registers (r0 to r15) with their current values, all set to 00000000.
- Stack:** A section for viewing the stack memory.
- Debugger:** A panel containing three buttons: "Step Into" (with a blue arrow), "Step Over" (with a blue arrow), and "Continue" (with a green play button).
- Options:** A section at the bottom with a red border containing the text "Options" and "Save/Load File".
- Assembly Code:** A list of instructions with line numbers 7 to 37. Red dots indicate the current instruction pointer at lines 13 and 34.

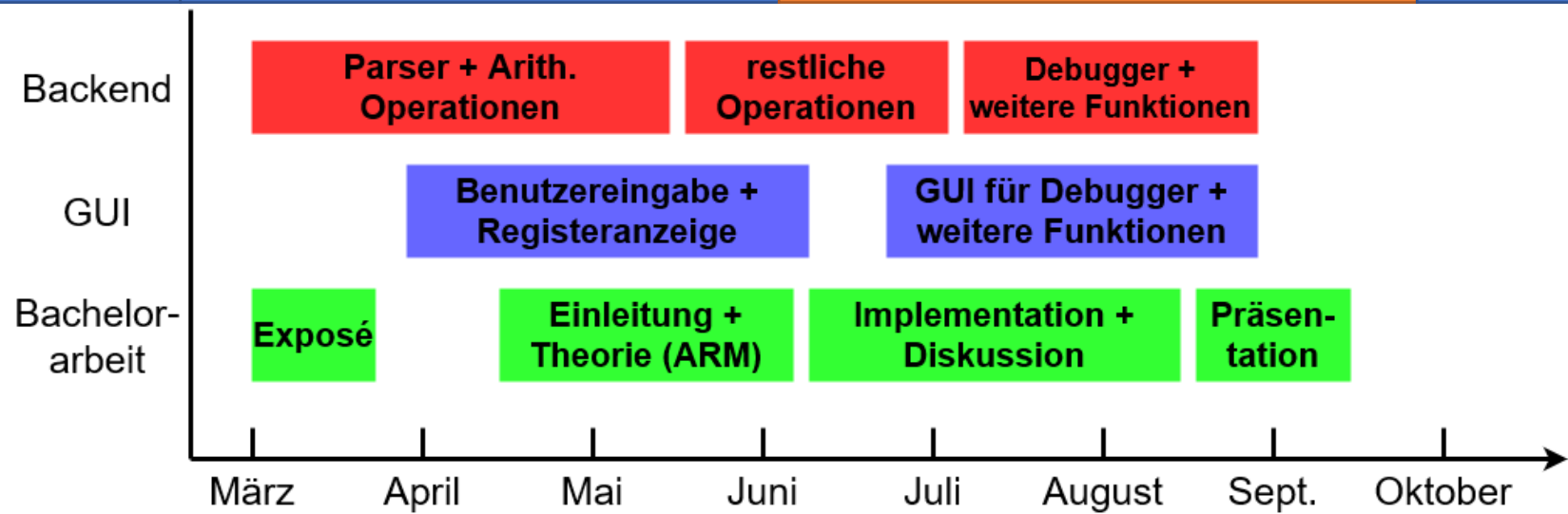

```

7  _start:
8
9  LDR r0, =13           // n
10 LDR r1, =7            // k
11 BL pas               // Routine für Pascal-Loop
12
13 MOV r1, r0            // Wert nach r1 kopieren für dec Ausgabe
14 BL dec               // Dezimal Ausgabe von vorigem Blatt
15
16 MOV r0, #0            // exit syscall
17 MOV r7, #1
18 SWI #0
19
20
21 pas:
22 STMFD sp!, {r2-r12, lr} // Register sichern
23
24 CMP r1, #0            // Vergleiche k mit 0
25 MOVEQ r0, #1          // Wenn k = 0, ist der Wert...
26 BEQ rec_end           // ...an dieser Stelle 1
27 MOVL r0, #0           // Wenn k < 0, wird der Wert...
28 BLT rec_end           // ...mit 0 initialisiert
29
30 CMP r1, r0            // Vergleiche k mit n
31 MOVEQ r0, #1          // Wenn k = n, ist der Wert...
32 BEQ rec_end           // ...an dieser Stelle 1
33 MOVG r0, #0           // Wenn k > n, wird der Wert...
34 BGT rec_end           // ...mit 0 initialisiert
35
36 CMP r0, #1            // Vergleiche n mit 1
37 ...

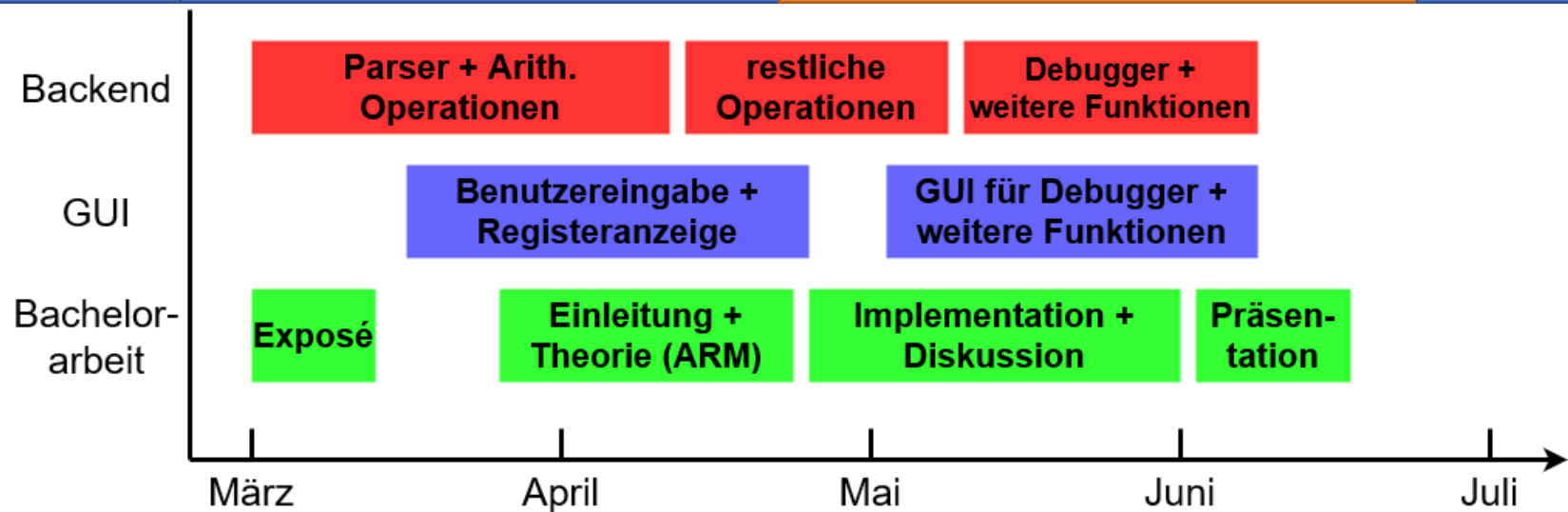
```
- Terminal:** A black box at the bottom right showing the output "1716".



- Gesamte Zeit für Bachelorarbeit zur Verfügung – Zeitplan für Präsentation dieses Semester

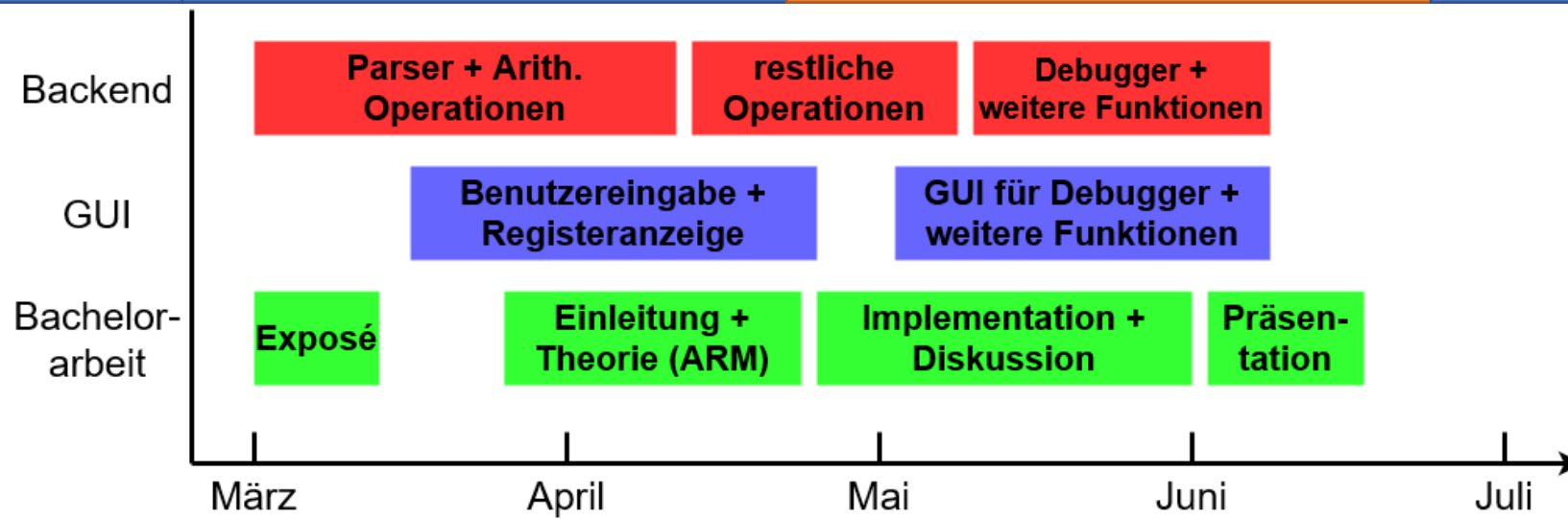


- Gesamte Zeit für Bachelorarbeit zur Verfügung – Zeitplan für Präsentation dieses Semester
 - Falls trotzdem zu viel Arbeit – Präsentation Anfang des nächsten Semesters



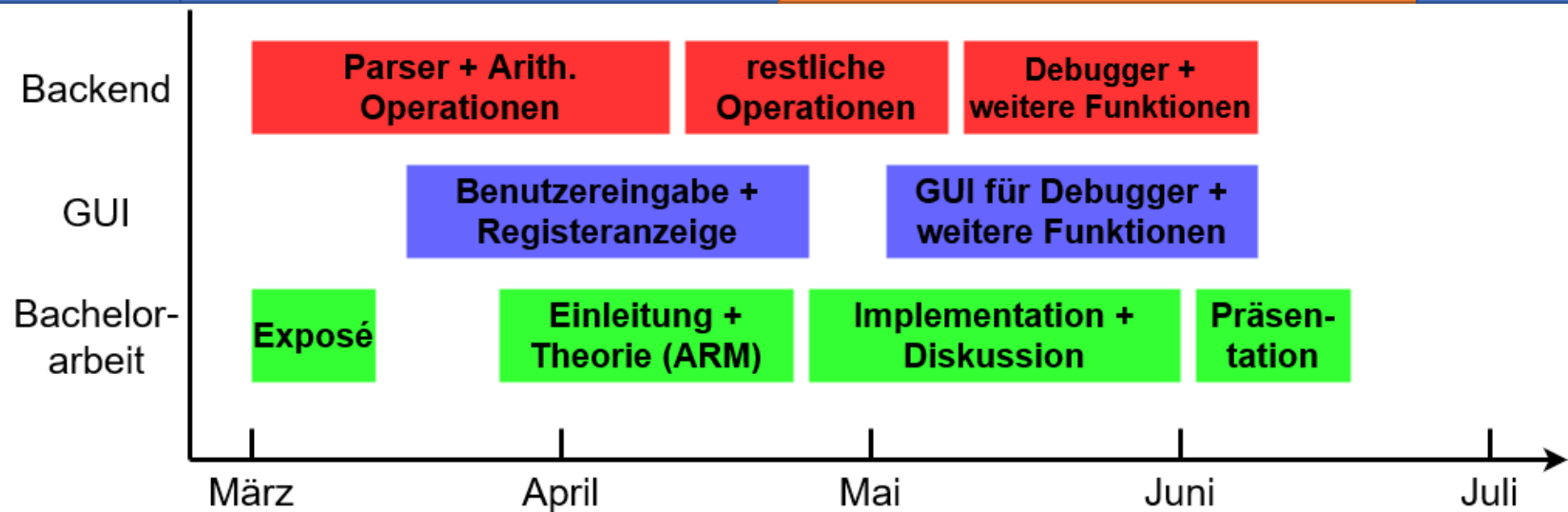
- Gesamte Zeit für Bachelorarbeit zur Verfügung – Zeitplan für Präsentation dieses Semester
 - Falls trotzdem zu viel Arbeit – Präsentation Anfang des nächsten Semesters

1. Beginn mit Parser und arithmetischen Operation + Visualisierung zum Testen



- Gesamte Zeit für Bachelorarbeit zur Verfügung – Zeitplan für Präsentation dieses Semester
 - Falls trotzdem zu viel Arbeit – Präsentation Anfang des nächsten Semesters

1. Beginn mit Parser und arithmetischen Operation + Visualisierung zum Testen
2. Restliche ARmv5 Instruktionen und Beginn Theorie der Bachelorarbeit



- Gesamte Zeit für Bachelorarbeit zur Verfügung – Zeitplan für Präsentation dieses Semester
 - Falls trotzdem zu viel Arbeit – Präsentation Anfang des nächsten Semesters
1. Beginn mit Parser und arithmetischen Operation + Visualisierung zum Testen
 2. Restliche ARmv5 Instruktionen und Beginn Theorie der Bachelorarbeit
 3. Debugger und weitere Funktionen (Speicher/Laden von Dateien, ...)

Voraussetzungen:

- Die in der Vorlesung vorgestellten bzw. für das Proseminar benötigten ARMv5-Instruktionen sind implementiert.

Voraussetzungen:

- Die in der Vorlesung vorgestellten bzw. für das Proseminar benötigten ARMv5-Instruktionen sind implementiert.
- Die Webanwendung weist eine Benutzeroberfläche (ähnlich [Folie 10](#)) mit Anzeige von Registern, Stack und Teilen des Hauptspeichers auf.

Voraussetzungen:

- Die in der Vorlesung vorgestellten bzw. für das Proseminar benötigten ARMv5-Instruktionen sind implementiert.
- Die Webanwendung weist eine Benutzeroberfläche (ähnlich [Folie 10](#)) mit Anzeige von Registern, Stack und Teilen des Hauptspeichers auf.
- Der Debugger implementiert die auf [Folie 8](#) beschriebenen Funktionen.

Voraussetzungen:

- Die in der Vorlesung vorgestellten bzw. für das Proseminar benötigten ARMv5-Instruktionen sind implementiert.
- Die Webanwendung weist eine Benutzeroberfläche (ähnlich [Folie 10](#)) mit Anzeige von Registern, Stack und Teilen des Hauptspeichers auf.
- Der Debugger implementiert die auf [Folie 8](#) beschriebenen Funktionen.
- Die korrekte Funktionsweise wird mit den Musterlösungen der Beispiele aus dem Proseminar getestet.

Referenzen

- [1] ARM Limited. GNU Toolchain for Arm processors. Zugegriffen am: 04.03.2021. <https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain>.
- [2] ARM Limited. ARMv5 Architecture Reference Manual - Issue I, 2005.
- [3] E. Davey. tsPEG: A PEG Parser Generator for TypeScript. Zugegriffen am: 04.03.2021. <https://github.com/EoinDavey/tsPEG>.
- [4] Facebook. React. Zugegriffen am: 04.03.2021. <https://reactjs.org/>.
- [5] Microsoft. Typescript. Zugegriffen am: 04.03.2021. <https://www.typescriptlang.org/>.
- [6] Microsoft. Windows Subsystem for Linux. Zugegriffen am: 04.03.2021. <https://docs.microsoft.com/en-us/windows/wsl/install-win10>.
- [7] J. Mossberg. Use GDB on an ARM assembly program. Zugegriffen am: 04.03.2021. <https://jacobmossberg.se/posts/2017/01/17/use-gdb-on-arm-assembly-program.html>
- [8] The GNU Project. GDB: The GNU Project Debugger. Zugegriffen am: 04.03.2021. <https://www.gnu.org/software/gdb/>.
- [9] The QEMU Project Developers. QEMU User Mode Emulation. Zugegriffen am: 04.03.2021. <https://qemu.readthedocs.io/en/latest/user/index.html>.

