universität innsbruck

Exposé: ARM Simulator, Interpreter und Debugger als Webanwendung

Leopold-Franzens-Universität Innsbruck
Institut für Informatik
Security and Privacy Lab

Dominik Zangerl
Betreuer: Alexander Schlögl

Innsbruck, 19. März 2021

1 Motivation

Das Ziel meiner Bachelorarbeit ist es eine Webanwendung zu entwickeln, mit der die ARMv5 Entwicklungsumgebung simuliert wird. ARMv5 [2] wird im ersten Semester als Beispiel für eine Befehlssatzarchitektur unterrichtet. Studierende sollen ihre eigenen Programme in Assembler [8] schreiben und diese dann auf einer ARMv5 Architektur ausführen. Diese Entwicklungsumgebung wird zurzeit mit verschiedenen Linux-Programmen simuliert. Die GNU Toolchain für die ARM Cortex-A Architektur [1] wird für das Kompilieren und Linken der Assembler-Dateien verwendet. Das kompilierte ARM-Programm läuft dann nicht auf der Architektur des Hostrechners, sondern wird mit Hilfe des QEMU User-Space-Emulators [13] ausgeführt. Dieser Prozess kann vereinfacht werden, indem man die Toolchain in einer virtuellen Maschine oder dem Windows Subsystem for Linux [11] installiert und sich für die Befehlskette ein Skript schreibt.

Das größere Problem bei dieser Toolchain ist die Fehlersuche und das Debugging des Programms. Den Fehler auf eine bestimmte Instruktion oder ein Registers zurückzuführen nimmt oft die größte Zeit in Anspruch. Der ARM-Emulator kann zusammen mit dem GNU Debugger [12] verwendet werden, welcher auch die Inhalte der Register anzeigen kann. Dies bedeutet jedoch häufig einen großen Zeitaufwand um alles aufzusetzen. Auch das Arbeiten mit Debuggern, besonders auf der Kommandozeile, könnte vielen noch nicht geläufig sein.

An dieser Stelle greift dieses Bachelorprojekt ein und versucht die ARMv5 Entwicklungsumgebung inklusive Debugging mit einer Webanwendung zu simulieren. Benutzer:innen schreiben den ARM-Code direkt in die Webanwendung, welcher dann auf einer simulierten CPU und simuliertem Hauptspeicher direkt im Browser ausgeführt wird. Die Inhalte der Register, des Stacks und Teile des Hauptspeichers werden dauerhaft angezeigt und helfen Benutzer:innen bei der Fehlerbehebung, da sie sofort sehen, an welcher Stelle ein ungewünschter Wert in ein Register geschrieben wird. Zusammen mit den Funktionen eines Debuggers, wie zeilenweise Abarbeitung des Codes oder setzen von Breakpoints, wird den Studierenden die zeitaufwändigste Arbeit abgenommen und sie können sich auf den wichtigen Teil konzentrieren, nämlich das Schreiben und Verstehen von ARM-Assembler Code.

2 Technologien und Implementation

2.1 Technologien

Das Backend der Anwendung wird in TypeScript geschrieben. TypeScript [3][10] ist eine Programmiersprache, die auf JavaScript aufbaut und statische Typisierung und Klassen hinzufügt. Sie wurde von Microsoft entwickelt um die Schwächen von JavaScript zu umgehen. Die fehlende Typisierung in JavaScript kann bei größeren Programmen leichter zu Fehlern führen. Für die Erstellung der Webanwendung wird React verwendet. React [5] ist ein Webframework von Facebook zur Erstellung von Benutzeroberflächen für JavaScript/TypeScript und funktioniert in jedem modernen Browser. Dies garantiert eine einheitliche und plattformunabhängige Darstellung der TypeScript-Applikation.

2.2 Parser

Zuerst benötigen wir einen Parser, der den ARM-Code der Benutzer:innen aus der Webanwendung einliest und interpretiert. Dafür verwende ich den Parser-Generator tsPEG [4] für TypeScript. Mit tsPEG kann eine Parsing Expression Grammatik (PEG) [6] definiert werden, die alle nötigen Instruktionen und Deklarationen von ARMv5 enthält, und daraus einen Parser generieren.

Listing 1: Beispielgrammatik mit tsPEG für die `MOV`-Instruktion mit 2 Registern und Barrel-Shifter oder Daten eines Speicherbereichs mit Label.

```
start := inst | data

inst := inst='MOV' '[ \t]+' r1='r[0-9]+' ', ' r2='r[0-9]+' shift=barrel?
barrel := ', ' shift_type=shift_type ' #' shift_amount='[0-9]+'
shift_type := 'LSL' | 'LSR' | 'ASR' | 'ROR' | 'RRX' | 'ASL'

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

Listing 1 zeigt ein Beispiel einer solchen Grammatik. Diese Beispielgrammatik kann eine `MOV`-Instruktion mit zwei Register und Barrel-Shifter oder das Label und die Daten eines Datenbereichs erkennen. Mit den Zuweisungen innerhalb einer Zeile, wie „`r1=`“ oder „`data=`“, werden die geparsten Werte als Variablen in einem abstrakten Syntaxbaum (AST) gespeichert. Dabei muss auch beachtet werden, dass Operationen eine unterschiedliche Anzahl von Parametern aufweisen können. In diesem Beispiel kennzeichnet das „`?`“ am Ende der dritten Zeile den optionalen Barrel-Shifter. Der Parser sollte dabei die richtige Syntax kontrollieren, die Datenbereiche und Labels einlesen und anschließend die Instruktionen in der richtigen Reihenfolge parsen. Diese können dann an die simulierte CPU weitergegeben werden, welche die Instruktionen ausführt.

2.3 Simulator und Debugger

Die CPU und der Hauptspeicher werden ebenfalls mit TypeScript simuliert. Die CPU liest die einzelnen Instruktionen des Parsers und führt diese dann aus, indem es die Register verändert, vom Hauptspeicher liest bzw. in den Hauptspeicher schreibt oder etwas auf dem Terminal ausgibt. Die CPU simuliert auch die Standardfunktionen eines Debuggers. Dazu zählen das Setzen von Breakpoints, das zeilenweise Ausführen der Instruktionen, Ausführung bis zum nächsten Breakpoint oder das Ausführen einer einzelnen Subroutine. Die wichtigsten Funktionen sind in Abbildung 1 angeführt. Mit *Step Into* wird die nächste Zeile ausgeführt und einer möglichen Subroutine gefolgt. Mit *Step Over* wird, statt der Subroutine zu folgen, diese

ausgeführt und das Programm danach wieder gestoppt. *Continue* führt das Programm bis zum nächsten Breakpoint aus. Weitere Funktionen sind *Pause/Stop*, um das Programm zu pausieren bzw. im Falle einer Schleife zu beenden oder *Step Return*, um das Programm bis zum Ende der Subroutine auszuführen. Zuletzt beinhaltet der Simulator noch den Zustand der Register, des Stacks und des Hauptspeichers. Diese werden nach jedem Instruktionsschritt aktualisiert und Benutzer:innen angezeigt.

2.4 Frontend

In Abbildung 1 ist das geplante Layout der Webanwendung dargestellt, die sich an ähnlichen Online Compilern und Debuggern orientiert (Online GDB [7], CPUlator [14], emuARM [9]).

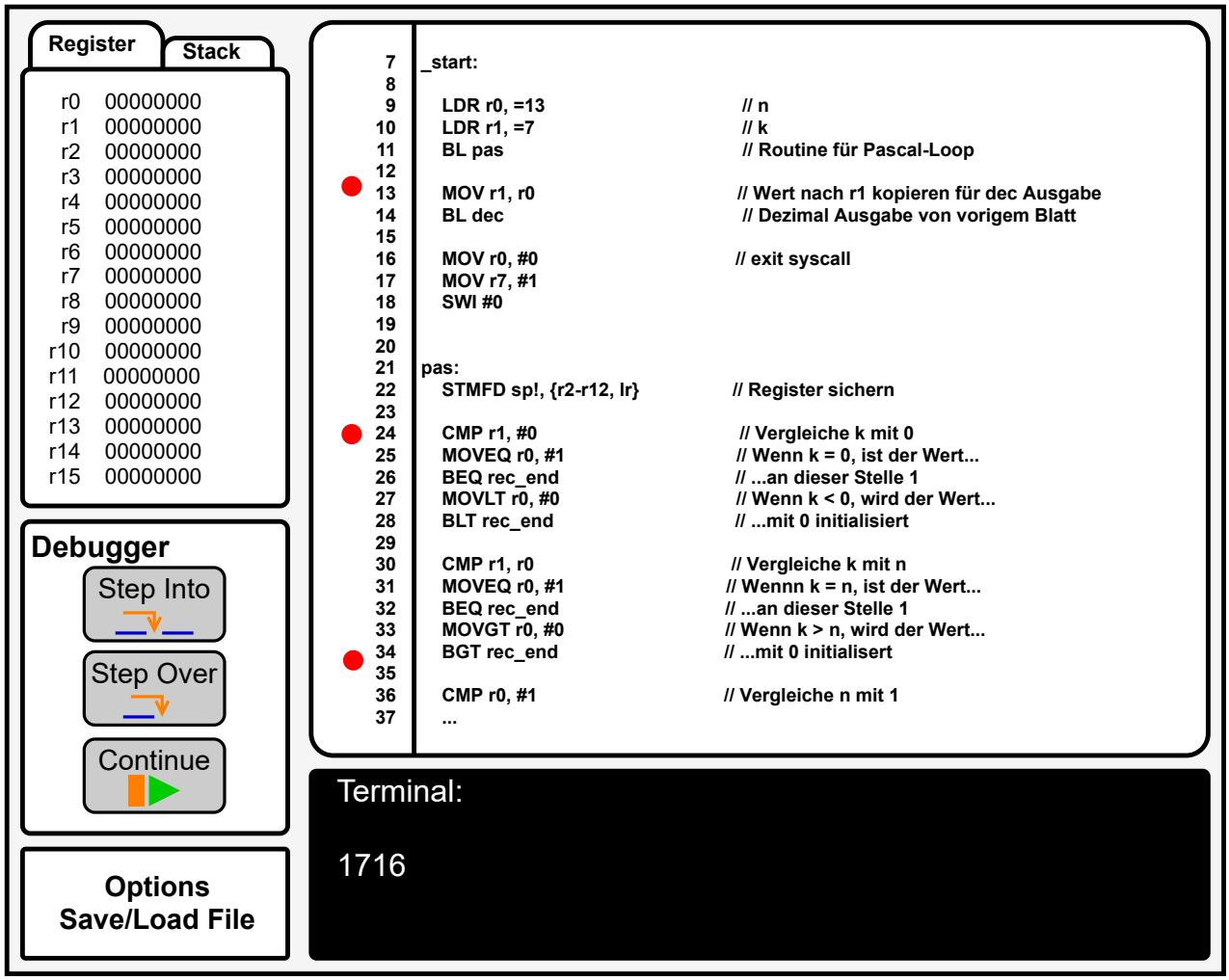

Abbildung 1: Geplantes Layout der Webanwendung mit Eingabefeld, Terminal, Ansicht für Register und Stack, Debugger und Optionsfeld. Code-Ausschnitt aus meiner Lösung für das Pascal-Dreieck.

Auf der rechten Seite befindet sich ein großes Textfeld für die Eingabe des ARM-Codes der Benutzer:innen. Bei den einzelnen Zeilennummern können Breakpoints für den Debugger gesetzt werden. Darunter befindet sich das Terminal für die Ausgabe eines Ergebnisses oder etwaigen Fehlern/Warnungen. Auf der linken Seite wird der jetzige Zustand des Programms ausgegeben, wie der Inhalt der einzelnen Register oder der Stack Trace. Der Inhalt der Register

sollte an dieser Stelle auch durch die Benutzer:innen verändert werden können. Darunter befinden sich die Funktionen des Debuggers und etwaige andere Optionen, wie das Speichern und Laden von Dateien.

3 Vorgehensweise und Zeitplan

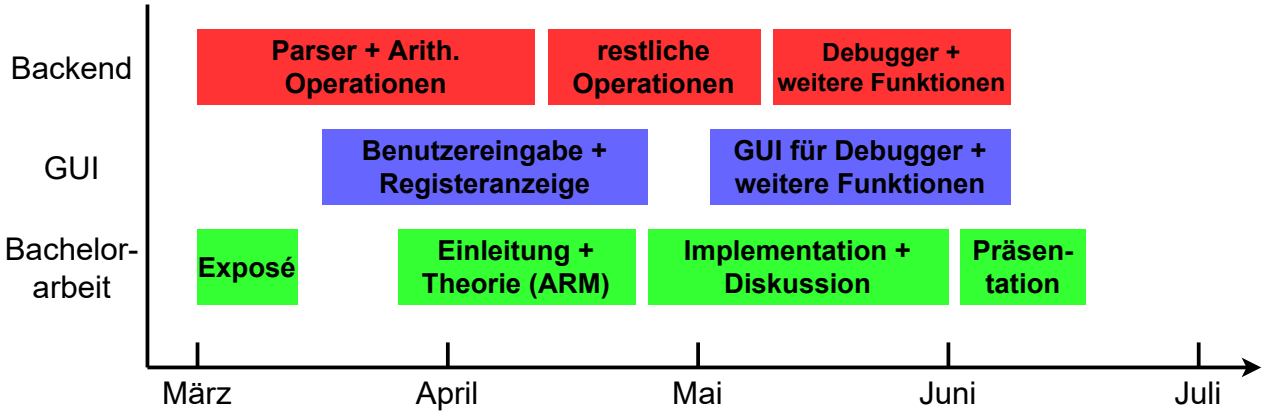

Abbildung 2: Voraussichtliche zeitliche Planung der einzelnen Teile des Bachelorprojekts aufgeteilt in Backend, Gestalten der Benutzeroberfläche und Schreiben der Bachelorarbeit für Präsentation in diesem Semester. Andernfalls Hinzunahme der Sommermonate und Präsentation Anfang des nächsten Semesters.

Der Zeitplan für mein Bachelorprojekt ist in Abbildung 2 abgebildet. Da ich keine anderen Lehrveranstaltungen mehr habe und meine volle Zeit auf die Bachelorarbeit konzentrieren kann, habe ich meinen Zeitplan auf eine Präsentation in diesem Semester ausgelegt. Falls in den ersten Monaten klar wird, dass dieser Zeitplan zu eng gefasst ist und ich für eine Präsentation in diesem Semester nicht fertig werde, nehme ich die Sommermonate zur Implementation und Fertigstellung hinzu und plane die Präsentation für einen der ersten Termine des Bachelorseminars des nächsten Semesters.

Für die finale Implementierung müssen folgende Voraussetzungen erfüllt sein:

- Die in der Vorlesung vorgestellten bzw. für das Proseminar benötigten ARMv5-Instruktionen sind implementiert.
- Die Webanwendung weist eine Benutzeroberfläche (ähnlich Abbildung 1) mit Anzeige von Registern, Stack und Teilen des Hauptspeichers auf.
- Der Debugger implementiert die in Abschnitt 2.3 beschriebenen Funktionen.
- Die korrekte Funktionsweise wird mit den Musterlösungen der Beispiele aus dem Proseminar getestet.

Optionale Ziele meiner Bachelorarbeit sind:

- Das Erstellen von Vorlagen für die PS-Aufgaben (bereitgestellte Skeletons, Code aus der Vorlesung) und Überprüfung der Korrektheit des Ergebnisses im Hintergrund (z.B. korrekte Umwandlung in Dezimalzahl, richtige Zahlen im Pascalschen Dreieck).

- Die Implementierung einer automatischen Code-Vervollständigung mit Hinweisen zur Verwendung der eingetippten Instruktionen (Anzahl Parameter, optionale Parameter).

Der Umfang meiner Bachelorarbeit beinhaltet nicht:

- Die Implementierung aller Funktionen einer ARMv5-Entwicklungsumgebung (z.B. Thumb-Instruktionen, alle Exceptions).
- Die Simulation findet auf Ebene des Assembler-Codes statt. Es gibt keine Unterstützung für die Anzeige des Maschinencodes nach der Kompilation, wie bei CPUlator [14].

Literatur

[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] G. Bierman, M. Abadi, and M. Torgersen. Understanding TypeScript. In *ECOOP 2014 – Object-Oriented Programming*, pages 257–281, 2014.

[4] E. Davey. tsPEG: A PEG Parser Generator for TypeScript. Zugegriffen am: 04.03.2021. `https://github.com/EoinDavey/tsPEG`.

[5] Facebook. React. Zugegriffen am: 04.03.2021. `https://reactjs.org/`.

[6] B. Ford. Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. *SIGPLAN Not.*, 39(1):111–122, January 2004.

[7] GDB Online. OnlineGDB - Online Compiler and Debugger for C/C++. Zugegriffen am: 04.03.2021. `https://www.onlinegdb.com/`.

[8] P. Knaggs. ARM Assembly Language Programming, 2016.

[9] G. Malhotra, N. Atri, and S. R. Sarangi. emuARM: A tool for teaching the ARM assembly language. In *2013 Second International Conference on E-Learning and E-Technologies in Education (ICEEE)*, pages 115–120, 2013.

[10] Microsoft. TypeScript. Zugegriffen am: 04.03.2021. `https://www.typescriptlang.org/`.

[11] Microsoft. Windows Subsystem for Linux. Zugegriffen am: 04.03.2021. `https://docs.microsoft.com/en-us/windows/wsl/install-win10`.

[12] The GNU Project. GDB: The GNU Project Debugger. Zugegriffen am: 04.03.2021. `https://www.gnu.org/software/gdb/`.

[13] The QEMU Project Developers. QEMU User Mode Emulation. Zugegriffen am: 04.03.2021. `https://qemu.readthedocs.io/en/latest/user/index.html`.

[14] H. Wong. CPUlator: A CPU and I/O device simulator. Zugegriffen am: 04.03.2021. `https://cpulator.01xz.net/?sys=arm`.