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, 5. März 2021

1 Einleitung

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 selber Programme in Assembler 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 Familien Architektur [1] wird für das Kompilieren und Linken der Assembler-Datei verwendet. Dieses Programm läuft dann nicht auf der Architektur des Hostrechners, sondern wird mit Hilfe des QEMU User-Space-Emulators [9] ausgeführt. Um also ein einfaches Assembler-Programm kompilieren und ausführen zu können, braucht ein Student eine Linux-Umgebung mit oben erwähnten Tools. Diese Toolchain kann man leicht mit einer virtuellen Maschine oder dem Windows Subsystem for Linux [7] unter Windows 10 zum Laufen bringen. Auch die verschiedenen Befehle bis zu einem ausführbaren Programm können mit einem Skript zu einem Befehl vereinfacht werden.

Das größere Problem bei dieser Toolchain ist die Fehlersuche und Debugging des Programms. Den Fehler auf eine bestimmte Instruktion oder ein Registers zurückzuführen nimmt oft die größte Zeit in Anspruch. Der ARM-Emulator kann mit dem GNU Debugger [8] zusammen verwendet werden, welcher auch die Inhalte der Register anzeigen kann. Dies bedeutet jedoch häufig, zusammen mit der Toolchain und der Assemblersprache, für Studierende des ersten Semester sehr viel Aufwand 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. Dabei sollen Benutzer ihren ARM-Code direkt in die Anwendung schreiben können, welcher dann auf einer simulierten CPU und simuliertem Hauptspeicher ausgeführt werden. Dies ersetzt die GNU Toolchain und man benötigt auch keine Linux-Umgebung, da alles direkt im Browser ausgeführt werden kann. Zusätzlich sollen dauerhaft die Inhalte der Register, des Stacks und Teile des Hauptspeichers angezeigt werden. Dies hilft dem Benutzer bei der Fehlerbehebung und er sieht sofort, falls in einem Register ein ungewünschter oder falscher Wert steht. Zusammen mit den Funktionen eines Debuggers, wie zeilenweise Abarbeitung des Codes oder setzen von Breakpoints,

vereinfacht dies die Fehlersuche und Behebung von Fehlern.

Wie oben erwähnt, beschränkt sich die Webanwendung auf die Simulation von Instruktionen der ARM Version 5, da diese im ersten Semester unterrichtet wird. Es werden von allen Instruktionen auch nur jene implementiert, die auch in der Vorlesung bzw. dem Proseminar Rechnerarchitektur verwendet werden. Dabei handelt es sich um die gebräuchlichsten arithmetischen, logischen und Vergleichsoperationen inklusive Bedingungen, sowie die wichtigsten Kopier-, Verschiebe- und Sprungoperationen und Operationen für den Speicherzugriff. Die schlussendlich implementieren Operationen werden dann im finalen Paper aufgelistet und kurz erklärt.

2 Technologien und Implementation

2.1 TypeScript

Das Backend der Anwendung wird in TypeScript geschrieben. TypeScript [6] 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. Bei JavaScript gibt es zwar auch Typen, wie Nummer oder String, aber es wird nicht überprüft, ob diese korrekt zugewiesen werden. Diese fehlende Typisierung kann bei größeren Programmen leichter zu Fehlern führen. Der fertige TypeScript Code wird dann in ein ausführbares JavaScript Programm kompiliert.

Zuerst benötigen wir einen Parser, der den ARM-Code des Benutzers aus der Webanwendung einliest und interpretiert. Für TypeScript verwende ich dafür voraussichtlich tsPEG [3], einen Parser-Generator speziell für diese Sprache. Damit definiere ich eine Grammatik und erzeuge einen Parser, der alle nötigen Instruktionen und Deklarationen von ARMv5 einlesen kann. Diese können dann an die simulierte CPU weitergegeben werden, welche die Instruktionen ausführt.

Die CPU und der Hauptspeicher werden ebenfalls mit TypeScript emuliert. 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 implementiert auch die Standardfunktionen eines Debuggers. Dazu zählen das Setzen von Breakpoints, das zeilenweise Ausführen der Instruktionen, Ausführung bis zum nächsten Breakpoint oder das Überspringen von Subroutinen. Zuletzt beinhaltet der Simulator noch den Zustand der Register, des Stacks und den nötigen Inhalt des Hauptspeichers. Diese werden nach jedem Instruktionsschritt aktualisiert und dem Benutzer angezeigt.

2.2 React

Für die Erstellung der Webanwendung wird React verwendet. React [4] ist ein Webframework von Facebook zur Erstellung von Benutzeroberflächen für JavaScript und TypeScript Applikationen. Damit lassen sich leicht die einzelnen Komponenten der ARMv5 Entwicklungsumgebung, wie das Textfeld für die Benutzereingabe oder der Zustand der Register, visualisieren.

In Abbildung 1 ist das geplante Layout der Webanwendung dargestellt, die sich an ähnlichen Online Compilern und Debuggern orientiert (Online GDB [5], CPUlator [10]). Auf der rechten Seite befindet sich ein großes Textfeld für die Eingabe des ARM-Codes vom Benutzer. 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.

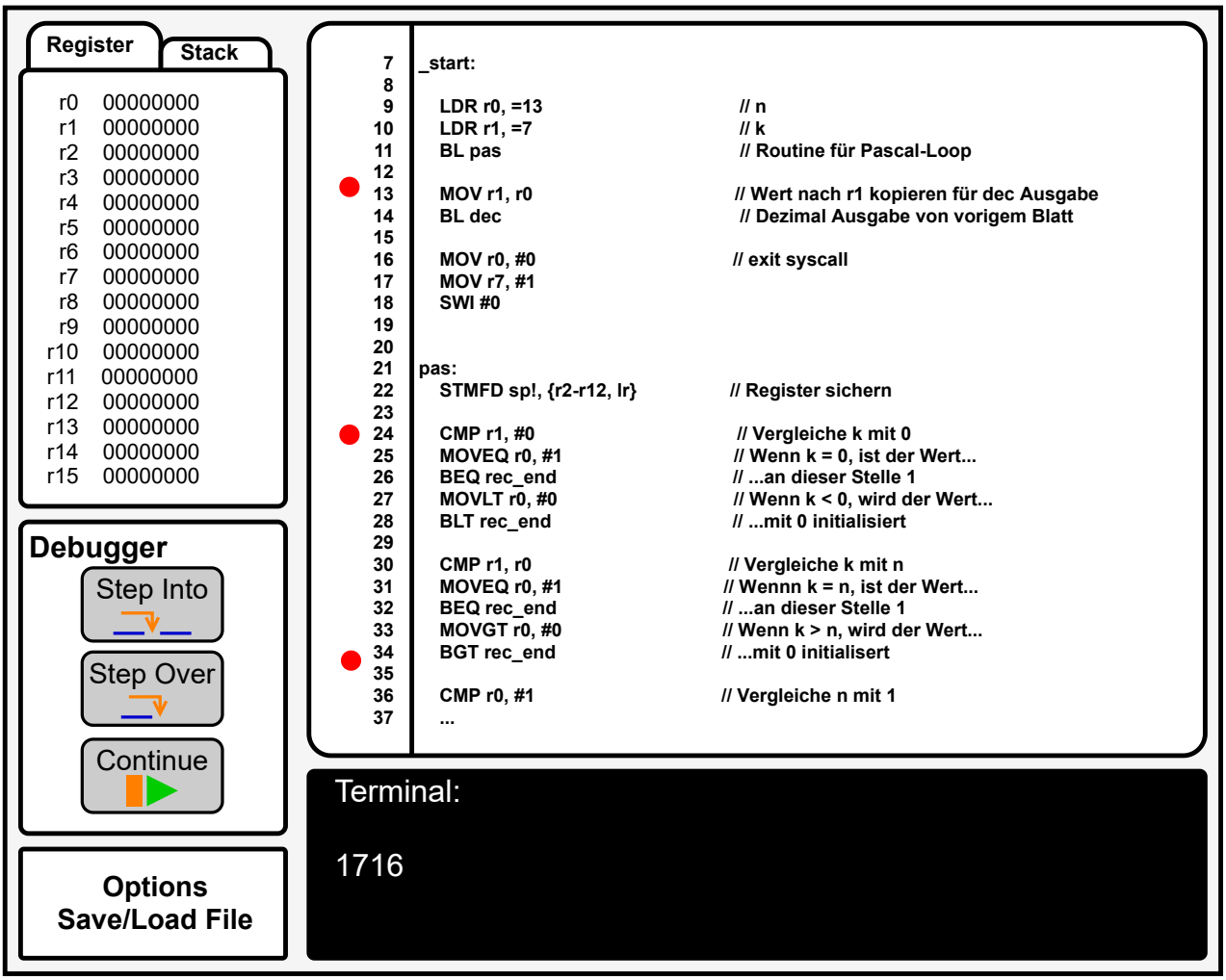

Abbildung 1: Geplantes Layout der Webanwendung mit Eingabefeld, Terminal, Ansicht für Register und Stack, Debugger und Optionsfeld. Code-Ausschnitt aus meiner Lösung für das Pascal-Dreieck.

Auf der 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 vom Benutzer verändert werden können. Darunter befinden sich die Optionen für den Debugger. Hier orientiere ich mich auch den den Standardfunktionen anderer Debugger. 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 die Subroutine übersprungen. *Continue* führt das Programm bis zum nächsten Breakpoint aus. Weitere Funktionen sind *Stop*, um das Programm im Falle einer Schleife zu beenden oder *Step Return*, um aus einer Subroutine herauszuspringen.

3 Vorgehensweise und Zeitplan

Der Zeitplan für mein Bachelorprojekt ist in Abbildung 2 abgebildet. Anfang März beginne ich mit dem Schreiben des Exposé und der dazugehörigen Präsentation. Auf der Seite des Backends beginne ich mit dem Parser für einfache arithmetische Operationen und erstelle dann

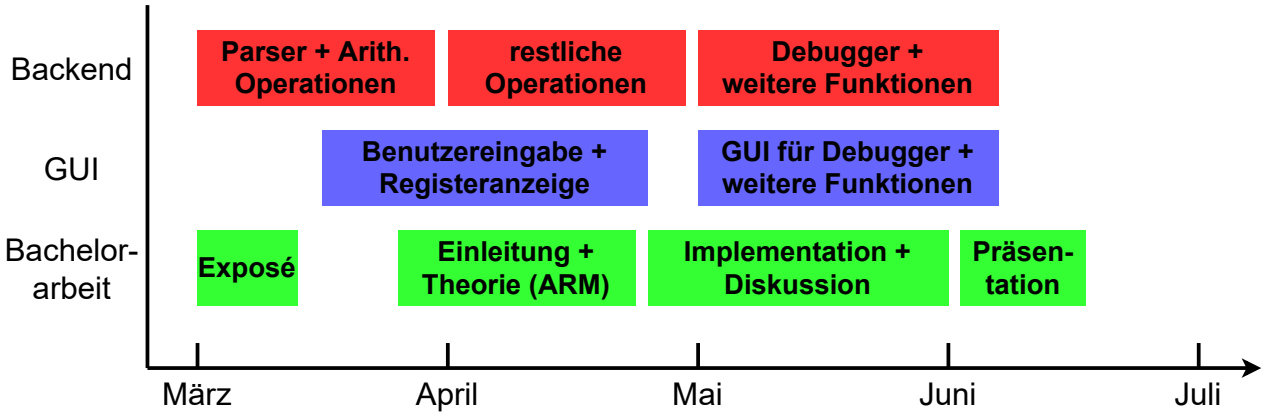

Abbildung 2: Zeitliche Planung der einzelnen Teile des Bachelorprojekts aufgeteilt in Backend, Gestalten der Benutzeroberfläche und Schreiben der Bachelorarbeit

die dazugehörige Benutzeroberfläche, um den Simulator für diese einfachen Instruktionen zu testen.

Im April füge ich dann die restlichen Operationen (Logische-, Kopier- und Speicherzugriffsoperationen) hinzu, arbeite weiter an der Visualisierung und beginne mit dem schreiben der Einleitung und Theorie für die Bachelorarbeit.

Im Mai implementiere ich dann den Debugger und die restlichen Funktionen (z.B. Speichern und Laden von Dateien oder Funktionen die ich vorher nicht fertigstellen konnte) inklusive deren Visualisierung. Nebenbei schreibe ich den Implementations- und Diskussionsteil der Bachelorarbeit.

Im Juni habe ich noch Zeit für etwaige Fehlerbehebungen, Verbesserungen und die Vorbereitung der Präsentation. Damit sollte ich meine Bachelorarbeit im Juni vorstellen können. Da ich keine anderen Lehrveranstaltungen mehr habe und meine volle Zeit auf die Bachelorarbeit konzentrieren kann, sollte dieser Zeitplan einhaltbar sein.

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

[6] Microsoft. Typescript. Zugegriffen am: 04.03.2021. `https://www.typescriptlang.org/`.

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

[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-project.gitlab.io/qemu/user/index.html.

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