Design und Implementierung eines Debuggers für Mikro-Assembler-Programme

STUDIENARBEIT

des Studiengangs Angewandte Informatik
der Dualen Hochschule Baden-Württemberg Stuttgart

von

CHRISTIAN RÖSCH

Oktober 2011 - Mai 2012

Bearbeitungszeitraum	8 Monate
Matrikelnummer	0487930
Kurs	TIT09AIC
Ausbildungsfirma	icon Systemhaus GmbH - Stuttgart
Gutachter der Studienakademie	Prof. Dr. Karl Stroetmann

Einleitung

Die vorliegende Arbeit stellt einen Debugger für einen CISC-Prozessor vor. Hierbei handelt es sich um den in [Str07] vorgestellten Prozessor *Mic-1*:

> "Die Abkürzung CISC steht hier für *complex instruction set computer* und drückt aus, dass die Maschinensprache wesentlich umfangreicher ist als bei einem RISC-Prozessor."

Im Gegensatz zu einem RISC-Prozessor sind die Maschinenbefehle in einem CISC-Prozessor nicht in Hardware implementiert, sondern werden durch einen Mikroprogramm-Speicher definiert. Dieser wird in Mikro-Assembler programmiert und ermöglicht es, die Maschinenbefehle flexibel zu steuern.

Der Debugger soll dazu die Entwicklung von Mikro-Assembler- und Assembler-Programmen für den CISC-Prozessor erleichtern.

Inhaltsverzeichnis

I Bedienung **1**

1 Allgemein **2**
1.1 Parameter . . . 2
1.2 Logs . . . 2

2 Interaktion per Konsole **3**

3 Interaktion per GUI **7**

II Implementierung **8**

4 Werkzeuge **9**
4.1 Versionskontrollsystem . . . 9
4.1.1 Code auschecken . . . 9
4.1.2 Feature implementieren . . . 9
4.1.3 Bug fixen . . . 9
4.2 Build-Management . . . 9
4.2.1 Code kompilieren . . . 9
4.2.2 Tests ausführen . . . 9
4.2.3 Abhängigkeiten ändern . . . 9
4.3 Entwicklungsumgebung . . . 9
4.3.1 Integration von git und maven . . . 9
4.3.2 Navigatieren im Code . . . 9
4.3.3 Finden von Referenzen . . . 9

5 Prozessor Mic1 **10**
5.1 ALU . . . 10
5.2 MPC-Berechnung . . . 10
5.3 Mic1-Instruktionen . . . 10
5.4 Register . . . 10
5.5 Hauptspeicher . . . 10
5.6 Zusammensetzung der Komponenten . . . 10

6 Interaktion per Konsole **11**

7 Interaktion per GUI **12**

Erklärung **IV**

Teil I

Bedienung

Kapitel 1

Allgemein

1.1 Parameter

1.2 Logs

Kapitel 2

Interaktion per Konsole

Zunächst soll der Debugger für eine Bedienung per Kommandozeile entwickelt werden. Der Benutzer soll den Debugger starten können und anschließend über die Eingabe von verschiedenen Kommandos steuern können.

Um entsprechenden (Mikro-)Assembler-Bytecode debuggen zu können, ist es nötig den CISC-Prozessor simulieren zu können. Im Jahr 2004 wurde von Thomas Kutzer eine Studienarbeit zum Thema "Mic-1 Simulator" verfasst. Der Debugger soll den darin entwickelten Simulator ersetzen und um einige Funktionen ergänzen. Der Debugger soll sich daher per Kommandozeile bedienen lassen, wie der Simulator — das heißt, er muss folgende Befehle unterstützen:

help
: Zeigt einen Hilfe-Text an, der die möglichen Befehle und jeweils eine kurze Beschreibung anzeigt.

run
: Der Debugger simuliert die Abarbeitung des Assembler- und Mikro-Assembler-Programms bis zu einem Haltepunkt oder bis zum Programmende.

step *[n]*
: Der Debugger simuliert die Ausführung von n Instruktionen des Assembler-Programms. Hierbei ist der Parameter n eine natürliche Zahl und kann weggelassen werden; in diesem Fall wird eine Instruktion ausgeführt.

reset
: Der Debugger wird in den Anfangszustand zurückgesetzt: Der Debugger verhält sich, als wäre er gerade gestartet worden.

set *Register Wert*
: Es wird das angegebene *Register* auf den übergebenen *Wert* gesetzt; dieser muss als Dezimalzahl eingegeben werden.

setmemory *Adresse Wert*
: Es wird das *Wort* an der angegebenen *Adresse* auf den übergebenen *Wert* gesetzt; *Adresse* und *Wert* müssen als Dezimalzahl eingegeben werden. Der Speicher wird byteweise addressiert.

break *Register=Wert*

Es wird ein bedingter Haltepunkt gesetzt: Sobald das angegebene *Register* den Wert *Wert* erhält, hält der Debugger.

ls-break
Es werden alle Haltepunkte angezeigt — mit der jeweiligen Bedingung.

Beachte: Dieser Befehl entspricht in etwa dem Befehl `rmbreak` aus dem Simulator von Thomas Kutzer. Da sich die Funktionalität allerdings verändert hat, wird alternativ der Befehl `ls-break` benutzt.

ls-reg *[Register]*

Es wird der Inhalt von *Register* angezeigt. Wird der optionale Parameter weggelassen, werden die Inhalte aller Register angezeigt.

Hinweis: Dieser Befehl entspricht dem Befehl `show` aus dem Simulator von Thomas Kutzer.

ls-mem *Start Stop*

Es wird der Inhalt des Hauptspeichers zwischen den Adressen *Start* und *Stop* angezeigt. Die Adressen müssen als Dezimalzahl angegeben werden und adressieren den Speicher byteweise.

Hinweis: Dieser Befehl entspricht dem Befehl `dump` aus dem Simulator von Thomas Kutzer.

trace-reg *[Register]*

Es wird das *Register* zum *Beobachten* registriert, dadurch wird dessen Inhalt nach jedem Simulationsschritt des Debuggers angezeigt. Wird der optionale Parameter weggelassen, werden alle Register zum *Beobachten* registriert.

Hinweis: Dieser Befehl entspricht dem Befehl `trace` aus dem Simulator von Thomas Kutzer.

untrace-reg *[Register]*

Es wird das *Register* vom *Beobachten* deregistriert, dadurch wird dessen Inhalt nach einem Simulationsschritt des Debuggers nicht mehr angezeigt. Wird der optionale Parameter weggelassen, werden alle Register deregistriert.

Hinweis: Dieser Befehl entspricht dem Befehl `untrace` aus dem Simulator von Thomas Kutzer.

trace-var *[Variable]*

Es wird die *Variable* zum *Beobachten* registriert, dadurch wird deren Inhalt nach jedem Simulationsschritt des Debuggers angezeigt. Wird der optionale Parameter weggelassen, werden alle Variablen zum *Beobachten* registriert.

untrace-reg *[Register]*

Es wird die *Variable* vom *Beobachten* deregistriert, dadurch wird deren Inhalt nach einem Simulationsschritt des Debuggers nicht mehr angezeigt. Wird der optionale Parameter weggelassen, werden alle Variablen deregistriert.

trace-mic
Es wird der *Mikro-Assembler-Code* zum *Beobachten* registriert, dadurch wird jede ausgeführte Mikro-Instruktion des Debuggers angezeigt.

untrace-mic
Es wird der *Mikro-Assembler-Code* zum *Beobachten* deregistriert, dadurch wird ausgeführte Mikro-Instruktionen des Debuggers nicht mehr angezeigt.

trace-mac
Es wird der *Assembler-Code* zum *Beobachten* registriert, dadurch wird jede ausgeführte Assembler-Instruktion des Debuggers angezeigt.

untrace-mac
Es wird der *Assembler-Code* zum *Beobachten* deregistriert, dadurch wird ausgeführte Assembler-Instruktionen des Debuggers nicht mehr angezeigt.

exit
Der Debugger wird beendet.

Zusätzlich zu den im Simulator vorhandenen Befehlen soll der Debugger folgende Befehle unterstützen:

micro-step *[n]*

Der Debugger simuliert die Ausführung von n Instruktionen des Mikro-Assembler-Programms. Hierbei ist der Parameter n eine natürliche Zahl und kann weggelassen werden; in diesem Fall wird eine Instruktion ausgeführt.

rm-break *Nummer*

Es wird der Haltepunkt *Nummer* entfernt. Der Parameter *Nummer* ist eine natürliche Zahl und bezieht sich auf die Nummerierung, die durch `ls-break` erhalten wird. Die Haltepunkte werden anschließend neu nummeriert, so dass ein erneuter Aufruf von `ls-break` empfohlen wird.

micro-break *Adresse*

Es wird ein Haltepunkt hinzugefügt, der die Simulation des Debuggers anhält, sobald der Bytecode des Mikro-Assemblers an der *Adresse* ausgeführt werden soll. *Adresse* wird hierbei als Dezimalzahl übergeben.

macro-break *Adresse*

Es wird ein Haltepunkt hinzugefügt, der die Simulation des Debuggers anhält, sobald der Bytecode des Assemblers an der *Adresse* ausgeführt werden soll. *Adresse* wird hierbei als Dezimalzahl übergeben.

ls-stack
Es wird der aktuelle Inhalt des Stacks angezeigt.

ls-micro-code *[Nummer1 [Nummer2]]*

Es wird der disassemblierte Programmcode des Mikro-Assemblers angezeigt. Dabei gibt es drei Varianten des Befehls:

- Ohne Parameter wird der komplette Mikro-Assembler-Code angezeigt.
- Mit einem Parameter *Nummer1* wird die angegebene Anzahl an Zeilen vor und nach der nächsten auszuführenden Codezeile angezeigt.
- Mit zwei Parametern, die beide Zeilennummern darstellen, wird der Code von Zeile *Nummer1* bis zur Zeile *Nummer2* angezeigt.

ls-macro-code *[Nummer1 [Nummer2]]*

Dieser Befehl entspricht dem Befehl `ls-micro-code`, mit dem Unterschied, dass `ls-macro-code` sich auf den disassemblierten Assembler-Code bezieht.

Unabhängig von den Befehlen soll es möglich sein gewisse Zahlen sowohl als Dezimal- als auch Hexadezimalzahl anzugeben. Möglicherweise sind auch Binär- und Oktalzahlen nötig. Es ist möglich dies über ein Parameter beim Programmstart global zu setzen, oder für jeden Befehl einzeln zu bestimmen.

Kapitel 3

Interaktion per GUI

Teil II

Implementierung

Kapitel 4

Werkzeuge

4.1 Versionskontrollsystem

4.1.1 Code auschecken

4.1.2 Feature implementieren

4.1.3 Bug fixen

4.2 Build-Management

4.2.1 Code kompilieren

4.2.2 Tests ausführen

4.2.3 Abhängigkeiten ändern

4.3 Entwicklungsumgebung

4.3.1 Integration von git und maven

4.3.2 Navigatieren im Code

4.3.3 Finden von Referenzen

Kapitel 5

Prozessor Mic1

5.1 ALU

5.2 MPC-Berechnung

5.3 Mic1-Instruktionen

5.4 Register

5.5 Hauptspeicher

5.6 Zusammensetzung der Komponenten

Kapitel 6

Interaktion per Konsole

Kapitel 7

Interaktion per GUI

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[Str07] Stroetmann, Karl: Computer-Architektur - Modellierung, Entwicklung und Verifikation mit Verilog: Grundlagen der Elektro- und Informationstechnik., 2007

Erklärung

gemäß §5 (2) der „Studien- und Prüfungsordnung DHBW Technik“ vom 18. Mai 2009.
Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema

Design und Implementierung eines Debuggers für Mikro-Assembler-Programme

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Stuttgart, 24. November 2011
Ort, Datum

Christian Rösch