
ALBERT LUDWIGS UNIVERSITÄT FREIBURG

TECHNISCHE FAKULTÄT

PicoC-Compiler

Übersetzung einer Untermenge von C in den Befehlssatz der RETI-CPU

BACHELORARBEIT

Abgabedatum: 28th April 2022

Author:
Jürgen Mattheis

Gutachter:
Prof. Dr. Scholl

Betreuung:
M.Sc. Seufert

Eine Bachelorarbeit am Lehrstuhl für
Betriebssysteme

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis	I
Codeverzeichnis	II
Tabellenverzeichnis	III
Definitionsverzeichnis	IV
Grammatikverzeichnis	V
1 Ergebnisse und Ausblick	1
1.1 Funktionsumfang	1
1.1.1 Kommandozeilenoptionen	1
1.1.2 Shell-Mode	3
1.1.3 Show-Mode	4
1.2 Qualitätssicherung	5
1.3 Erweiterungsideen	5
Literatur	A

Abbildungsverzeichnis

Codeverzeichnis

1.1	Shellaufruf und die Befehle <code>compile</code> und <code>quit</code>	3
1.2	Shell-Mode und der Befehl <code>most_used</code>	4

Tabellenverzeichnis

1.1	Kommandozeilenoptionen	2
1.2	Makefileoptionen	5

Definitionsverzeichnis

Grammatikverzeichnis

1 Ergebnisse und Ausblick

Zum Schluss soll ein **Überblick** über das gegeben werden, was im Kapitel ?? implementiert wurde. In Unterkapitel 1.1 wird mithilfe **kurzer Anleitungen** ein grober Einblick in die **wichtigsten Funktionalitäten** des implementierten **PicoC-Compilers** und **anderer mitimplementierter Tools** gegeben. Im Unterkapitel 1.2 wird aufgezeigt, was zur **Qualitätssicherung** implementiert wurde, um zu gewährleisten, dass der **PicoC-Compiler** die Kompilierung der **Programmiersprache** L_{PicoC} in **Syntax** und **Semantik** **identisch** zur entsprechenden **Untermenge** der Programmiersprache L_C umsetzt. Als allerletztes wird im Unterkapitel 1.3 ein Ausblick gegeben, wie der PicoC-Compiler **erweitert** werden könnte.

1.1 Funktionsumfang

Bei der Implementierung des **PicoC-Compilers** wurden verschiedene **Kommandozeilenoptionen** und **Modes** implementiert. Diese werden in den folgenden Kapiteln 1.1.1, 1.1.2 und 1.1.3 mithilfe kurzer **Anleitungen** erklärt.

Die kurzen **Anleitungen** in dieser **Schriftlichen Ausarbeitung** der Bachelorarbeit sollen nur zu einem **schnellen, grundlegenden Verständnis** der Verwendung des **PicoC-Compilers** und seiner **Kommandozeilenoptionen** und **Befehle** beihelfen, sowie zum Verständnis der **weiteren implementierten Tools**. Alle weiteren **Kommandozeilenoptionen** und **Befehle** sind für die Verwendung des PicoC-Compilers **unwichtig** und erweisen sich nur in **speziellen Situationen** als nützlich, weshalb für diese auf die **ausführlichere Dokumentation** unter [Link](#)¹ verwiesen wird.

1.1.1 Kommandozeilenoptionen

Will man einfach nur ein **Programm** `program.picoc` kompilieren ist das mit dem **PicoC-Compiler** genauso **unkompliziert** wie mit dem **GCC** durch einfaches **Angaben der Datei**, die kompiliert werden soll: `> picoc_compiler program.picoc`. Als Ergebnis des Kompiliervorgangs wird eine Datei `program.reti` mit dem entsprechenden **RETI-Code** erstellt, wobei für die **Benennung der Datei** einfach nur der **Basisname** der Datei `program` an eine neue **Dateiendung** `.reti` angehängt wird².

Daneben gibt es allerdings auch die Möglichkeit **Kommandozeilenoptionen** `<cli-options>` in der Form `> picoc_compiler <cli-options> program.picoc` mitanzugeben, von denen die **wichtigsten** in Tabelle 1.1 erklärt sind. Alle weiteren **Kommandozeilenoptionen** können in der **Dokumentation** unter [Link](#) nachgelesen werden.

¹https://github.com/matthejue/PicoC-Compiler/blob/new_architecture/doc/help-page.txt

²Beim **GCC** wird bei **Nicht-Angabe** eines **Dateinamen** mit der `-o` Option dagegen eine Datei mit der festen Namen `a.out` erstellt.

Kommandozeilenoption	Beschreibung	Standardwert
<code>-i, --intermediate_stages</code>	Gibt Zwischenschritte der Kompilierung in Form der verschiedenen Tokens , Ableitungsbäume , Abstrakten Syntaxbäume der verschiedenen Passes in Dateien mit entsprechenden Dateieindungen aber gleichem Basinamen aus. Im Shell-Mode erfolgt keine Ausgabe in Dateien, sondern nur im Terminal .	false , most_used: true
<code>-p, --print</code>	Gibt alle Dateiausgaben auch im Terminal aus. Diese Option ist im Shell-Mode dauerhaft aktiviert.	false (true im Shell-Mode und für den most_used- Befehl)
<code>-v, --verbose</code>	Fügt den verschiedenen Zwischenschritten der Kompilierung , unter anderem auch dem finalen RETI-Code Kommentare hinzu, welche ein Statement oder Befehl aus einem vorherigen Pass beinhalten, der durch die darunterliegenden Statements oder Befehle ersetzt wurde. Wenn die <code>--run</code> -Option aktiviert ist, wird der Zustand der virtuellen RETI-CPU vor und nach jedem Befehl angezeigt.	false
<code>-vv, --double_verbose</code>	Hat dieselben Effekte , wie die <code>--verbose</code> -Option, aber bewirkt zusätzlich weitere Effekte . PicoC-Knoten erhalten bei der Ausgabe in den Abstrakten Syntaxbäumen zusätzliche runde Klammern , sodass direkter abgelesen werden kann, wo ein Knoten anfängt und wo einer aufhört. In Fehlermeldungen werden mehr Tokens angezeigt, die an der Stelle der Fehlermeldung erwartet worden wären. Bei Aktivierung der <code>--intermediate_stages</code> -Option werden in den dadurch ausgegebenen Abstrakten Syntaxbäumen ebenfalls versteckte Attribute , die Informationen zu Datentypen und für Fehlermeldungen beinhalten angezeigt.	false
<code>-h, --help</code>	Zeigt die Dokumentation , welche ebenfalls unter Link gefunden werden kann im Terminal an. Mit der <code>--color</code> -Option kann die Dokumentation mit farblicher Hervorhebung im Terminal angezeigt werden.	false
<code>-R, --run</code>	Führt die RETI-Befehle , die das Ergebnis der Kompilierung sind mit einer virtuellen RETI-CPU aus. Wenn die <code>--intermediate_stages</code> -Option aktiviert ist, wird eine Datei <code><basename>.reti_states</code> erstellt, welche den Zustand der RETI-CPU nach dem letzten ausgeführten RETI-Befehl enthält. Wenn die <code>--verbose</code> - oder <code>--double_verbose</code> -Option aktiviert ist, wird der Zustand der RETI-CPU vor und nach jedem Befehl auch noch zusätzlich in die Datei <code><basename>.reti_states</code> ausgegeben.	false , most_used: true
<code>-B, --process_begin</code>	Setzt die relative Adresse , wo der Prozess bzw. das Codesegment für das ausgeführte Programm beginnt.	3
<code>-D, --datasegment_size</code>	Setzt die Größe des Datensegments . Diese Option muss mit Vorsicht gesetzt werden, denn wenn der Wert zu niedrig gesetzt wird, dann können die Globalen Statischen Daten und der Stack miteinander kollidieren.	32

Tabelle 1.1: Kommandozeilenoptionen

Alle **kleingeschriebenen** Kommandozeilenoptionen, wie `-i`, `-p`, `-v` usw. betreffen dabei den **PicoC-Compiler** und alle **großgeschriebenen** Kommandozeilenoptionen, wie `-R`, `-B`, `-D` usw. betreffen den **RETI-Interpreter**.

1.1.2 Shell-Mode

Will man z.B. eine **Folge von Statements** in der Programmiersprache L_{PicoC} **schnell** kompilieren ohne eine Datei erstellen zu müssen, so kann der **PicoC-Compiler** im sogenannten **Shell-Mode** aufgerufen werden. Hierzu wird der PicoC-Compiler **ohne Argumente** `> picoc_compiler` aufgerufen, wie es in Code 1.1 zu sehen ist. Die angegebene **Folge von Statements** `<seq-of-stmts>` wird dabei automatisch in eine `main`-Funktion eingefügt: `void main(){<seq-of-stmts>}`.

Mit dem `> compile <cli-options> <filename>`-Befehl (oder der **Abkürzung** `cpl`) kann **PicoC-Code** zu **RETI-Code** kompiliert werden. Die Kommandozeilenoptionen `<cli-options>` sind dieselben, wie wenn der Compiler **direkt** mit Kommandozeilenoptionen aufgerufen wird. Die **wichtigsten** dieser **Kommandozeilenoptionen** sind in Tabelle 1.1 angegeben.

Mit dem Befehl `> quit` kann der **Shell-Mode** wieder **verlassen** werden.

```
> picoc_compiler
PicoC Shell. Enter `help` (shortcut `?`) to see the manual.
PicoC> cpl "6 * 7;";
----- RETI -----
SUBI SP 1;
LOADI ACC 6;
STOREIN SP ACC 1;
SUBI SP 1;
LOADI ACC 7;
STOREIN SP ACC 1;
LOADIN SP ACC 2;
LOADIN SP IN2 1;
MULT ACC IN2;
STOREIN SP ACC 2;
ADDI SP 1;
LOADIN BAF PC -1;

Compilation successfull

PicoC> quit
```

Code 1.1: Shellaufruf und die Befehle *compile* und *quit*

Wenn man möglichst alle nützlichen **Kommandozeilenoptionen** direkt aktiviert haben will, bei denen es **keinen** Grund gibt, sie nicht mitanzugeben, kann der Befehl `> most_used <cli-options> <filename>` (oder seine **Abkürzung** `mu`) genutzt werden, um diese Kommandozeilenoptionen mit dem `compile`-Befehl **nicht** jedes mal **selbst** Angeben zu müssen. In der Tabelle 1.1 sind in grau die Werte der einzelnen **Kommandozeilenoptionen** angegeben, die bei dem Befehl `most_used` gesetzt werden. In Code 1.2 ist der `most_used`-Befehl in seiner Verwendung zu sehen.






Dadurch, dass die `--intermediate_stages`- und die `--run`-Option beim `most_used`-Befehl aktiviert sind, werden die verschiedenen **Zwischenstufen** der Kompilierung, wie **Tokens**, **Derivation Tree** usw., sowie der **Zustand der RETI-CPU** nach der Ausführung des **letzten** Befehls angezeigt. Aus **Platzgründen** ist das meiste allerdings mit `'...'` ausgelassen.


```

PicoC> mu "int var = 42;";
----- Code -----
// stdin.picoc:
void main() {int var = 42;}
----- Tokens -----
...
----- Derivation Tree -----
...
----- Derivation Tree Simple -----
...
----- Abstract Syntax Tree -----
...
----- PicoC Shrink -----
...
----- PicoC Blocks -----
...
----- PicoC Mon -----
...
----- Symbol Table -----
...
----- RETI Blocks -----
...
----- RETI Patch -----
...
----- RETI -----
SUBI SP 1;
LOADI ACC 42;
STOREIN SP ACC 1;
LOADIN SP ACC 1;
STOREIN DS ACC 0;
ADDI SP 1;
LOADIN BAF PC -1;
----- RETI Run -----
...
Compilation successfull

```

Code 1.2: Shell-Mode und der Befehl *most_used*

Im **Shell-Mode** kann der **Cursor** mit den  und  Pfeiltasten bewegt werden. In der **Befehlshistorie** kann sich mit den  und  Pfeiltasten **rückwärts** und **vorwärts** bewegt werden. Mit  kann ein Befehl **automatisch vervollständigt** werden.

Es gibt für den **Shell-Mode** noch **weitere Befehle**, wie `color_toggle`, `history` etc. und **kleinere Funktionalitäten** für die Shell, die sich in der ein oder anderen Situation als **nützlich** erweisen können. Da in der Bachelorarbeit mit den Anleitungen allerdings nur eine **kleine Einführung** in die **Verwendung** gegeben werden soll, wird an dieser Stelle beim Bestehen von weitergehendem Interesse auf die **Dokumentation** unter [Link](#) verwiesen, welche auch über den Befehl  angezeigt werden kann.

1.1.3 Show-Mode

Der **Show-Mode** ist ein Nebenprodukt der Implementierung des **PicoC-Compilers**. Dieser **Mode** wurde eigentlich nur implementiert, um beim **Testen** des PicoC-Compilers **Bugs** bei der Generierung des **RETI-**

Code zu finden, indem im Terminal eine **virtuelle RETI-CPU** angezeigt wird, welches den **kompletten Zustand** einer virtuell ausgeführten RETI mit allen **Registern**, **SRAM**, **UART**, **EPROM** und einigen **weiteren Informationen** anzeigt.

Allerdings bringt die Möglichkeit des **Show-Mode** die **RETI-Befehle** des übersetzten Programmes in **Ausführung zu sehen** auch einen großen **Lerneffekt** mit sich, weshalb der **Show-Mode** noch **weiterentwickelt** wurde, sodass auch **Studenten** ihn auf unkomplizierte Weise nutzen können.

Der **Show-Mode** kann auf die **einfachste Weise** mittels der `/Makefile` des **PicoC-Compilers** mit dem Befehl `make show FILEPATH=<path-to-file>` gestartet werden. Alle **weiteren einstellbaren Optionen** `<more-options>` für die **Makefile** sind in Tabelle 1.2 aufgelistet.

Kommandozeilenoption	Beschreibung	Standardwert
	beschreibung	false

Tabelle 1.2: Makefileoptionen

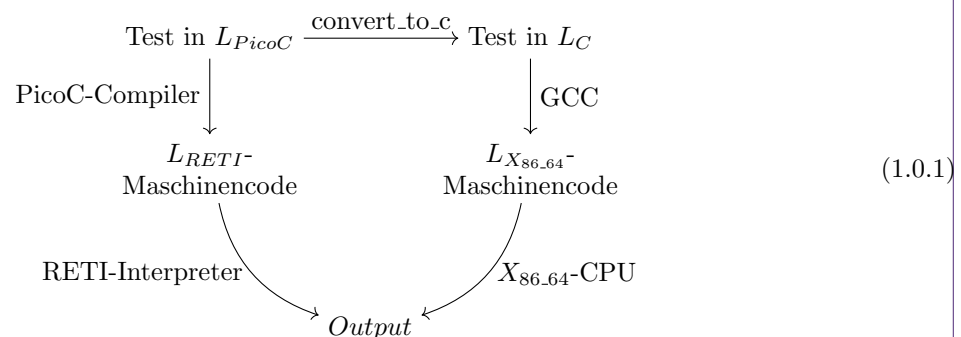
Alternativ kann der **Show-Mode** mit dem Befehl `make test-show TESTNAME=<testname> <more-options>` auch für einen der geschriebenen **Tests** im Ordner `/Tests` gestartet werden. Der **Test** wird bei diesem Befehl **erst ausgeführt** und dann im **Show-Mode** angezeigt.

Der **Show-Mode** nutzt den Terminal Texteditor **Neovim**³ um einen Dateinhalt über möglichst viele Fenster verteilt anzuzeigen und alle Fenster gleichzeitig zu scrollen

Wird der **Show-Mode** für den FILETYPE=reti gestartet, so kann dieser als eine ART RETI-CODE-Debugger angesehen werden.

Durch Drücken von `Tab` und `↑ -Tab` kann zwischen den **verschiedenen Zuständen** der RETI-CPU **vor** und **nach** der Ausführung eines Befehls **gewechselt** werden. Durch drücken von `Esc` oder `q` kann der **Show-Mode** wieder verlassen werden.

1.2 Qualitätssicherung



1.3 Erweiterungsideen

³Home - Neovim.

Literatur

Online

- *Home - Neovim*. URL: <http://neovim.io/> (besucht am 04.08.2022).