
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	6
1.3 Erweiterungsideen	8
Literatur	A

Abbildungsverzeichnis

1.1 Show-Mode in der Verwendung	6
---	---

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
1.3	Typischer Test	7

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 **Angeben 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. Für die **Erklärung** dieser wird allerdings 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> <more-options>` gestartet werden. Alle **einstellbaren Optionen**, die z.B. unter `<more-options>` noch für die **Makefile** gesetzt werden können sind in Tabelle 1.2 aufgelistet.

Kommandozeilenoption	Beschreibung	Standardwert
FILEPATH	Pfad zur Datei, die im Show-Mode angezeigt werden soll	<code>()</code>
TESTNAME	Name des Tests. Alles andere als der Basisname , wie die Dateiendung wird abgeschnitten	<code>()</code>
EXTENSION	Dateiendung , die an TESTNAME angehängt werden soll zu <code>./tests/TESTNAME.EXTENSION</code>	<code>reti_states</code>
NUM_WINDOWS	Anzahl Fenster auf die ein Dateiinhalte verteilt werden soll	<code>5</code>
VERBOSE	Möglichkeit die Kommandozeilenoption <code>-v</code> oder <code>-vv</code> zu aktivieren für eine ausführlichere Ausgabe	<code>()</code>
DEBUG	Möglichkeit die Kommandozeilenoption <code>-d</code> zu aktivieren, um bei <code>make test-show TESTNAME=<testname></code> den Debugger für den entsprechenden Test <code><testname></code> zu starten	<code>()</code>

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 der **Show-Mode** gestartet.

Der **Show-Mode** nutzt den Terminal Texteditor **Neovim**³ um einen **Dateiinhalte** über mehrere **Fenster** verteilt anzuzeigen, so wie es in Abbildung 1.1 zu sehen ist. Für den **Show-Mode** wird eine eigene **Konfiguration für Neovim** verwendet, welche in der **Konfigurationsdatei** `/interpr_showcase.vim` spezifiziert ist.

Gedacht ist der **Show-Mode** vor allem dafür etwas ähnliches wie ein **RETI-Debugger** zu sein und wird daher standardmäßig bei **Nicht-Angabe** einer **EXTENSION** auf die Datei `<program>.reti_states` angewandt. Der **Show-Mode** kann aber auch dazu genutzt werden **andere Dateien**, welche verschiedene Zwischenschritte der Kompilierung darstellen anzuzeigen, indem **EXTENSION** auf eine andere **Dateiendung** gesetzt wird.

Im **Show-Mode** wird ein Trick angewandt, indem die verschiedenen **Zustände der RETI-CPU nicht zur Laufzeit** des **Show-Mode** berechnet werden, sondern schon berechnet wurden und nacheinander in die Datei `<program>.reti_states` ausgegeben wurden. Der **Show-Mode** macht nichts anderes, als immer an die Stelle zu springen, an welcher der nächste Zustand anfängt. Durch Drücken von `Tab` und `↑ -Tab` können auf diese Weise die **verschiedenen Zuständen der RETI-CPU vor** und **nach** der Ausführung eines Befehls **angezeigt** werden.

³Home - Neovim.

index: 43	00019 ADDI SP 1;	00057 LOADIN DS ACC 0;	00095 STOREIN SP ACC 2;	00133 0
instruction: ADDI SP 1;	00020 LOADIN SP ACC 1;	00058 STOREIN SP ACC 1;	00096 ADDI SP 1;	00134 0
ACC: 1	00021 STOREIN DS ACC 0;	00059 LOADIN SP ACC 1;	00097 LOADIN SP ACC 1;	00135 0
ACC_SIMPLE: 1	00022 ADDI SP 1;	00060 ADDI SP 1;	00098 STOREIN DS ACC 0;	00136 0
IN1: 0	00023 SUBI SP 1;	00061 CALL PRINT ACC;	00099 ADDI SP 1;	00137 0
IN1_SIMPLE: 0	00024 LOADIN DS ACC 0;	00062 SUBI SP 1;	00100 JUMP -32;	00138 0
IN2: 4	00025 STOREIN SP ACC 1;	00063 LOADI ACC 0;	00101 SUBI SP 1;	00139 0
IN2_SIMPLE: 4	00026 SUBI SP 1;	00064 STOREIN SP ACC 1;	00102 LOADIN DS ACC 0;	00140 0
PC: 2147483686	00027 LOADI ACC 4;	00065 LOADIN SP ACC 1;	00103 STOREIN SP ACC 1;	00141 0
PC_SIMPLE: 38	00028 STOREIN SP ACC 1;	00066 STOREIN DS ACC 0;	00104 LOADIN SP ACC 1;	00142 0
SP: 2147483792	00029 LOADIN SP ACC 2;	00067 ADDI SP 1;	00105 ADDI SP 1;	00143 0
SP_SIMPLE: 144	00030 LOADIN SP IN2 1;	00068 SUBI SP 1;	00106 JUMP== 7;	00144 4 <- SP
BAF: 2147483650	00031 SUB ACC IN2;	00069 LOADIN DS ACC 0;	00107 SUBI SP 1;	00145 1
BAF_SIMPLE: 2	00032 JUMP< 3;	00070 STOREIN SP ACC 1;	00108 LOADIN DS ACC 0;	UART:
CS: 2147483651	00033 LOADI ACC 0;	00071 SUBI SP 1;	00109 STOREIN SP ACC 1;	00000 0
CS_SIMPLE: 3	00034 JUMP 2;	00072 LOADI ACC 2;	00110 LOADIN SP ACC 1;	00001 0
DS: 2147483762	00035 LOADI ACC 1;	00073 STOREIN SP ACC 1;	00111 ADDI SP 1;	00002 0
DS_SIMPLE: 114	00036 STOREIN SP ACC 2;	00074 LOADIN SP ACC 2;	00112 CALL PRINT ACC;	00003 0
SRAM:	00037 ADDI SP 1;	00075 LOADIN SP IN2 1;	00113 LOADIN BAF PC -1;	EPROM:
00000 JUMP 0;	00038 LOADIN SP ACC 1; <- PC	00076 SUB ACC IN2;	00114 3 <- DS	00000 LOADI DS -2097152; <- IN1
00001 2147483648	00039 ADDI SP 1;	00077 JUMP< 3;	00115 0	00001 MULTI DS 1824; <- ACC
00002 0 <- BAF	00040 JUMP== 2;	00078 LOADI ACC 0;	00116 0	00002 MOVE DS SP;
00003 CALL INPUT ACC; <- CS	00041 JUMP -32;	00079 JUMP 2;	00117 0	00003 MOVE DS BAF;
00004 SUBI SP 1;	00042 SUBI SP 1;	00080 LOADI ACC 1;	00118 0	00004 MOVE DS CS; <- IN2
00005 STOREIN SP ACC 1;	00043 LOADIN DS ACC 0;	00081 STOREIN SP ACC 2;	00119 0	00005 ADDI SP 145;
00006 LOADIN SP ACC 1;	00044 STOREIN SP ACC 1;	00082 ADDI SP 1;	00120 0	00006 ADDI BAF 2;
00007 STOREIN DS ACC 0;	00045 SUBI SP 1;	00083 LOADIN SP ACC 1;	00121 0	00007 ADDI CS 3;
00008 ADDI SP 1;	00046 LOADI ACC 2;	00084 ADDI SP 1;	00122 0	00008 ADDI DS 114;
00009 SUBI SP 1;	00047 STOREIN SP ACC 1;	00085 JUMP== 16;	00123 0	00009 MOVE CS PC;
00010 LOADIN DS ACC 0;	00048 LOADIN SP ACC 2;	00086 SUBI SP 1;	00124 0	
00011 STOREIN SP ACC 1;	00049 LOADIN SP IN2 1;	00087 LOADIN DS ACC 0;	00125 0	index: 44
00012 SUBI SP 1;	00050 SUB ACC IN2;	00088 STOREIN SP ACC 1;	00126 0	instruction: LOADIN SP ACC 1;
00013 LOADI ACC 1;	00051 STOREIN SP ACC 2;	00089 SUBI SP 1;	00127 0	ACC: 1
00014 STOREIN SP ACC 1;	00052 ADDI SP 1;	00090 LOADI ACC 1;	00128 0	ACC_SIMPLE: 1
00015 LOADIN SP ACC 2;	00053 LOADIN SP ACC 1;	00091 STOREIN SP ACC 1;	00129 0	IN1: 0
00016 LOADIN SP IN2 1;	00054 ADDI SP 1;	00092 LOADIN SP ACC 2;	00130 0	IN1_SIMPLE: 0
00017 ADD ACC IN2;	00055 JUMP== 13;	00093 LOADIN SP IN2 1;	00131 0	IN2: 4
00018 STOREIN SP ACC 2;	00056 SUBI SP 1;	00094 ADD ACC IN2;	00132 0	IN2_SIMPLE: 4
00019 ADDI SP 1;	00057 LOADIN DS ACC 0;	00095 STOREIN SP ACC 2;	00133 0	PC: 2147483687

Abbildung 1.1: Show-Mode in der Verwendung

Zur **besseren Orientierung** wird für alle Register ebenfalls ein mit der Registerbezeichnung beschrifteter **Zeiger** <- REG an Adressen im **EPROM**, **UART** und **SRAM** angezeigt, je nachdem, ob der **Wert im Register** nach der **Memory Map** dem **Adressbereich** von **EPROM**, **UART** oder **SRAM** entspricht.

Durch Drücken von **Esc** oder **q** kann der **Show-Mode** wieder verlassen werden. Es gibt für den **Show-Mode** noch viele weitere **Tastenkürzel**, die sich in der ein oder anderen Situation als **nützlich** erweisen können. Für die **Erklärung** dieser wieder allerdings auf die **Dokumentation** unter [Link](#) verwiesen. Des Weiteren stehen durch die Nutzung des Terminal Texteditors **Neovim** auch alle **Funktionalitäten** dieses mächtigen Terminal Texteditors zur Verfügung, welche mittels der Eingabe von **:help** **nachgelesen** werden können oder mittels der Eingabe von **:Tutor** mithilfe einer kurzen **Einführungsanleitung** **erlernt** werden können.

1.2 Qualitätssicherung

Um verifizieren zu können, dass der **PicoC-Compiler** sich genauso verhält, wie er soll, müssen die Beziehungen aus Diagramm ?? in Unterkapitel ?? genauso für den **PicoC-Compiler** gelten. Für den **PicoC-Compiler** lässt sich ein ebensolches Diagramm 1.3 definieren. Der Test T_1 in der Sprache L_{PicoC} muss die gleiche **Semantik** haben, wie der Test T_2 in der Sprache L_{RETI} , trotz der unterschiedlichen **Syntax**. Dass die Tests in beiden Sprachen die gleiche Semantik haben, lässt sich dadurch verifizieren, dass beide die gleiche Ausgabe haben.

Die **Qualität** des **PicoC-Compilers** ist **zweifach** gesichert. Die Kante von Test T_1 zur **Ausgabe** aus Diagramm 1.2.1 ist dadurch erfüllt, dass jeder Test im **/Tests**-Verzeichnis eine **expected:<space_seperated_output>**-Zeile hat, in welcher der Schreiber des Tests die Rolle des entsprechenden Interpreters⁴ aus Diagramm ?? übernimmt.

Ein Beispiel für einen Test ist in Code 1.3 zu sehen. Sobald die Tests mithilfe der **/Makefile** mit dem Befehl

⁴Der die **Semantik** des Tests umsetzt.

`make test` ausgeführt werden, wird für jeden Test das Bashscript `/extract_input_and_expected.sh` ausgeführt, welches die Zeilen `// in:<space_seperated_input>`, `// expected:<space_seperated_output>` und `// datasegment:<datasegment_size>` extrahiert und die entsprechenden Werte in neu erstellte Dateien `<program>.in`, `<program>.out_expected` und `<program>.datasegment_size` schreibt.

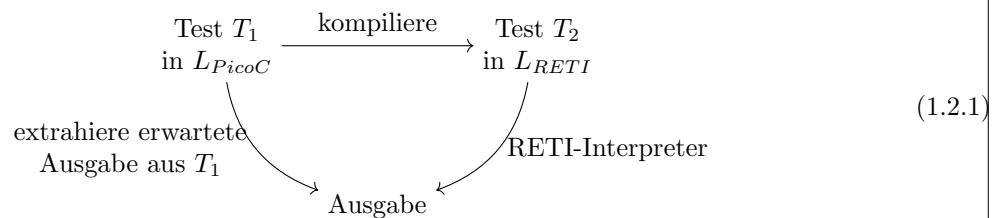
```
// in:21 2 6 7
// expected:42 42
// datasegment:4

void main() {
    print(input() * input());
    print(input() * input());
}
```

Code 1.3: *Typischer Test*

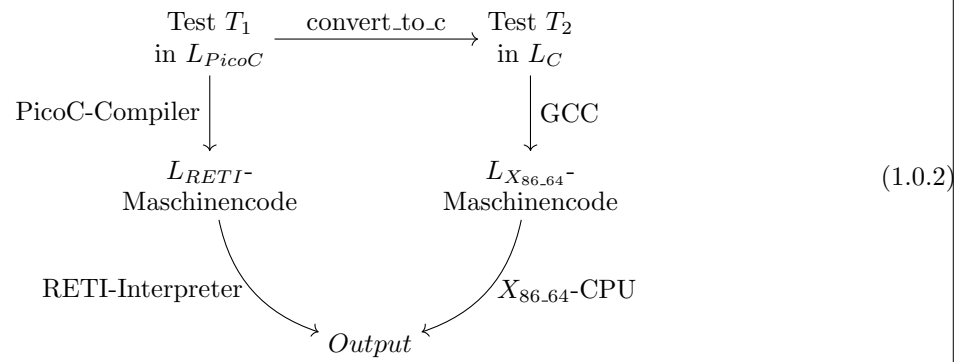
Die Kante von Test T_2 zur *Ausgabe* aus Abbildung 1.2.1 ist dadurch erfüllt, dass der kompilierte Test T_2 vom **RETI-Interpreter** interpretiert wird und jedes mal beim Antreffen des **RETI-Befehls** `CALL PRINT ACC` der entsprechende Inhalt des **ACC**-Registers in eine Datei `<program>.out` ausgegeben wird. Ein Test, der einen bestimmten Teil des Semantik des **PicoC-Compilers** abdeckt, kann die Korrektheit dieser Semantik verifizieren, wenn der Inhalt von `<program>.out_expected` und `<program>.out` identisch ist.

Wenn immer mehr Tests, die alle einen unterschiedlichen Teil der Semantik der Sprache L_{PicoC} abdecken vorliegen, bei denen allen der Inhalt der Dateien `<program>.out_expected` und `<program>.out` identisch ist, dann kann mit immer höherer Wahrscheinlichkeit von einem funktionierenden Compiler ausgegangen werden.



Darüberhinaus ist die Qualität des **PicoC-Compilers** doppelt gesichert, denn zusätzlich zu der Verifikation in Diagramm 1.2.1, wird auch die Ausgabe des RETI-Interpreters, der den RETI-Code eines kompilierten Tests T_1 interpretiert mit der Ausgabe eines auf der CPU einer jeweiligen Maschine ausgeführten Maschinencodes, der das Ergebnis eines durch den GCC kompilierten Tests T_2 ist verglichen.

in Diagramm 1.0.2.



1.3 Erweiterungsideen

Literatur

Online

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