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

Betreung:
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

Codeverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

Grammatikverzeichnis

0.0.1 Umsetzung von Pointern

0.0.1.1 Referenzierung

Die **Referenzierung** (z.B. `&var`) wird im Folgenden anhand des Beispiels in Code 0.1 erklärt.

```
1 void main() {
2   int var = 42;
3   int *pntr = &var;
4 }
```

Code 0.1: PicoC-Code für Pointer Referenzierung

Der Knoten `Ref(Name('var'))` repräsentiert im **Abstract Syntax Tree** in Code 0.2 eine **Referenzierung** `&var` und der Knoten `PntrDecl(Num('1'), IntType('int'))` repräsentiert einen Pointer `*pntr`.

```
1  File
2    Name './example_pntr_ref.ast',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
10         Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr')),
           ↪  Ref(Name('var')))
11       ]
12   ]
```

Code 0.2: Abstract Syntax Tree für Pointer Referenzierung

Bevor man einem **Pointer** eine eine **Adresse** (z.B. `&var`) zuweisen kann, muss dieser erstmal **definiert** sein. Dafür braucht es einen Eintrag in der **Symboltabelle** in Code 0.3.

> Die **Größe** eines Pointers (z.B. eines Pointers auf ein Array von `int`: `pntr = int *pntr[3]`), die ihm `size`-Feld der **Symboltabelle** eingetragen ist, ist dabei immer: `size(pntr) = 1`.

```
1  SymbolTable
2    [
3      Symbol
4        {
5          type qualifier:          Empty()
6          datatype:                FunDecl(VoidType('void'), Name('main'), [])
7          name:                    Name('main')
8          value or address:        Empty()
9          position:                Pos(Num('1'), Num('5'))
10         size:                    Empty()
```

```
    },
  Symbol
    {
      type qualifier:         Writeable()
      datatype:               IntType('int')
      name:                   Name('var@main')
      value or address:       Num('0')
      position:               Pos(Num('2'), Num('6'))
      size:                   Num('1')
    },
  Symbol
    {
      type qualifier:         Writeable()
      datatype:               PntrDecl(Num('1'), IntType('int'))
      name:                   Name('pntr@main')
      value or address:       Num('1')
      position:               Pos(Num('3'), Num('7'))
      size:                   Num('1')
    }
]
```

Code 0.3: Symboltabelle für Pointer Referenzierung

Im **PicoC-Mon Pass** in Code 0.4 wird der Knoten `Ref(Name('var'))` durch die Knoten `Ref(GlobalRead(Num('0')))` und `Assign(GlobalWrite(Num('1')), Tmp(Num('1')))` ersetzt. Im Fall, dass in `Ref(exp))` das `exp` vielleicht nicht direkt ein `Name('var')` enthält und `exp` z.B. ein `Subscr(Attr(Name('var')))` ist, sind noch weitere Anweisungen zwischen den Zeilen `11` und `12` nötig, die sich in diesem Beispiel um das Übersetzen von `Subscr(exp)` und `Attr(exp)` nach dem Schema in Subkapitel 0.0.4.2 kümmern.

```
File
  Name './example_pntr_ref.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('var'), Num('42'))
        Exp(Num('42'))
        Assign(Global(Num('0')), Stack(Num('1')))
        // Assign(Name('pntr'), Ref(Name('var')))
        Ref(Global(Num('0')))
        Assign(Global(Num('1')), Stack(Num('1')))
        Return(Empty())
      ]
  ]
```

Code 0.4: PicoC-Mon Pass für Pointer Referenzierung

Im **RETI-Blocks Pass** in Code 0.5 werden die **PicoC-Knoten** `Ref(Global(Num('0')))` und `Assign(Global(Num('1')), Stack(Num('1')))` durch ihre entsprechenden **RETI-Knoten** ersetzt.

```
File
  Name './example_pntr_ref.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('var'), Num('42'))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Assign(Global(Num('0')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 0;
        ADDI SP 1;
        # // Assign(Name('pntr'), Ref(Name('var')))
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
        # Assign(Global(Num('1')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 1;
        ADDI SP 1;
        # Return(Empty())
        LOADIN BAF PC -1;
      ]
  ]
```

Code 0.5: RETI-Blocks Pass für Pointer Referenzierung

0.0.1.2 Dereferenzierung durch Zugriff auf Arrayindex ersetzen

Die **Dereferenzierung** (z.B. `*var`) wird im Folgenden anhand des Beispiels in Code 0.6 erklärt.

```
void main() {
  int var = 42;
  int *pntr = &var;
  *pntr;
}
```

Code 0.6: PicoC-Code für Pointer Dereferenzierung

Der Knoten `Deref(Name('var'))` repräsentiert im **Abstract Syntax Tree** in Code 0.7 eine **Dereferenzierung** `*var`.

```
File
  Name './example_pntr_deref.ast',
  [
    FunDef
```

```
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
10        Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr')),
          ↪  Ref(Name('var')))
11        Exp(Deref(Name('pntr'), Num('0')))
12      ]
13    ]
```

Code 0.7: Abstract Syntax Tree für Pointer Dereferenzierung

Im **PicoC-Shrink Pass** in Code 0.8 wird ein Trick angewandet, bei dem jeder Knoten `Deref(Name('pntr'), Num('0'))` einfach durch den Knoten `Subscr(Name('pntr'), Num('0'))` ersetzt wird. Der Trick besteht darin, dass der **Dereferenzoperator** (z.B. `*(var + 1)`) sich identisch zum **Operator für den Zugriff auf einen Arrayindex** (z.B. `var[1]`) verhält[1]. Damit sparrt man sich viele vermeidbare **Fallunterscheidungen** und **doppelten Code** und kann die **Derefenzierung** (z.B. `*(var + 1)`) einfach von den Routinen für einen **Zugriff auf einen Arrayindex** (z.B. `var[1]`) übernehmen lassen.

```
1 File
2   Name './example_pntr_deref.picoc_shrink',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
10        Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr')),
          ↪  Ref(Name('var')))
11        Exp(Subscr(Name('pntr'), Num('0')))
12      ]
13    ]
```

Code 0.8: PicoC-Shrink Pass für Pointer Dereferenzierung

0.0.2 Umsetzung von Arrays

0.0.2.1 Initialisierung von Arrays

Die **Initialisierung** eines **Arrays** (z.B. `int ar[2][1] = {{3+1}, {4}}`) wird im Folgenden anhand des Beispiels in Code 0.9 erklärt.

[1] In der Sprache L_C gibt es einen Unterschied bei der Initialisierung bei z.B. `int *var = "string"` und z.B. `int var[1] = "string"`, der allerdings nichts mit den beiden Operatoren zu tuen hat, sondern mit der **Initialisierung**, bei der die Sprache L_C verwirrenderweise die eckigen Klammern `[]` genauso, wie beim **Operator für den Zugriff auf einen Arrayindex**, vor den Bezeichner schreibt (z.B. `var[1]`), obwohl es ein **Derived Datatype** ist.

```c
void main() {
  int ar[2][1] = {{3+1}, {4}};
}

void fun() {
  int ar[2][2] = {{3, 4}, {5, 6}};
}
```

Code 0.9: PicoC-Code für Array Initialisierung

Die **Initialisierung** eines **Arrays** `int ar[2][1] = {{3+1}, {4}}` wird im **Abstract Syntax Tree** in Code 0.10 mithilfe der Komposition `Assign(Alloc(Writeable(), ArrayDecl([Num('2'), Num('1')], IntType('int')), Name('ar')), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]), Array([Num('4')])]))` dargestellt.

```
File
  Name './example_array_init.ast',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign(Alloc(Writeable(), ArrayDecl([Num('2'), Num('1')], IntType('int')),
        ↪  Name('ar')), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]),
        ↪  Array([Num('4')])]))
      ],
    FunDef
      VoidType 'void',
      Name 'fun',
      [],
      [
        Assign(Alloc(Writeable(), ArrayDecl([Num('2'), Num('2')], IntType('int')),
        ↪  Name('ar')), Array([Array([Num('3'), Num('4')]), Array([Num('5'), Num('6')])]))
      ]
  ]
```

Code 0.10: Abstract Syntax Tree für Array Initialisierung

Bei der **Initialisierung** eines **Arrays** wird zuerst `Alloc(Writeable(), ArrayDecl([Num('2'), Num('1')], IntType('int')))` ausgewertet, da eine Variable zuerst definiert sein muss, bevor man sie verwenden kann[2]. Das **Definieren** der Variable `ar` erfolgt mittels der **Symboltabelle**, die in Code 0.11 dargestellt ist.

Bei Variablen auf dem **Stackframe** wird ein Array **rückwärts** auf das Stackframe geschrieben und auch die **Adresse des ersten Elements** als Adresse des Arrays genommen. Dies macht den **Zugriff auf einen Arrayindex** in Subkapitel 0.0.2.2 deutlich unkomplizierter, da man so nicht mehr zwischen **Stackframe** und **Globalen Statischen Daten** beim **Zugriff auf einen Arrayindex** unterscheiden muss, da es Probleme macht, dass ein **Stackframe** in die entgegengesetzte Richtung wächst, verglichen mit den **Globalen**

[2] Das Widerspricht der üblichen Auswertungsreihenfolge beim **Zuweisungsoperator** `=`, der **rechtsassoziativ** ist. Der **Zuweisungsoperator** `=` tritt allerdings erst später in Aktion.

Statischen Daten[3].

Das **Größe** des Arrays `datatype ar`$[\mathtt{dim}_1]\dots[\mathtt{dim}_k]$, die ihm `size`-Feld des **Symboltabelleneintrags** eingetragen ist, berechnet sich dabei aus der **Mächtigkeit** der einzelnen **Dimensionen** des Arrays multipliziert mit der **Größe** des **grundlegenden Datentyps** der einzelnen **Arrayelemente**: $\mathtt{size}(\mathtt{datatype}(\mathtt{ar})) = \left(\prod_{j=1}^{n} \mathtt{dim}_j\right) \cdot \mathtt{size}(\mathtt{datatype})$[a].

[a] Die **Funktion** `type` ordnet einer **Variable** ihren **Datentyp** zu. Das ist notwendig, weil die **Funktion** `size` nur bei einem **Datentyp** als **Funktionsargument** die **Größe dieses Datentyps** als **Zielwert** liefert

```
SymbolTable
  [
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('main'), [])
        name:                   Name('main')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('5'))
        size:                   Empty()
      },
    Symbol
      {
        type qualifier:         Writeable()
        datatype:               ArrayDecl([Num('2'), Num('1')], IntType('int'))
        name:                   Name('ar@main')
        value or address:       Num('0')
        position:               Pos(Num('2'), Num('6'))
        size:                   Num('2')
      },
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('fun'), [])
        name:                   Name('fun')
        value or address:       Empty()
        position:               Pos(Num('5'), Num('5'))
        size:                   Empty()
      },
    Symbol
      {
        type qualifier:         Writeable()
        datatype:               ArrayDecl([Num('2'), Num('2')], IntType('int'))
        name:                   Name('ar@fun')
        value or address:       Num('3')
        position:               Pos(Num('6'), Num('6'))
        size:                   Num('4')
      }
  ]
```

Code 0.11: Symboltabelle für Array Initialisierung

[3] Wenn man beim **GCC** *GCC, the GNU Compiler Collection - GNU Project* einen Stackframe mittels des **GDB** *GCC, the GNU Compiler Collection - GNU Project* beobachtet, sieht man, dass dieser es genauso macht.

Im **PiocC-Mon Pass** in Code 0.12 werden zuerst die **Logischen Ausdrücke** in den Blättern des Teilbaums, der beim **Array-Initializers Container-Knoten** `Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]), Array([Num('4')])])` anfängt nach dem **Depth-First-Search** Schema, von **links-nach-rechts** ausgewertet und auf den **Stack** geschrieben[4].

Im finalen Schritt muss zwischen **Globalen Statischen Daten** bei der `main`-Funktion und **Stackframe** bei der Funktion `fun` unterschieden werden. Die auf den Stack ausgewerteten Expressions werden mittels der Komposition `Assign(Global(Num('0')), Stack(Num('2')))` bzw. `Assign(Stackframe(Num('3')), Stack(Num('4')))`, die in Tabelle **??** genauer beschrieben ist, versetzt in der selben Reihenfolge zu den **Globalen Statischen Daten** bzw. auf den **Stackframe** geschrieben.

Der **Trick** ist hier, dass egal wieviele Dimensionen und was für einen Datentyp das **Array** hat, man letztendlich immer das gesamte Array erwischt, wenn man einfach die **Größe des Arrays** viele **Speicherzellen** mit z.B. der **Komposition** `Assign(Global(Num('0')), Stack(Num('2')))` verschiebt.

In die Knoten `Global('0')` und `Stackframe('3')` wurde hierbei die **Startadresse** des jeweiligen Arrays geschrieben, sodass man nach dem **PicoC-Mon Pass** nie mehr Variablen in der **Symboltabelle** nachsehen muss und gleich weiß, ob sie in Bezug zu den **Globalen Statischen Daten** oder dem **Stackframe** stehen.

```
File
  Name './example_array_init.picoc_mon',
  [
    Block
      Name 'main.1',
      [
        // Assign(Name('ar'), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]),
        ↪  Array([Num('4')])]))
        Exp(Num('3'))
        Exp(Num('1'))
        Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
        Exp(Num('4'))
        Assign(Global(Num('0')), Stack(Num('2')))
        Return(Empty())
      ],
    Block
      Name 'fun.0',
      [
        // Assign(Name('ar'), Array([Array([Num('3'), Num('4')]), Array([Num('5'),
        ↪  Num('6')])]))
        Exp(Num('3'))
        Exp(Num('4'))
        Exp(Num('5'))
        Exp(Num('6'))
        Assign(Stackframe(Num('3')), Stack(Num('4')))
        Return(Empty())
      ]
  ]
```

Code 0.12: PicoC-Mon Pass für Array Initialisierung

Im **RETI-Blocks Pass** in Code 0.13 werden die **Kompositionen** `Exp(exp)` und `Assign(Global(Num('0'))`,

[4] Da der **Zuweisungsoperator** `=` **rechtsassoziativ** ist und auch rein **logisch**, weil man nichts zuweisen kann, was man noch nicht berechnet hat.

`Stack(Num('2'))` bzw. `Assign(Stackframe(Num('3')), Stack(Num('4')))` durch ihre entsprechenden **RETI-Knoten** ersetzt.

```
File
  Name './example_array_init.reti_blocks',
  [
    Block
      Name 'main.1',
      [
        # // Assign(Name('ar'), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]),
        ↪  Array([Num('4')])]))
        # Exp(Num('3'))
        SUBI SP 1;
        LOADI ACC 3;
        STOREIN SP ACC 1;
        # Exp(Num('1'))
        SUBI SP 1;
        LOADI ACC 1;
        STOREIN SP ACC 1;
        # Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
        LOADIN SP ACC 2;
        LOADIN SP IN2 1;
        ADD ACC IN2;
        STOREIN SP ACC 2;
        ADDI SP 1;
        # Exp(Num('4'))
        SUBI SP 1;
        LOADI ACC 4;
        STOREIN SP ACC 1;
        # Assign(Global(Num('0')), Stack(Num('2')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 1;
        LOADIN SP ACC 2;
        STOREIN DS ACC 0;
        ADDI SP 2;
        # Return(Empty())
        LOADIN BAF PC -1;
      ],
    Block
      Name 'fun.0',
      [
        # // Assign(Name('ar'), Array([Array([Num('3'), Num('4')]), Array([Num('5'),
        ↪  Num('6')])]))
        # Exp(Num('3'))
        SUBI SP 1;
        LOADI ACC 3;
        STOREIN SP ACC 1;
        # Exp(Num('4'))
        SUBI SP 1;
        LOADI ACC 4;
        STOREIN SP ACC 1;
        # Exp(Num('5'))
        SUBI SP 1;
        LOADI ACC 5;
        STOREIN SP ACC 1;
        # Exp(Num('6'))
```

```
52          SUBI SP 1;
53          LOADI ACC 6;
54          STOREIN SP ACC 1;
55          # Assign(Stackframe(Num('3')), Stack(Num('4')))
56          LOADIN SP ACC 1;
57          STOREIN BAF ACC -2;
58          LOADIN SP ACC 2;
59          STOREIN BAF ACC -3;
60          LOADIN SP ACC 3;
61          STOREIN BAF ACC -4;
62          LOADIN SP ACC 4;
63          STOREIN BAF ACC -5;
64          ADDI SP 4;
65          # Return(Empty())
66          LOADIN BAF PC -1;
67        ]
68    ]
```

Code 0.13: RETI-Blocks Pass für Array Initialisierung

0.0.2.2 Zugriff auf einen Arrayindex

Der **Zugriff auf einen Arrayindex** (z.B. `ar[0]`) wird im Folgenden anhand des Beispiels in Code 0.14 erklärt.

```
1 void main() {
2   int ar[1] = {42};
3   ar[0];
4 }
5
6 void fun() {
7   int ar[3] = {1, 2, 3};
8   ar[1+1];
9 }
```

Code 0.14: PicoC-Code für Zugriff auf einen Arrayindex

Der **Zugriff auf einen Arrayindex** `ar[0]` wird im **Abstract Syntax Tree** in Code 0.15 mithilfe des **Container-Knotens** `Subscr(Name('ar'), Num('0'))` dargestellt.

```
1 File
2   Name './example_array_access.ast',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Assign(Alloc(Writeable(), ArrayDecl([Num('1')], IntType('int')), Name('ar')),
          ↪  Array([Num('42')]))
10        Exp(Subscr(Name('ar'), Num('0')))
```

```
11      ],
12    FunDef
13      VoidType 'void',
14      Name 'fun',
15      [],
16      [
17        Assign(Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')), Name('ar')),
          ↪  Array([Num('1'), Num('2'), Num('3')]))
18        Exp(Subscr(Name('ar'), BinOp(Num('1'), Add('+'), Num('1'))))
19      ]
20  ]
```

Code 0.15: Abstract Syntax Tree für Zugriff auf einen Arrayindex

Im **PicoC-Mon Pass** in Code 0.16 wird vom **Container-Knoten** `Subscr(Name('ar'), Num('0'))` zuerst im **Anfangsteil** 0.0.4.1 die **Adresse** der Variable `Name('ar')` auf den **Stack** geschrieben. Bei den **Globalen Statischen Daten** der `main`-Funktion wird das durch die Komposition `Ref(Global(Num('0')))` dargestellt und beim **Stackframe** der Funktionm `fun` wird das durch die Komposition `Ref(Stackframe(Num('2')))` dargestellt.

In nächsten Schritt, dem **Mittelteil** 0.0.4.2 wird die Adresse des **Index**, des Arrays auf das Zugegriffen werden soll berechnet. Da der **Index** auf den Zugegriffen werden soll auch durch das Ergebnis eines **komplexeren Ausdrucks**, z.B. `ar[1 + var]` bestimmt sein kann, indem auch **Variablen** vorkommen können, kann dieser nicht während des **Kompilierens** berechnet werden, sondern muss zur **Laufzeit** berechnet werden.

Daher muss zuerst der Wert des **Index**, dessen Adresse berechnet werden soll bestimmt werden, z.B. im einfachen Fall durch `Exp(Num('0'))` und dann muss die **Adresse des Index** berechnet werden, was durch die Komposition `Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))` dargestellt wird. Die Bedeutung der Komposition `Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))` ist in Tabelle **??** dokumentiert.

Je nachdem, ob mehrere `Subscr(exp, exp)` eine Komposition bilden (z.B. `Subscr(Subscr(Name('var'), Num('1')), Num('1'))`) ist es notwendig mehrere **Adressberechnungsschritte für den Index** `Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))` einzuleiten und es muss auch möglich sein, z.B. einen **Attributzugriff** `var.attr` und eine **Zugriff auf einen Arryindex** `var[1]` miteinander zu kombinieren, was in Subkapitel 0.0.4.2 allgemein erklärt ist.

Im letzten Schritt, dem **Schlussteil** 0.0.4.3 wird der **Inhalt** des **Index**, dessen **Adresse** in den vorherigen Schritten berechnet wurde, nun auf den **Stack** geschrieben, wobei dieser die **Adresse** auf dem Stack ersetzt, die es zum Finden des **Index** brauchte. Dies wird durch den Knoten `Exp(Stack(Num('1')))` dargestellt. Je nachdem, welchen **Datentyp** die Variable `ar` hat und auf welchen **Subdatentyp**, welcher ein versteckter Attribut des `Exp(Stack(Num('1')))` Knoten ist folglich im **Kontext** zuletzt zugegriffen wird, abhängig davon wird der **Schlussteil** `Exp(Stack(Num('1')))` auf eine andere Weise verarbeitet (siehe Subkapitel 0.0.4.3).

Der einzige **Unterschied**, je nachdem, ob der **Zugriff auf einen Arrayindex** (z.B. `ar[1]`) in der `main`-Funktion oder der Funktion `fun` erfolgt, ist eigentlich nur beim **Anfangsteil**, beim Schreiben der **Adresse** der Variable `ar` auf den **Stack** zu finden, bei dem unterschiedliche **RETI-Instructions** für eine Variable, die in den **Globalen Statischen Daten** liegt und eine Variable, die auf dem **Stackframe** liegt erzeugt werden müssen.

```
1 File
2   Name './example_array_access.picoc_mon',
```

```
3  [
4    Block
5      Name 'main.1',
6      [
7        // Assign(Name('ar'), Array([Num('42')]))
8        Exp(Num('42'))
9        Assign(Global(Num('0')), Stack(Num('1')))
10       // Exp(Subscr(Name('ar'), Num('0')))
11       Ref(Global(Num('0')))
12       Exp(Num('0'))
13       Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
14       Exp(Stack(Num('1')))
15       Return(Empty())
16     ],
17   Block
18     Name 'fun.0',
19     [
20       // Assign(Name('ar'), Array([Num('1'), Num('2'), Num('3')]))
21       Exp(Num('1'))
22       Exp(Num('2'))
23       Exp(Num('3'))
24       Assign(Stackframe(Num('2')), Stack(Num('3')))
25       // Exp(Subscr(Name('ar'), BinOp(Num('1'), Add('+'), Num('1'))))
26       Ref(Stackframe(Num('2')))
27       Exp(Num('1'))
28       Exp(Num('1'))
29       Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
30       Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
31       Exp(Stack(Num('1')))
32       Return(Empty())
33     ]
34 ]
```

Code 0.16: PicoC-Mon Pass für Zugriff auf einen Arrayindex

Im **RETI-Blocks Pass** in Code 0.17 werden die **Kompositionen** `Ref(Global(Num('0')))`, `Ref(Subscr(Stack(Num('2')) und Stack(Num('1'))))` durch ihre entsprechenden **RETI-Knoten** ersetzt.

```
1 File
2   Name './example_array_access.reti_blocks',
3   [
4     Block
5       Name 'main.1',
6       [
7         # // Assign(Name('ar'), Array([Num('42')]))
8         # Exp(Num('42'))
9         SUBI SP 1;
10        LOADI ACC 42;
11        STOREIN SP ACC 1;
12        # Assign(Global(Num('0')), Stack(Num('1')))
13        LOADIN SP ACC 1;
14        STOREIN DS ACC 0;
15        ADDI SP 1;
```

```
16        # // Exp(Subscr(Name('ar'), Num('0')))
17        # Ref(Global(Num('0')))
18        SUBI SP 1;
19        LOADI IN1 0;
20        ADD IN1 DS;
21        STOREIN SP IN1 1;
22        # Exp(Num('0'))
23        SUBI SP 1;
24        LOADI ACC 0;
25        STOREIN SP ACC 1;
26        # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
27        LOADIN SP IN1 2;
28        LOADIN SP IN2 1;
29        MULTI IN2 1;
30        ADD IN1 IN2;
31        ADDI SP 1;
32        STOREIN SP IN1 1;
33        # Exp(Stack(Num('1')))
34        LOADIN SP IN1 1;
35        LOADIN IN1 ACC 0;
36        STOREIN SP ACC 1;
37        # Return(Empty())
38        LOADIN BAF PC -1;
39      ],
40    Block
41      Name 'fun.0',
42      [
43        # // Assign(Name('ar'), Array([Num('1'), Num('2'), Num('3')]))
44        # Exp(Num('1'))
45        SUBI SP 1;
46        LOADI ACC 1;
47        STOREIN SP ACC 1;
48        # Exp(Num('2'))
49        SUBI SP 1;
50        LOADI ACC 2;
51        STOREIN SP ACC 1;
52        # Exp(Num('3'))
53        SUBI SP 1;
54        LOADI ACC 3;
55        STOREIN SP ACC 1;
56        # Assign(Stackframe(Num('2')), Stack(Num('3')))
57        LOADIN SP ACC 1;
58        STOREIN BAF ACC -2;
59        LOADIN SP ACC 2;
60        STOREIN BAF ACC -3;
61        LOADIN SP ACC 3;
62        STOREIN BAF ACC -4;
63        ADDI SP 3;
64        # // Exp(Subscr(Name('ar'), BinOp(Num('1'), Add('+'), Num('1'))))
65        # Ref(Stackframe(Num('2')))
66        SUBI SP 1;
67        MOVE BAF IN1;
68        SUBI IN1 4;
69        STOREIN SP IN1 1;
70        # Exp(Num('1'))
71        SUBI SP 1;
72        LOADI ACC 1;
```

```
73        STOREIN SP ACC 1;
74        # Exp(Num('1'))
75        SUBI SP 1;
76        LOADI ACC 1;
77        STOREIN SP ACC 1;
78        # Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
79        LOADIN SP ACC 2;
80        LOADIN SP IN2 1;
81        ADD ACC IN2;
82        STOREIN SP ACC 2;
83        ADDI SP 1;
84        # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
85        LOADIN SP IN1 2;
86        LOADIN SP IN2 1;
87        MULTI IN2 1;
88        ADD IN1 IN2;
89        ADDI SP 1;
90        STOREIN SP IN1 1;
91        # Exp(Stack(Num('1')))
92        LOADIN SP IN1 1;
93        LOADIN IN1 ACC 0;
94        STOREIN SP ACC 1;
95        # Return(Empty())
96        LOADIN BAF PC -1;
97      ]
98  ]
```

Code 0.17: RETI-Blocks Pass für Zugriff auf einen Arrayindex

0.0.2.3 Zuweisung an Arrayindex

Die **Zuweisung** eines Wertes an einen **Arrayindex** (z.B. `ar[2] = 42;`) wird im Folgenden anhand des Beispiels in Code 0.18 erläutert.

```
1 void main() {
2   int ar[2];
3   ar[2] = 42;
4 }
```

Code 0.18: PicoC-Code für Zuweisung an Arrayindex

Im **Abstract Syntax Tree** in Code 0.19 wird eine **Zuweisung** an einen **Arrayindex** `ar[2] = 42;` durch die Komposition `Assign(Subscr(Name('ar'), Num('2')), Num('42'))` dargestellt.

```
1 File
2   Name './example_array_assignment.ast',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
```

```
8       [
9         Exp(Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar')))
10        Assign(Subscr(Name('ar'), Num('2')), Num('42'))
11      ]
12    ]
```

Code 0.19: Abstract Syntax Tree für Zuweisung an Arrayindex

Im **PicoC-Mon Pass** in Code 0.20 wird zuerst die **rechte** Seite des **rechtsassoziativen** Zuweisungsoperators `=`, bzw. des **Container-Knotens** der diesen darstellt ausgewertet: `Exp(Num('42'))`.

Danach ist das Vorgehen, bzw. sind die Kompostionen, die dieses darauffolgende Vorgehen darstellen: `Ref(Global(Num('0')))`, `Exp(Num('2'))` und `Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))` identisch zum **Anfangsteil** und **Mittelteil** aus dem vorherigen Subkapitel 0.0.2.2. Es wird die **Adresse** des **Index**, dem das Ergebnis der Ausdrucks auf der rechten Seite des **Zuweisungsoperators** `=` zugewiesen wird berechet, wie in Subkapitel 0.0.2.2.

Zum Schluss stellt die **Komposition** `Assign(Stack(Num('1')), Stack(Num('2')))`[5] die Zuweisung `=` des Ergebnisses des Ausdrucks auf der **rechten** Seite der Zuweisung zum **Arrayindex**, dessen **Adresse** im Schritt danach berechnet wurde dar.

Die Berechnung der **Adresse**, ab der ein **Arrayelement** eines Arrays $\texttt{datatype ar}[\texttt{dim}_1]\dots[\texttt{dim}_n]$ abgespeichert ist, kann mittels der Formel 0.0.1:

$$\texttt{ref}(\texttt{ar}[\texttt{idx}_1]\dots[\texttt{idx}_n]) = \texttt{ref}(\texttt{ar}) + \left(\sum_{i=1}^{n}\left(\prod_{j=i+1}^{n}\texttt{dim}_j\right)\cdot\texttt{idx}_i\right)\cdot\text{size}(\texttt{datatype}) \tag{0.0.1}$$

aus der Betriebssysteme Vorlesung[a] berechnet werden[b].

Die Kompositionen `Ref(Global(Num('0')))` und `Ref(Stackframe(Num('2')))` repräsentiert dabei den Summanden `ref(ar)` in der Formel.

Die Komposition `Exp(Num('2'))` repräsentiert dabei einen **Subindex** (z.B. `i` in `a[i][j][k]`) beim **Zugriff auf ein Arrayelement**, der als Faktor $\texttt{idx}_i$ in der Formel auftaucht.

Der Komposition `Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))` repräsentiert dabei einen ausmultiplizierten Summanden $\left(\prod_{j=i+1}^{n}\texttt{dim}_j\right)\cdot\texttt{idx}_i\cdot\texttt{size}(\texttt{datatpye})$ in der Formel.

Die Komposition `Exp(Stack(Num('1')))` repräsentiert dabei das Lesen des **Inhalts** $\texttt{M}[\texttt{ref}(\texttt{ar}[\texttt{idx}_1]\dots[\texttt{idx}_n])]$ der Speicherzelle an der finalen **Adresse** $\texttt{ref}(\texttt{ar}[\texttt{idx}_1]\dots[\texttt{idx}_n])$.

[a] Scholl, „Betriebssysteme“.
[b] `ref(exp)` steht dabei für die Berechnung der **Adresse** von `exp`, wobei `exp` z.B. `ar[3][2]` sein könnte

```
1 File
2   Name './example_array_assignment.picoc_mon',
3   [
4     Block
5       Name 'main.0',
```

[5] Ist in Tabelle ?? genauer beschrieben ist

```
6       [
7         // Exp(Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar')))
8         // Assign(Subscr(Name('ar'), Num('2')), Num('42'))
9         Exp(Num('42'))
10        Ref(Global(Num('0')))
11        Exp(Num('2'))
12        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
13        Assign(Stack(Num('1')), Stack(Num('2')))
14        Return(Empty())
15      ]
16    ]
```

Code 0.20: PicoC-Mon Pass für Zuweisung an Arrayindex

Im **RETI-Blocks Pass** in Code 0.21 werden die **Kompositionen** `Ref(Global(Num('0')))`, `Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))` und `Assign(Stack(Num('1')), Stack(Num('2')))` durch ihre entsprechenden **RETI-Knoten** ersetzt.

```
1 File
2   Name './example_array_assignment.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Exp(Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar')))
8         # // Assign(Subscr(Name('ar'), Num('2')), Num('42'))
9         # Exp(Num('42'))
10        SUBI SP 1;
11        LOADI ACC 42;
12        STOREIN SP ACC 1;
13        # Ref(Global(Num('0')))
14        SUBI SP 1;
15        LOADI IN1 0;
16        ADD IN1 DS;
17        STOREIN SP IN1 1;
18        # Exp(Num('2'))
19        SUBI SP 1;
20        LOADI ACC 2;
21        STOREIN SP ACC 1;
22        # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
23        LOADIN SP IN1 2;
24        LOADIN SP IN2 1;
25        MULTI IN2 1;
26        ADD IN1 IN2;
27        ADDI SP 1;
28        STOREIN SP IN1 1;
29        # Assign(Stack(Num('1')), Stack(Num('2')))
30        LOADIN SP IN1 1;
31        LOADIN SP ACC 2;
32        ADDI SP 2;
33        STOREIN IN1 ACC 0;
34        # Return(Empty())
35        LOADIN BAF PC -1;
36      ]
```

```
37  ]
```

Code 0.21: RETI-Blocks Pass für Zuweisung an Arrayindex

0.0.3 Umsetzung von Structs

0.0.3.1 Deklaration und Definition von Structtypen

Die **Deklaration** eines neuen **Structtyps** (z.B. `struct st {int len; int ar[2];};`) und die **Definition** einer Variable mit diesem **Structtyp** (z.B. `struct st st_var;`) wird im Folgenden anhand des Beispiels in Code 0.22 erläutert.

```
1 struct st {int len; int ar[2];};
2
3 void main() {
4   struct st st_var;
5 }
```

Code 0.22: PicoC-Code für die Deklaration eines Structtyps

Bevor irgendwas definiert werden kann, muss erstmal ein **Structtyp** deklariert werden. Im **Abstract Syntax Tree** in Code 0.24 wird die **Deklaration eines Structtyps** `struct st {int len; int ar[2];};` durch die Komposition `StructDecl(Name('st'), [Alloc(Writeable(), IntType('int'), Name('len')) Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar'))])` dargestellt.

Die **Definition** einer Variable mit diesem **Structtyp** `struct st st_var;` wird durch die Komposition `Alloc(Writeable(), StructSpec(Name('st')), Name('st_var'))` dargestellt.

```
1  File
2    Name './example_struct_decl_def.ast',
3    [
4      StructDecl
5        Name 'st',
6        [
7          Alloc(Writeable(), IntType('int'), Name('len'))
8          Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar'))
9        ],
10     FunDef
11       VoidType 'void',
12       Name 'main',
13       [],
14       [
15         Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('st_var')))
16       ]
17   ]
```

Code 0.23: Abstract Syntax Tree für die Deklaration eines Structtyps

Für den **Structtyp** selbst wird in der **Symboltabelle**, die in Code 0.24 dargestellt ist ein Eintrag mit dem **Schlüssel** `st` erstellt. Die Felder dieses Eintrags `type_qualifier`, `datatype`, `name`, `position` und `size` sind wie üblich belegt, allerdings sind in dem `value_address`-Feld die Attribute des **Structtyps** `[Name('len@st'), Name('ar@st')]` aufgelistet, sodass man über den **Structtyp** `st` die **Attribute** des Structtyps in der **Symboltabelle** nachschlagen kann. Die Schlüssel der **Attribute** haben einen **Suffix** `@st` angehängt, der eine Art **Scope** innerhalb des **Structtyps** für seine Attribut darstellt. Es gilt foglich, dass **innerhalb** eines **Structtyps** zwei Attribute nicht gleich benannt werden können, aber dafür zwei **unterschiedliche Structtypen** ihre Attribute gleich benennen können.

Jedes der **Attribute** `[Name('len@st'), Name('ar@st')]` erhält auch einen eigenen Eintrag in der **Symboltabelle**, wobei die Felder `type_qualifier`, `datatype`, `name`, `value_address`, `position` und `size` wie üblich belegt werden. Die Felder `type_qualifier`, `datatype` und `name` werden z.B. bei `Name('ar@st')` mithilfe der Attribute von `Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar'))])` belegt.

Für die **Definition** einer Variable `st_var@main` mit diesem **Structtyp** `st` wird ein Eintrag in der **Symboltabelle** angelegt. Das `datatyp`-Feld enhält dabei den Namen des **Structtyps** als Komposition `StructSpec(Name('st'))`, wodurch jederzeit alle wichtigen Informationen zu diesem **Structyp** und seinen **Attributen** in der **Symboltabelle** nachgeschlagen werden können.

> Die **Größe** einer Variable `st_var`, die ihm `size`-Feld des **Symboltabelleneintrags** eingetragen ist und mit dem **Structtyp** `struct st` $\{$`datatype`$_1$ `attr`$_1$; ... `datatype`$_n$ `attr`$_n$;$\}$;[a] definiert ist (`struct st st_var;`), berechnet sich dabei aus der Summe der **Größen** der einzelnen **Datentypen** `datatype`$_1$ ... `datatpye`$_n$ der **Attribute** `attr`$_1$, ... `attr`$_n$ des **Structtyps**: $\texttt{size(st)} = \sum_{i=1}^{n} \texttt{size(datatype}_i\texttt{)}$.
>
> [a] Hier wird es der Einfachheit halber so dargestellt, als hätte die Programmiersprache L_{PicoC} nicht die Fragwürdige Designentscheidung, auch die eckigen Klammern `[]` für die Definition eines Arrays **vor** die Variable zu schreiben von Lc übernommen. Es wird so getann, als würde der komplette **Datentyp** immer **hinter** der Variable stehen: `datatype var`.

```
1  SymbolTable
2    [
3      Symbol
4        {
5          type qualifier:          Empty()
6          datatype:                IntType('int')
7          name:                    Name('len@st')
8          value or address:        Empty()
9          position:                Pos(Num('1'), Num('15'))
10         size:                    Num('1')
11       },
12     Symbol
13       {
14         type qualifier:          Empty()
15         datatype:                ArrayDecl([Num('2')], IntType('int'))
16         name:                    Name('ar@st')
17         value or address:        Empty()
18         position:                Pos(Num('1'), Num('24'))
19         size:                    Num('2')
20       },
21     Symbol
22       {
23         type qualifier:          Empty()
```

```
24        datatype:               StructDecl(Name('st'), [Alloc(Writeable(), IntType('int'),
          ↪  Name('len'))Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')),
          ↪  Name('ar'))])
25        name:                   Name('st')
26        value or address:       [Name('len@st'), Name('ar@st')]
27        position:               Pos(Num('1'), Num('7'))
28        size:                   Num('3')
29      },
30    Symbol
31      {
32        type qualifier:         Empty()
33        datatype:               FunDecl(VoidType('void'), Name('main'), [])
34        name:                   Name('main')
35        value or address:       Empty()
36        position:               Pos(Num('3'), Num('5'))
37        size:                   Empty()
38      },
39    Symbol
40      {
41        type qualifier:         Writeable()
42        datatype:               StructSpec(Name('st'))
43        name:                   Name('st_var@main')
44        value or address:       Num('0')
45        position:               Pos(Num('4'), Num('12'))
46        size:                   Num('3')
47      }
48  ]
```

Code 0.24: Symboltabelle für die Deklaration eines Structtyps

0.0.3.2 Initialisierung von Structs

Die **Initialisierung eines Structs** wird im Folgenden mithilfe des Beispiels in Code 0.25 erklärt.

```
1 struct st1 {int *attr[2];};
2
3 struct st2 {int attr1; struct st1 attr2;};
4
5 void main() {
6   int var = 42;
7   struct st2 st = {.attr1=var, .attr2={.attr={{&var, &var}}}};
8 }
```

Code 0.25: PicoC-Code für Initialisierung von Structs

Im **Abstract Syntax Tree** in Code 0.26 wird die **Initialisierung eines Structs** `struct st1 st = {.attr1=var, .attr2={.attr={{&var, &var}}}};` mithilfe der **Komposition** `Assign(Alloc(Writeable(), StructSpec(Name('st1')), Name('st')), Struct(...))` dargestellt.

```
1 File
2   Name './example_struct_init.ast',
```

```
3  [
4    StructDecl
5      Name 'st1',
6      [
7        Alloc(Writeable(), ArrayDecl([Num('2')], PntrDecl(Num('1'), IntType('int'))),
         ↪  Name('attr'))
8      ],
9    StructDecl
10     Name 'st2',
11     [
12       Alloc(Writeable(), IntType('int'), Name('attr1'))
13       Alloc(Writeable(), StructSpec(Name('st1')), Name('attr2'))
14     ],
15   FunDef
16     VoidType 'void',
17     Name 'main',
18     [],
19     [
20       Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
21       Assign(Alloc(Writeable(), StructSpec(Name('st2')), Name('st')),
         ↪  Struct([Assign(Name('attr1'), Name('var')), Assign(Name('attr2'),
         ↪  Struct([Assign(Name('attr'), Array([Array([Ref(Name('var')),
         ↪  Ref(Name('var'))])])])])])
22     ]
23 ]
```

Code 0.26: Abstract Syntax Tree für Initialisierung von Structs

Im **PicoC-Mon Pass** in Code 0.27 wird die **Komposition** `Assign(Alloc(Writeable(), StructSpec(Name('st1')), Name('st')), Struct(...))` auf fast dieselbe Weise ausgewertet, wie bei der **Initialisierung eines Arrays** in Subkapitel 0.0.2.1 daher wird um keine Wiederholung zu betreiben auf Subkapitel 0.0.2.1 verwiesen. Um das ganze interressanter zu gestalten wurde das Beispiel in Code 0.25 so gewählt, dass sich daran eine komplexere, mehrstufige Initialisierung mit **verschiedenen** Datentypen erklären lässt.

Der **Struct-Initializer** Teilbaum `Struct([Assign(Name('attr1'), Name('var')), Assign(Name('attr2'), Struct([Assign(Name('attr'), Array([Array([Ref(Name('var')), Ref(Name('var'))])])])])`, der beim **Struct-Initializer Container-Knoten** anfängt, wird auf dieselbe Weise nach dem **Depth-First-Search** Prinzip von **links-nach-rechts** ausgewertet, wie es bei der **Initialisierung eines Arrays** in Subkapitel 0.0.2.1 bereits erklärt wurde.

Beim **Iterieren** über den **Teilbaum**, muss beim **Struct-Initializer** nur beachtet werden, dass bei den `Assign(lhs, exp)`-Knoten, über welche die **Attributzuweisung** dargestellt wird (z.B. `Assign(Name('attr2'), Struct([Assign(Name('attr'), Array([Array([Ref(Name('var')), Ref(Name('var'))])])])))`) der Teilbaum beim rechten `exp` Attribut weitergeht.

Im Allgemeinen gibt es beim **Initialisieren** eines **Arrays** oder **Structs** im Teilbaum auf der **rechten Seite**, der beim jeweiligen obersten **Initializer** anfängt immer nur 3 Fällte, man hat es auf der **rechten** Seite entweder mit einem **Struct-Initialiser**, einem **Array-Initialiser** oder einem **Logischen Ausdruck** zu tuen. Bei **Array-** und **Struct-Initialisier** wird einfach über diese nach dem **Depth-First-Search** Schema von **links-nach-rechts** iteriert und die Ergebnisse der **Logischen Ausdrücken** in den **Blättern** auf den **Stack** gespeichert. Der Fall, dass ein **Logischer Ausdruck** vorliegt erübrigt sich damit.

```
File
  Name './example_struct_init.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('var'), Num('42'))
        Exp(Num('42'))
        Assign(Global(Num('0')), Stack(Num('1')))
        // Assign(Name('st'), Struct([Assign(Name('attr1'), Name('var')),
        ↪  Assign(Name('attr2'), Struct([Assign(Name('attr'),
        ↪  Array([Array([Ref(Name('var')), Ref(Name('var'))])])])])])
        Exp(Global(Num('0')))
        Ref(Global(Num('0')))
        Ref(Global(Num('0')))
        Assign(Global(Num('1')), Stack(Num('3')))
        Return(Empty())
      ]
  ]
```

Code 0.27: PicoC-Mon Pass für Initialisierung von Structs

Im **RETI-Blocks Pass** in Code 0.28 werden die **Kompositionen** `Exp(exp)`, `Ref(exp)` und `Assign(Global(Num('1')), Stack(Num('3')))` durch ihre entsprechenden **RETI-Knoten** ersetzt.

```
File
  Name './example_struct_init.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('var'), Num('42'))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Assign(Global(Num('0')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 0;
        ADDI SP 1;
        # // Assign(Name('st'), Struct([Assign(Name('attr1'), Name('var')),
        ↪  Assign(Name('attr2'), Struct([Assign(Name('attr'),
        ↪  Array([Array([Ref(Name('var')), Ref(Name('var'))])])])])])
        # Exp(Global(Num('0')))
        SUBI SP 1;
        LOADIN DS ACC 0;
        STOREIN SP ACC 1;
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
        # Ref(Global(Num('0')))
        SUBI SP 1;
```

```
28        LOADI IN1 0;
29        ADD IN1 DS;
30        STOREIN SP IN1 1;
31        # Assign(Global(Num('1')), Stack(Num('3')))
32        LOADIN SP ACC 1;
33        STOREIN DS ACC 3;
34        LOADIN SP ACC 2;
35        STOREIN DS ACC 2;
36        LOADIN SP ACC 3;
37        STOREIN DS ACC 1;
38        ADDI SP 3;
39        # Return(Empty())
40        LOADIN BAF PC -1;
41      ]
42  ]
```

Code 0.28: RETI-Blocks Pass für Initialisierung von Structs

0.0.3.3 Zugriff auf Structattribut

Der **Zugriff auf ein Structattribut** wird im Folgenden mithilfe des Beispiels in Code 0.25 erklärt.

```
1 struct pos {int x; int y;};
2
3 void main() {
4   struct pos st = {.x=4, .y=2};
5   st.y;
6 }
```

Code 0.29: PicoC-Code für Zugriff auf Structattribut

```
1  File
2    Name './example_struct_attr_access.ast',
3    [
4      StructDecl
5        Name 'pos',
6        [
7          Alloc(Writeable(), IntType('int'), Name('x'))
8          Alloc(Writeable(), IntType('int'), Name('y'))
9        ],
10     FunDef
11       VoidType 'void',
12       Name 'main',
13       [],
14       [
15         Assign(Alloc(Writeable(), StructSpec(Name('pos')), Name('st')),
           ↪  Struct([Assign(Name('x'), Num('4')), Assign(Name('y'), Num('2'))]))
16         Exp(Attr(Name('st'), Name('y')))
17       ]
18   ]
```

Code 0.30: Abstract Syntax Tree für Zugriff auf Structattribut

```
File
  Name './example_struct_attr_access.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
        ↪  Num('2'))]))
        Exp(Num('4'))
        Exp(Num('2'))
        Assign(Global(Num('0')), Stack(Num('2')))
        // Exp(Attr(Name('st'), Name('y')))
        Ref(Global(Num('0')))
        Ref(Attr(Stack(Num('1')), Name('y')))
        Exp(Stack(Num('1')))
        Return(Empty())
      ]
  ]
```

Code 0.31: PicoC-Mon Pass für Zugriff auf Structattribut

```
File
  Name './example_struct_attr_access.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
        ↪  Num('2'))]))
        # Exp(Num('4'))
        SUBI SP 1;
        LOADI ACC 4;
        STOREIN SP ACC 1;
        # Exp(Num('2'))
        SUBI SP 1;
        LOADI ACC 2;
        STOREIN SP ACC 1;
        # Assign(Global(Num('0')), Stack(Num('2')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 1;
        LOADIN SP ACC 2;
        STOREIN DS ACC 0;
        ADDI SP 2;
        # // Exp(Attr(Name('st'), Name('y')))
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
        # Ref(Attr(Stack(Num('1')), Name('y')))
        LOADIN SP IN1 1;
```

```
30          ADDI IN1 1;
31          STOREIN SP IN1 1;
32          # Exp(Stack(Num('1')))
33          LOADIN SP IN1 1;
34          LOADIN IN1 ACC 0;
35          STOREIN SP ACC 1;
36          # Return(Empty())
37          LOADIN BAF PC -1;
38        ]
39    ]
```

Code 0.32: RETI-Blocks Pass für Zugriff auf Structattribut

0.0.3.4 Zuweisung an Structattribut

```
1 struct pos {int x; int y;};
2
3 void main() {
4   struct pos st = {.x=4, .y=2};
5   st.y = 42;
6 }
```

Code 0.33: PicoC-Code für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.ast',
3   [
4     StructDecl
5       Name 'pos',
6       [
7         Alloc(Writeable(), IntType('int'), Name('x'))
8         Alloc(Writeable(), IntType('int'), Name('y'))
9       ],
10    FunDef
11      VoidType 'void',
12      Name 'main',
13      [],
14      [
15        Assign(Alloc(Writeable(), StructSpec(Name('pos')), Name('st')),
          ↪  Struct([Assign(Name('x'), Num('4')), Assign(Name('y'), Num('2'))]))
16        Assign(Attr(Name('st'), Name('y')), Num('42'))
17      ]
18  ]
```

Code 0.34: Abstract Syntax Tree für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.picoc_mon',
```

```
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
        ↪  Num('2'))]))
        Exp(Num('4'))
        Exp(Num('2'))
        Assign(Global(Num('0')), Stack(Num('2')))
        // Assign(Attr(Name('st'), Name('y')), Num('42'))
        Exp(Num('42'))
        Ref(Global(Num('0')))
        Ref(Attr(Stack(Num('1')), Name('y')))
        Assign(Stack(Num('1')), Stack(Num('2')))
        Return(Empty())
      ]
  ]
```

Code 0.35: PicoC-Mon Pass für Zuweisung an Structattribut

```
File
  Name './example_struct_attr_assignment.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
        ↪  Num('2'))]))
        # Exp(Num('4'))
        SUBI SP 1;
        LOADI ACC 4;
        STOREIN SP ACC 1;
        # Exp(Num('2'))
        SUBI SP 1;
        LOADI ACC 2;
        STOREIN SP ACC 1;
        # Assign(Global(Num('0')), Stack(Num('2')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 1;
        LOADIN SP ACC 2;
        STOREIN DS ACC 0;
        ADDI SP 2;
        # // Assign(Attr(Name('st'), Name('y')), Num('42'))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
        # Ref(Attr(Stack(Num('1')), Name('y')))
        LOADIN SP IN1 1;
```

```
34          ADDI IN1 1;
35          STOREIN SP IN1 1;
36          # Assign(Stack(Num('1')), Stack(Num('2')))
37          LOADIN SP IN1 1;
38          LOADIN SP ACC 2;
39          ADDI SP 2;
40          STOREIN IN1 ACC 0;
41          # Return(Empty())
42          LOADIN BAF PC -1;
43        ]
44    ]
```

Code 0.36: RETI-Blocks Pass für Zuweisung an Structattribut

0.0.4 Umsetzung der Derived Datatypes im Zusammenspiel

0.0.4.1 Anfangsteil für Globale Statische Daten und Stackframe

```
1  struct ar_with_len {int len; int ar[2];};
2
3  void main() {
4    struct ar_with_len st_ar[3];
5    int *(*pntr2)[3];
6    pntr2;
7  }
8
9  void fun() {
10   struct ar_with_len st_ar[3];
11   int (*pntr1)[3];
12   pntr1;
13 }
```

Code 0.37: PicoC-Code für den Anfangsteil

```
1  File
2    Name './example_derived_dts_introduction_part.ast',
3    [
4      StructDecl
5        Name 'ar_with_len',
6        [
7          Alloc(Writeable(), IntType('int'), Name('len'))
8          Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar'))
9        ],
10     FunDef
11       VoidType 'void',
12       Name 'main',
13       [],
14       [
15         Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
           ↪  Name('st_ar')))
```

```
16        Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], PntrDecl(Num('1'),
          ↪  IntType('int')))), Name('pntr2')))
17        Exp(Name('pntr2'))
18      ],
19    FunDef
20      VoidType 'void',
21      Name 'fun',
22      [],
23      [
24        Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
25        Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
          ↪  Name('pntr1')))
26        Exp(Name('pntr1'))
27      ]
28  ]
```

Code 0.38: Abstract Syntax Tree für den Anfangsteil

```
1 File
2   Name './example_derived_dts_introduction_part.picoc_mon',
3   [
4     Block
5       Name 'main.1',
6       [
7         // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
8         // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], PntrDecl(Num('1'),
          ↪  IntType('int')))), Name('pntr2')))
9         // Exp(Name('pntr2'))
10        Exp(Global(Num('9')))
11        Return(Empty())
12      ],
13    Block
14      Name 'fun.0',
15      [
16        // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
17        // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
          ↪  Name('pntr1')))
18        // Exp(Name('pntr1'))
19        Exp(Stackframe(Num('9')))
20        Return(Empty())
21      ]
22  ]
```

Code 0.39: PicoC-Mon Pass für den Anfangsteil

```
1 File
2   Name './example_derived_dts_introduction_part.reti_blocks',
3   [
```

```
4     Block
5       Name 'main.1',
6       [
7         # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
8         # // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')],
          ↪  PntrDecl(Num('1'), IntType('int')))), Name('pntr2')))
9         # // Exp(Name('pntr2'))
10        # Exp(Global(Num('9')))
11        SUBI SP 1;
12        LOADIN DS ACC 9;
13        STOREIN SP ACC 1;
14        # Return(Empty())
15        LOADIN BAF PC -1;
16      ],
17    Block
18      Name 'fun.0',
19      [
20        # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
21        # // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')],
          ↪  IntType('int'))), Name('pntr1')))
22        # // Exp(Name('pntr1'))
23        # Exp(Stackframe(Num('9')))
24        SUBI SP 1;
25        LOADIN BAF ACC -11;
26        STOREIN SP ACC 1;
27        # Return(Empty())
28        LOADIN BAF PC -1;
29      ]
30  ]
```

Code 0.40: RETI-Blocks Pass für den Anfangsteil

0.0.4.2 Mittelteil für die verschiedenen Derived Datatypes

```
1 struct st1 {int (*ar)[1];};
2
3 void main() {
4   int var[1] = {42};
5   struct st1 st_first = {.ar=&var};
6   (*st_first.ar)[0];
7 }
```

Code 0.41: PicoC-Code für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.ast',
3   [
4     StructDecl
5       Name 'st1',
6       [
```

```
7         Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('1')], IntType('int'))),
          ↪  Name('ar'))
8       ],
9     FunDef
10      VoidType 'void',
11      Name 'main',
12      [],
13      [
14        Assign(Alloc(Writeable(), ArrayDecl([Num('1')], IntType('int')), Name('var')),
          ↪  Array([Num('42')]))
15        Assign(Alloc(Writeable(), StructSpec(Name('st1')), Name('st_first')),
          ↪  Struct([Assign(Name('ar'), Ref(Name('var')))]))
16        Exp(Subscr(Deref(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0')))
17      ]
18  ]
```

Code 0.42: Abstract Syntax Tree für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('var'), Array([Num('42')]))
8         Exp(Num('42'))
9         Assign(Global(Num('0')), Stack(Num('1')))
10        // Assign(Name('st_first'), Struct([Assign(Name('ar'), Ref(Name('var')))]))
11        Ref(Global(Num('0')))
12        Assign(Global(Num('1')), Stack(Num('1')))
13        // Exp(Subscr(Subscr(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0')))
14        Ref(Global(Num('1')))
15        Ref(Attr(Stack(Num('1')), Name('ar')))
16        Exp(Num('0'))
17        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
18        Exp(Num('0'))
19        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
20        Exp(Stack(Num('1')))
21        Return(Empty())
22      ]
23  ]
```

Code 0.43: PicoC-Mon Pass für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('var'), Array([Num('42')]))
```

```
 8         # Exp(Num('42'))
 9         SUBI SP 1;
10         LOADI ACC 42;
11         STOREIN SP ACC 1;
12         # Assign(Global(Num('0')), Stack(Num('1')))
13         LOADIN SP ACC 1;
14         STOREIN DS ACC 0;
15         ADDI SP 1;
16         # // Assign(Name('st_first'), Struct([Assign(Name('ar'), Ref(Name('var')))]))
17         # Ref(Global(Num('0')))
18         SUBI SP 1;
19         LOADI IN1 0;
20         ADD IN1 DS;
21         STOREIN SP IN1 1;
22         # Assign(Global(Num('1')), Stack(Num('1')))
23         LOADIN SP ACC 1;
24         STOREIN DS ACC 1;
25         ADDI SP 1;
26         # // Exp(Subscr(Subscr(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0')))
27         # Ref(Global(Num('1')))
28         SUBI SP 1;
29         LOADI IN1 1;
30         ADD IN1 DS;
31         STOREIN SP IN1 1;
32         # Ref(Attr(Stack(Num('1')), Name('ar')))
33         LOADIN SP IN1 1;
34         ADDI IN1 0;
35         STOREIN SP IN1 1;
36         # Exp(Num('0'))
37         SUBI SP 1;
38         LOADI ACC 0;
39         STOREIN SP ACC 1;
40         # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
41         LOADIN SP IN2 2;
42         LOADIN IN2 IN1 0;
43         LOADIN SP IN2 1;
44         MULTI IN2 1;
45         ADD IN1 IN2;
46         ADDI SP 1;
47         STOREIN SP IN1 1;
48         # Exp(Num('0'))
49         SUBI SP 1;
50         LOADI ACC 0;
51         STOREIN SP ACC 1;
52         # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
53         LOADIN SP IN1 2;
54         LOADIN SP IN2 1;
55         MULTI IN2 1;
56         ADD IN1 IN2;
57         ADDI SP 1;
58         STOREIN SP IN1 1;
59         # Exp(Stack(Num('1')))
60         LOADIN SP IN1 1;
61         LOADIN IN1 ACC 0;
62         STOREIN SP ACC 1;
63         # Return(Empty())
64         LOADIN BAF PC -1;
```

```
65      ]
66  ]
```

Code 0.44: RETI-Blocks Pass für den Mittelteil

0.0.4.3 Schlussteil für die verschiedenen Derived Datatypes

```
1  struct st {int attr[2];};
2
3  void main() {
4    int ar1[1][2] = {{42, 314}};
5    struct st ar2[1] = {.attr={42, 314}};
6    int var = 42;
7    int *pntr1 = &var;
8    int **pntr2 = &pntr1;
9
10   ar1[0];
11   ar2[0];
12   *pntr2;
13 }
```

Code 0.45: PicoC-Code für den Schlussteil

```
1  File
2    Name './example_derived_dts_final_part.ast',
3    [
4      StructDecl
5        Name 'st',
6        [
7          Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('attr'))
8        ],
9      FunDef
10       VoidType 'void',
11       Name 'main',
12       [],
13       [
14         Assign(Alloc(Writeable(), ArrayDecl([Num('1'), Num('2')], IntType('int')),
           ↪  Name('ar1')), Array([Array([Num('42'), Num('314')])]))
15         Assign(Alloc(Writeable(), ArrayDecl([Num('1')], StructSpec(Name('st'))),
           ↪  Name('ar2')), Struct([Assign(Name('attr'), Array([Num('42'), Num('314')]))]))
16         Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
17         Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr1')),
           ↪  Ref(Name('var')))
18         Assign(Alloc(Writeable(), PntrDecl(Num('2'), IntType('int')), Name('pntr2')),
           ↪  Ref(Name('pntr1')))
19         Exp(Subscr(Name('ar1'), Num('0')))
20         Exp(Subscr(Name('ar2'), Num('0')))
21         Exp(Deref(Name('pntr2'), Num('0')))
22       ]
23   ]
```

Code 0.46: Abstract Syntax Tree für den Schlussteil

```
File
  Name './example_derived_dts_final_part.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('ar1'), Array([Array([Num('42'), Num('314')])]))
        Exp(Num('42'))
        Exp(Num('314'))
        Assign(Global(Num('0')), Stack(Num('2')))
        // Assign(Name('ar2'), Struct([Assign(Name('attr'), Array([Num('42'),
        ↪  Num('314')]))]))
        Exp(Num('42'))
        Exp(Num('314'))
        Assign(Global(Num('2')), Stack(Num('2')))
        // Assign(Name('var'), Num('42'))
        Exp(Num('42'))
        Assign(Global(Num('4')), Stack(Num('1')))
        // Assign(Name('pntr1'), Ref(Name('var')))
        Ref(Global(Num('4')))
        Assign(Global(Num('5')), Stack(Num('1')))
        // Assign(Name('pntr2'), Ref(Name('pntr1')))
        Ref(Global(Num('5')))
        Assign(Global(Num('6')), Stack(Num('1')))
        // Exp(Subscr(Name('ar1'), Num('0')))
        Ref(Global(Num('0')))
        Exp(Num('0'))
        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
        Exp(Stack(Num('1')))
        // Exp(Subscr(Name('ar2'), Num('0')))
        Ref(Global(Num('2')))
        Exp(Num('0'))
        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
        Exp(Stack(Num('1')))
        // Exp(Subscr(Name('pntr2'), Num('0')))
        Ref(Global(Num('6')))
        Exp(Num('0'))
        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
        Exp(Stack(Num('1')))
        Return(Empty())
      ]
  ]
```

Code 0.47: PicoC-Mon Pass für den Schlussteil

```
File
  Name './example_derived_dts_final_part.reti_blocks',
  [
    Block
      Name 'main.0',
      [
```

```
 7         # // Assign(Name('ar1'), Array([Array([Num('42'), Num('314')])]))
 8         # Exp(Num('42'))
 9         SUBI SP 1;
10         LOADI ACC 42;
11         STOREIN SP ACC 1;
12         # Exp(Num('314'))
13         SUBI SP 1;
14         LOADI ACC 314;
15         STOREIN SP ACC 1;
16         # Assign(Global(Num('0')), Stack(Num('2')))
17         LOADIN SP ACC 1;
18         STOREIN DS ACC 1;
19         LOADIN SP ACC 2;
20         STOREIN DS ACC 0;
21         ADDI SP 2;
22         # // Assign(Name('ar2'), Struct([Assign(Name('attr'), Array([Num('42'),
           ↪  Num('314')]))]))
23         # Exp(Num('42'))
24         SUBI SP 1;
25         LOADI ACC 42;
26         STOREIN SP ACC 1;
27         # Exp(Num('314'))
28         SUBI SP 1;
29         LOADI ACC 314;
30         STOREIN SP ACC 1;
31         # Assign(Global(Num('2')), Stack(Num('2')))
32         LOADIN SP ACC 1;
33         STOREIN DS ACC 3;
34         LOADIN SP ACC 2;
35         STOREIN DS ACC 2;
36         ADDI SP 2;
37         # // Assign(Name('var'), Num('42'))
38         # Exp(Num('42'))
39         SUBI SP 1;
40         LOADI ACC 42;
41         STOREIN SP ACC 1;
42         # Assign(Global(Num('4')), Stack(Num('1')))
43         LOADIN SP ACC 1;
44         STOREIN DS ACC 4;
45         ADDI SP 1;
46         # // Assign(Name('pntr1'), Ref(Name('var')))
47         # Ref(Global(Num('4')))
48         SUBI SP 1;
49         LOADI IN1 4;
50         ADD IN1 DS;
51         STOREIN SP IN1 1;
52         # Assign(Global(Num('5')), Stack(Num('1')))
53         LOADIN SP ACC 1;
54         STOREIN DS ACC 5;
55         ADDI SP 1;
56         # // Assign(Name('pntr2'), Ref(Name('pntr1')))
57         # Ref(Global(Num('5')))
58         SUBI SP 1;
59         LOADI IN1 5;
60         ADD IN1 DS;
61         STOREIN SP IN1 1;
62         # Assign(Global(Num('6')), Stack(Num('1')))
```

```
63          LOADIN SP ACC 1;
64          STOREIN DS ACC 6;
65          ADDI SP 1;
66          # // Exp(Subscr(Name('ar1'), Num('0')))
67          # Ref(Global(Num('0')))
68          SUBI SP 1;
69          LOADI IN1 0;
70          ADD IN1 DS;
71          STOREIN SP IN1 1;
72          # Exp(Num('0'))
73          SUBI SP 1;
74          LOADI ACC 0;
75          STOREIN SP ACC 1;
76          # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
77          LOADIN SP IN1 2;
78          LOADIN SP IN2 1;
79          MULTI IN2 2;
80          ADD IN1 IN2;
81          ADDI SP 1;
82          STOREIN SP IN1 1;
83          # Exp(Stack(Num('1')))
84          # // Exp(Subscr(Name('ar2'), Num('0')))
85          # Ref(Global(Num('2')))
86          SUBI SP 1;
87          LOADI IN1 2;
88          ADD IN1 DS;
89          STOREIN SP IN1 1;
90          # Exp(Num('0'))
91          SUBI SP 1;
92          LOADI ACC 0;
93          STOREIN SP ACC 1;
94          # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
95          LOADIN SP IN1 2;
96          LOADIN SP IN2 1;
97          MULTI IN2 2;
98          ADD IN1 IN2;
99          ADDI SP 1;
100         STOREIN SP IN1 1;
101         # Exp(Stack(Num('1')))
102         LOADIN SP IN1 1;
103         LOADIN IN1 ACC 0;
104         STOREIN SP ACC 1;
105         # // Exp(Subscr(Name('pntr2'), Num('0')))
106         # Ref(Global(Num('6')))
107         SUBI SP 1;
108         LOADI IN1 6;
109         ADD IN1 DS;
110         STOREIN SP IN1 1;
111         # Exp(Num('0'))
112         SUBI SP 1;
113         LOADI ACC 0;
114         STOREIN SP ACC 1;
115         # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
116         LOADIN SP IN2 2;
117         LOADIN IN2 IN1 0;
118         LOADIN SP IN2 1;
119         MULTI IN2 1;
```

```
120         ADD IN1 IN2;
121         ADDI SP 1;
122         STOREIN SP IN1 1;
123         # Exp(Stack(Num('1')))
124         # Return(Empty())
125         LOADIN BAF PC -1;
126       ]
127   ]
```

Code 0.48: RETI-Blocks Pass für den Schlussteil

0.0.5 Umsetzung von Funktionen

0.0.5.1 Funktionen auflösen zu RETI Code

```
1  void main() {
2    return;
3  }
4
5  void fun1() {
6  }
7
8  int fun2() {
9    return 1;
10 }
```

Code 0.49: PicoC-Code für 3 Funktionen

```
1  File
2    Name './example_3_funs.ast',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Return(Empty())
10       ],
11     FunDef
12       VoidType 'void',
13       Name 'fun1',
14       [],
15       [],
16     FunDef
17       IntType 'int',
18       Name 'fun2',
19       [],
20       [
21         Return(Num('1'))
22       ]
23   ]
```

Code 0.50: Abstract Syntax Tree für 3 Funktionen

```
File
  Name './example_3_funs.picoc_blocks',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Block
          Name 'main.2',
          [
            Return(Empty())
          ]
      ],
    FunDef
      VoidType 'void',
      Name 'fun1',
      [],
      [
        Block
          Name 'fun1.1',
          []
      ],
    FunDef
      IntType 'int',
      Name 'fun2',
      [],
      [
        Block
          Name 'fun2.0',
          [
            Return(Num('1'))
          ]
      ]
  ]
```

Code 0.51: RETI-Blocks Pass für 3 Funktionen

```
File
  Name './example_3_funs.picoc_mon',
  [
    Block
      Name 'main.2',
      [
        Return(Empty())
      ],
    Block
      Name 'fun1.1',
      [
```

```
12        Return(Empty())
13      ],
14    Block
15      Name 'fun2.0',
16      [
17        // Return(Num('1'))
18        Exp(Num('1'))
19        Return(Stack(Num('1')))
20      ]
21  ]
```

Code 0.52: PicoC-Mon Pass für 3 Funktionen

```
1 File
2   Name './example_3_funs.reti_blocks',
3   [
4     Block
5       Name 'main.2',
6       [
7         # Return(Empty())
8         LOADIN BAF PC -1;
9       ],
10    Block
11      Name 'fun1.1',
12      [
13        # Return(Empty())
14        LOADIN BAF PC -1;
15      ],
16    Block
17      Name 'fun2.0',
18      [
19        # // Return(Num('1'))
20        # Exp(Num('1'))
21        SUBI SP 1;
22        LOADI ACC 1;
23        STOREIN SP ACC 1;
24        # Return(Stack(Num('1')))
25        LOADIN SP ACC 1;
26        ADDI SP 1;
27        LOADIN BAF PC -1;
28      ]
29  ]
```

Code 0.53: RETI-Blocks Pass für 3 Funktionen

0.0.5.1.1 Sprung zur Main Funktion

```
1 void fun1() {
2 }
3
4 int fun2() {
```

```
5   return 1;
6 }
7
8 void main() {
9   return;
10 }
```

Code 0.54: PicoC-Code für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
1 File
2   Name './example_3_funs_main.picoc_mon',
3   [
4     Block
5       Name 'fun1.2',
6       [
7         Return(Empty())
8       ],
9     Block
10      Name 'fun2.1',
11      [
12        // Return(Num('1'))
13        Exp(Num('1'))
14        Return(Stack(Num('1')))
15      ],
16    Block
17      Name 'main.0',
18      [
19        Return(Empty())
20      ]
21  ]
```

Code 0.55: PicoC-Mon Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
1 File
2   Name './example_3_funs_main.reti_blocks',
3   [
4     Block
5       Name 'fun1.2',
6       [
7         # Return(Empty())
8         LOADIN BAF PC -1;
9       ],
10    Block
11      Name 'fun2.1',
12      [
13        # // Return(Num('1'))
14        # Exp(Num('1'))
15        SUBI SP 1;
16        LOADI ACC 1;
17        STOREIN SP ACC 1;
18        # Return(Stack(Num('1')))
```

```
19        LOADIN SP ACC 1;
20        ADDI SP 1;
21        LOADIN BAF PC -1;
22      ],
23    Block
24      Name 'main.0',
25      [
26        # Return(Empty())
27        LOADIN BAF PC -1;
28      ]
29  ]
```

Code 0.56: RETI-Blocks Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
1 File
2   Name './example_3_funs_main.reti_patch',
3   [
4     Block
5       Name 'start.3',
6       [
7         # // Exp(GoTo(Name('main.0')))
8         Exp(GoTo(Name('main.0')))
9       ],
10    Block
11      Name 'fun1.2',
12      [
13        # Return(Empty())
14        LOADIN BAF PC -1;
15      ],
16    Block
17      Name 'fun2.1',
18      [
19        # // Return(Num('1'))
20        # Exp(Num('1'))
21        SUBI SP 1;
22        LOADI ACC 1;
23        STOREIN SP ACC 1;
24        # Return(Stack(Num('1')))
25        LOADIN SP ACC 1;
26        ADDI SP 1;
27        LOADIN BAF PC -1;
28      ],
29    Block
30      Name 'main.0',
31      [
32        # Return(Empty())
33        LOADIN BAF PC -1;
34      ]
35  ]
```

Code 0.57: PicoC-Patch Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

0.0.5.2 Funktionsdeklaration und -definition und Umsetzung von Scopes

```
1  int fun2(int var);
2
3  void fun1() {
4  }
5
6  void main() {
7    int var = fun2(42);
8    return;
9  }
10
11 int fun2(int var) {
12   return var;
13 }
```

Code 0.58: PicoC-Code für Funktionen, wobei eine Funktion vorher deklariert werden muss

Bei mehreren Funktionen werden die **Scopes** der unterschiedlichen **Funktionen** mittels eines **Suffix** `"<fun_name>@"` umgesetzt, der an den **Variablennamen** `<var>` drangehängt wird: `<var>@<fun_name>`. Dieser **Suffix** wird geändert sobald beim **Top-Down**[6] Durchiterieren über den **Abstract Syntax Tree** des aktuellen **Passes** nach dem **Depth-First-Search** Schema über den

```
SymbolTable
  [
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(IntType('int'), Name('fun2'), [Alloc(Writeable(),
        ↪  IntType('int'), Name('var'))])
        name:                   Name('fun2')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('4'))
        size:                   Empty()
      },
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('fun1'), [])
        name:                   Name('fun1')
        value or address:       Empty()
        position:               Pos(Num('3'), Num('5'))
        size:                   Empty()
      },
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('main'), [])
        name:                   Name('main')
        value or address:       Empty()
        position:               Pos(Num('6'), Num('5'))
        size:                   Empty()
```

[6] D.h. von der Wurzel zu den Blättern eines Baumes

```
29      },
30    Symbol
31      {
32        type qualifier:         Writeable()
33        datatype:               IntType('int')
34        name:                   Name('var@main')
35        value or address:       Num('0')
36        position:               Pos(Num('7'), Num('6'))
37        size:                   Num('1')
38      },
39    Symbol
40      {
41        type qualifier:         Writeable()
42        datatype:               IntType('int')
43        name:                   Name('var@fun2')
44        value or address:       Num('0')
45        position:               Pos(Num('11'), Num('13'))
46        size:                   Num('1')
47      }
48  ]
```

Code 0.59: Symboltabelle für Funktionen, wobei eine Funktion vorher deklariert werden muss

0.0.5.3 Funktionsaufruf

0.0.5.3.1 Ohne Rückgabewert

```
1  struct st {int attr1; int attr2[2];};
2
3  void stack_fun(struct st param[2][3]);
4
5  void main() {
6    struct st local_var[2][3];
7    stack_fun(local_var);
8    return;
9  }
10
11 void stack_fun(struct st param[2][3]) {
12   int local_var;
13 }
```

Code 0.60: PicoC-Code für Funktionsaufruf ohne Rückgabewert

```
1  File
2    Name './example_fun_call_no_return_value.picoc_mon',
3    [
4      Block
5        Name 'main.1',
6        [
7          // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], StructSpec(Name('st'))),
       ↪  Name('local_var')))
```

```
8         // Exp(Call(Name('stack_fun'), [Name('local_var')]))
9         StackMalloc(Num('2'))
10        Ref(Global(Num('0')))
11        NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
12        Exp(GoTo(Name('stack_fun.0')))
13        RemoveStackframe()
14        Return(Empty())
15      ],
16    Block
17      Name 'stack_fun.0',
18      [
19        // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('st'))),
          ↪  Name('param')))
20        // Exp(Alloc(Writeable(), IntType('int'), Name('local_var')))
21        Return(Empty())
22      ]
23  ]
```

Code 0.61: PicoC-Mon Pass für Funktionsaufruf ohne Rückgabewert

```
1 File
2   Name './example_fun_call_no_return_value.reti_blocks',
3   [
4     Block
5       Name 'main.1',
6       [
7         # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], StructSpec(Name('st'))),
          ↪  Name('local_var')))
8         # // Exp(Call(Name('stack_fun'), [Name('local_var')]))
9         # StackMalloc(Num('2'))
10        SUBI SP 2;
11        # Ref(Global(Num('0')))
12        SUBI SP 1;
13        LOADI IN1 0;
14        ADD IN1 DS;
15        STOREIN SP IN1 1;
16        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
17        MOVE BAF ACC;
18        ADDI SP 3;
19        MOVE SP BAF;
20        SUBI SP 4;
21        STOREIN BAF ACC 0;
22        LOADI ACC GoTo(Name('addr@next_instr'));
23        ADD ACC CS;
24        STOREIN BAF ACC -1;
25        # Exp(GoTo(Name('stack_fun.0')))
26        Exp(GoTo(Name('stack_fun.0')))
27        # RemoveStackframe()
28        MOVE BAF IN1;
29        LOADIN IN1 BAF 0;
30        MOVE IN1 SP;
31        # Return(Empty())
32        LOADIN BAF PC -1;
33      ],
```

```
34      Block
35        Name 'stack_fun.0',
36        [
37          # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('st'))),
            ↪  Name('param')))
38          # // Exp(Alloc(Writeable(), IntType('int'), Name('local_var')))
39          # Return(Empty())
40          LOADIN BAF PC -1;
41        ]
42    ]
```

Code 0.62: RETI-Blocks Pass für Funktionsaufruf ohne Rückgabewert

```
1  # // Exp(GoTo(Name('main.1')))
2  # // patched out Exp(GoTo(Name('main.1')))
3  # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], StructSpec(Name('st'))),
   ↪  Name('local_var')))
4  # // Exp(Call(Name('stack_fun'), [Name('local_var')]))
5  # StackMalloc(Num('2'))
6  SUBI SP 2;
7  # Ref(Global(Num('0')))
8  SUBI SP 1;
9  LOADI IN1 0;
10 ADD IN1 DS;
11 STOREIN SP IN1 1;
12 # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
13 MOVE BAF ACC;
14 ADDI SP 3;
15 MOVE SP BAF;
16 SUBI SP 4;
17 STOREIN BAF ACC 0;
18 LOADI ACC 14;
19 ADD ACC CS;
20 STOREIN BAF ACC -1;
21 # Exp(GoTo(Name('stack_fun.0')))
22 JUMP 5;
23 # RemoveStackframe()
24 MOVE BAF IN1;
25 LOADIN IN1 BAF 0;
26 MOVE IN1 SP;
27 # Return(Empty())
28 LOADIN BAF PC -1;
29 # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('st'))), Name('param')))
30 # // Exp(Alloc(Writeable(), IntType('int'), Name('local_var')))
31 # Return(Empty())
32 LOADIN BAF PC -1;
```

Code 0.63: RETI-Pass für Funktionsaufruf ohne Rückgabewert

0.0.5.3.2 Mit Rückgabewert

```
void stack_fun() {
  return 42;
}

void main() {
  int var = stack_fun();
}
```

Code 0.64: PicoC-Code für Funktionsaufruf mit Rückgabewert

```
File
  Name './example_fun_call_with_return_value.picoc_mon',
  [
    Block
      Name 'stack_fun.1',
      [
        // Return(Num('42'))
        Exp(Num('42'))
        Return(Stack(Num('1')))
      ],
    Block
      Name 'main.0',
      [
        // Assign(Name('var'), Call(Name('stack_fun'), []))
        StackMalloc(Num('2'))
        NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
        Exp(GoTo(Name('stack_fun.1')))
        RemoveStackframe()
        Assign(Global(Num('0')), Stack(Num('1')))
        Return(Empty())
      ]
  ]
```

Code 0.65: PicoC-Mon Pass für Funktionsaufruf mit Rückgabewert

```
File
  Name './example_fun_call_with_return_value.reti_blocks',
  [
    Block
      Name 'stack_fun.1',
      [
        # // Return(Num('42'))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Return(Stack(Num('1')))
        LOADIN SP ACC 1;
        ADDI SP 1;
        LOADIN BAF PC -1;
      ],
```

```
    Block
      Name 'main.0',
      [
        # // Assign(Name('var'), Call(Name('stack_fun'), []))
        # StackMalloc(Num('2'))
        SUBI SP 2;
        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
        MOVE BAF ACC;
        ADDI SP 2;
        MOVE SP BAF;
        SUBI SP 2;
        STOREIN BAF ACC 0;
        LOADI ACC GoTo(Name('addr@next_instr'));
        ADD ACC CS;
        STOREIN BAF ACC -1;
        # Exp(GoTo(Name('stack_fun.1')))
        Exp(GoTo(Name('stack_fun.1')))
        # RemoveStackframe()
        MOVE BAF IN1;
        LOADIN IN1 BAF 0;
        MOVE IN1 SP;
        # Assign(Global(Num('0')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 0;
        ADDI SP 1;
        # Return(Empty())
        LOADIN BAF PC -1;
      ]
  ]
```

Code 0.66: RETI-Blocks Pass für Funktionsaufruf mit Rückgabewert

```
# // Exp(GoTo(Name('main.0')))
JUMP 7;
# // Return(Num('42'))
# Exp(Num('42'))
SUBI SP 1;
LOADI ACC 42;
STOREIN SP ACC 1;
# Return(Stack(Num('1')))
LOADIN SP ACC 1;
ADDI SP 1;
LOADIN BAF PC -1;
# // Assign(Name('var'), Call(Name('stack_fun'), []))
# StackMalloc(Num('2'))
SUBI SP 2;
# NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
MOVE BAF ACC;
ADDI SP 2;
MOVE SP BAF;
SUBI SP 2;
STOREIN BAF ACC 0;
LOADI ACC 17;
ADD ACC CS;
```

```
STOREIN BAF ACC -1;
# Exp(GoTo(Name('stack_fun.1')))
JUMP -15;
# RemoveStackframe()
MOVE BAF IN1;
LOADIN IN1 BAF 0;
MOVE IN1 SP;
# Assign(Global(Num('0')), Stack(Num('1')))
LOADIN SP ACC 1;
STOREIN DS ACC 0;
ADDI SP 1;
# Return(Empty())
LOADIN BAF PC -1;
```

Code 0.67: RETI-Pass für Funktionsaufruf mit Rückgabewert

0.0.5.3.3 Umsetzung von Call by Sharing für Arrays

```
void stack_fun(int (*param1)[3], int param2[2][3]) {
}

void main() {
  int local_var1[2][3];
  int local_var2[2][3];
  stack_fun(local_var1, local_var2);
}
```

Code 0.68: PicoC-Code für Call by Sharing für Arrays

```
File
  Name './example_fun_call_by_sharing_array.picoc_mon',
  [
    Block
      Name 'stack_fun.1',
      [
        // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
        ↪  Name('param1')))
        // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')), Name('param2')))
        Return(Empty())
      ],
    Block
      Name 'main.0',
      [
        // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
        ↪  Name('local_var1')))
        // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
        ↪  Name('local_var2')))
        // Exp(Call(Name('stack_fun'), [Name('local_var1'), Name('local_var2')]))
        StackMalloc(Num('2'))
        Ref(Global(Num('0')))
        Ref(Global(Num('6')))
```

```
      NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
      Exp(GoTo(Name('stack_fun.1')))
      RemoveStackframe()
      Return(Empty())
    ]
]
```

Code 0.69: PicoC-Mon Pass für Call by Sharing für Arrays

```
SymbolTable
 [
   Symbol
     {
       type qualifier:         Empty()
       datatype:               FunDecl(VoidType('void'), Name('stack_fun'),
       ↪  [Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
       ↪  Name('param1')), Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')),
       ↪  Name('param2'))])
       name:                   Name('stack_fun')
       value or address:       Empty()
       position:               Pos(Num('1'), Num('5'))
       size:                   Empty()
     },
   Symbol
     {
       type qualifier:         Writeable()
       datatype:               PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
       name:                   Name('param1@stack_fun')
       value or address:       Num('0')
       position:               Pos(Num('1'), Num('21'))
       size:                   Num('1')
     },
   Symbol
     {
       type qualifier:         Writeable()
       datatype:               PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
       name:                   Name('param2@stack_fun')
       value or address:       Num('1')
       position:               Pos(Num('1'), Num('37'))
       size:                   Num('1')
     },
   Symbol
     {
       type qualifier:         Empty()
       datatype:               FunDecl(VoidType('void'), Name('main'), [])
       name:                   Name('main')
       value or address:       Empty()
       position:               Pos(Num('4'), Num('5'))
       size:                   Empty()
     },
   Symbol
     {
       type qualifier:         Writeable()
       datatype:               ArrayDecl([Num('2'), Num('3')], IntType('int'))
```

```
      name:                   Name('local_var1@main')
      value or address:       Num('0')
      position:               Pos(Num('5'), Num('6'))
      size:                   Num('6')
    },
  Symbol
    {
      type qualifier:         Writeable()
      datatype:               ArrayDecl([Num('2'), Num('3')], IntType('int'))
      name:                   Name('local_var2@main')
      value or address:       Num('6')
      position:               Pos(Num('6'), Num('6'))
      size:                   Num('6')
    }
]
```

Code 0.70: Symboltabelle für Call by Sharing für Arrays

```
File
 Name './example_fun_call_by_sharing_array.reti_blocks',
 [
   Block
     Name 'stack_fun.1',
     [
       # // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')],
       ↪  IntType('int'))), Name('param1')))
       # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')), Name('param2')))
       # Return(Empty())
       LOADIN BAF PC -1;
     ],
   Block
     Name 'main.0',
     [
       # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
       ↪  Name('local_var1')))
       # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
       ↪  Name('local_var2')))
       # // Exp(Call(Name('stack_fun'), [Name('local_var1'), Name('local_var2')]))
       # StackMalloc(Num('2'))
       SUBI SP 2;
       # Ref(Global(Num('0')))
       SUBI SP 1;
       LOADI IN1 0;
       ADD IN1 DS;
       STOREIN SP IN1 1;
       # Ref(Global(Num('6')))
       SUBI SP 1;
       LOADI IN1 6;
       ADD IN1 DS;
       STOREIN SP IN1 1;
       # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
       MOVE BAF ACC;
       ADDI SP 4;
       MOVE SP BAF;
```

```
34          SUBI SP 4;
35          STOREIN BAF ACC 0;
36          LOADI ACC GoTo(Name('addr@next_instr'));
37          ADD ACC CS;
38          STOREIN BAF ACC -1;
39          # Exp(GoTo(Name('stack_fun.1')))
40          Exp(GoTo(Name('stack_fun.1')))
41          # RemoveStackframe()
42          MOVE BAF IN1;
43          LOADIN IN1 BAF 0;
44          MOVE IN1 SP;
45          # Return(Empty())
46          LOADIN BAF PC -1;
47        ]
48    ]
```

Code 0.71: RETI-Block Pass für Call by Sharing für Arrays

0.0.5.3.4 Umsetzung von Call by Value für Structs

```
1  struct st {int attr1; int attr2[2];};
2
3  void stack_fun(struct st param) {
4  }
5
6  void main() {
7    struct st local_var;
8    stack_fun(local_var);
9  }
```

Code 0.72: PicoC-Code für Call by Value für Structs

```
1  File
2    Name './example_fun_call_by_value_struct.picoc_mon',
3    [
4      Block
5        Name 'stack_fun.1',
6        [
7          // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('param')))
8          Return(Empty())
9        ],
10     Block
11       Name 'main.0',
12       [
13         // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('local_var')))
14         // Exp(Call(Name('stack_fun'), [Name('local_var')]))
15         StackMalloc(Num('2'))
16         Assign(Stack(Num('3')), Global(Num('0')))
17         NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
18         Exp(GoTo(Name('stack_fun.1')))
19         RemoveStackframe()
```

```
20        Return(Empty())
21      ]
22  ]
```

Code 0.73: PicoC-Mon Pass für Call by Value für Structs

```
1 File
2   Name './example_fun_call_by_value_struct.reti_blocks',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
7         # // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('param')))
8         # Return(Empty())
9         LOADIN BAF PC -1;
10      ],
11    Block
12      Name 'main.0',
13      [
14        # // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('local_var')))
15        # // Exp(Call(Name('stack_fun'), [Name('local_var')]))
16        # StackMalloc(Num('2'))
17        SUBI SP 2;
18        # Assign(Stack(Num('3')), Global(Num('0')))
19        SUBI SP 3;
20        LOADIN DS ACC 0;
21        STOREIN SP ACC 1;
22        LOADIN DS ACC 1;
23        STOREIN SP ACC 2;
24        LOADIN DS ACC 2;
25        STOREIN SP ACC 3;
26        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
27        MOVE BAF ACC;
28        ADDI SP 5;
29        MOVE SP BAF;
30        SUBI SP 5;
31        STOREIN BAF ACC 0;
32        LOADI ACC GoTo(Name('addr@next_instr'));
33        ADD ACC CS;
34        STOREIN BAF ACC -1;
35        # Exp(GoTo(Name('stack_fun.1')))
36        Exp(GoTo(Name('stack_fun.1')))
37        # RemoveStackframe()
38        MOVE BAF IN1;
39        LOADIN IN1 BAF 0;
40        MOVE IN1 SP;
41        # Return(Empty())
42        LOADIN BAF PC -1;
43      ]
44  ]
```

Code 0.74: RETI-Block Pass für Call by Value für Structs

0.0.6 Umsetzung kleinerer Details

0.1 Fehlermeldungen

0.1.1 Error Handler

0.1.2 Arten von Fehlermeldungen

0.1.2.1 Syntaxfehler

0.1.2.2 Laufzeitfehler

Literatur

Online

- *GCC, the GNU Compiler Collection - GNU Project*. URL: https://gcc.gnu.org/ (besucht am 13. 07. 2022).

Vorlesungen

- Scholl, Christoph. „Betriebssysteme“. Vorlesung. Vorlesung. Universität Freiburg, 2020. URL: https://abs.informatik.uni-freiburg.de/src/teach_main.php?id=157 (besucht am 09. 07. 2022).