RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

MMIX – Crashkurs
Teil 3
Unterprogramme und Parameterübergabe

Freiling/Wienzek/Mink

Vorlesung Rechnerstrukturen

RWTH Aachen

Sommersemester 2005

Unterprogramme

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

Hauptproblem heutiger Softwareentwicklung liegt in der Bewältigung der zunehmenden Komplexität der entwickelten Systeme

- Trennung von Daten und Programm nur der erste Schritt
- zweiter Schritt: Strukturierung von Programm und Daten
- Unterprogramme sind das wesentliche technische Mittel zur Strukturierung von Programmen

Ein **Unterprogramm** ist eine Folge von Befehlen im Programm, die mit einem Namen versehen sind und unter Verwendung dieses Namens beliebig zur Ausführung gebracht werden können

- Befehlsfolge kann mehrfach durchlaufen werden
- Die im Unterprogramm zu verarbeitenden Daten werden als Parameter übergeben
- Die Form der Parameter ist in der Schnittstelle festgelegt
- Unterprogramme können Werte zurückliefern

Beispiele für Unterprogramme:

- Berechnung des ggT
- Berechnung des Maximums zweier Zahlen
- Umrechnen einer Zeichenkette in eine Dezimalzahl

Unterprogramme

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

Unterprogramme haben viele **Vorteile**:

- Strukturierung von Programmen, Verkürzung von Programmen, verteilte Softwareentwicklung, separate Übersetzung etc.
- Welche Funktionalität als Unterprogramm gekapselt wird, ist eine nichttriviale Frage

Grundregeln für den Entwurf von Unterprogrammen:

- Unterprogramme sollten nur eine einzige Aufgabe erledigen und die gut
- So allgemein wie möglich, so speziell wie nötig
- Daten möglichst ausschließlich über Parameter an das Unterprogramm übergeben (keine globalen Variablen verwenden)
- Ergebnisse über Rückgabewerte an das aufrufende Programm zurückgeben
- Seiteneffekte vermeiden (nur dann erzeugen, wenn der einzige Zweck des Unterprogramms im Seiteneffekt besteht)
  - Seiteneffekt: Veränderung im Speicher, die nicht durch die Schnittstelle beschrieben ist

Unterprogramme: Parameterübergabe

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

Es gibt verschiedene Möglichkeiten, Parameter an Unterprogramme zu übergeben

Arten der Wertübergabe:

- Werteübergabe (call by value): ein Zahlenwert wird direkt übergeben
- Referenzübergabe (call by reference): ein Zeiger auf den Zahlenwert wird übergeben, so können Werte ins aufrufende Programm zurückzugeben

Orte der Wertübergabe:

- Direkt über vereinbarte Register (einfach)
  - Hier kann man gut den Befehl `GO` verwenden
- Parameterübergabe über den Stapel (engl. Stack) (fortgeschritten)

Jetzt:

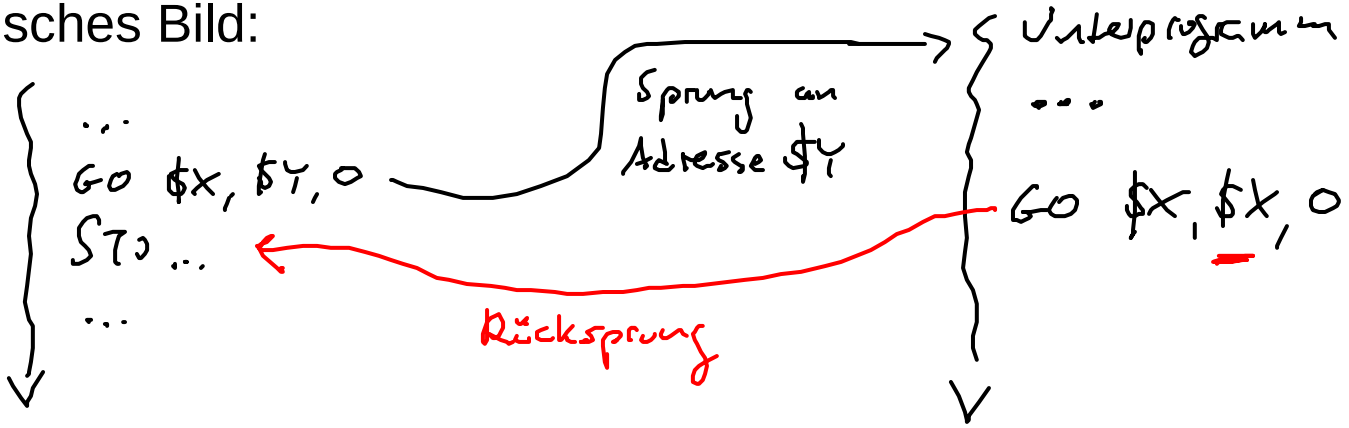
- Beispiel für Unterprogramme mit GO, Parameterübergabe über vereinbarte Register
- Beispiele für Werteübergabe und Referenzübergabe

Unterprogramme mit GO

GO kombiniert das Verzweigen zu einer Adresse mit dem Abspeichern der „Rücksprungadresse“ in einem Register

- Rücksprungadresse markiert, wo man eigentlich weitergemacht hätte
- `GO $X,$Y,Z` : Sprung an die Adresse $Y+Z, Adresse des unmittelbar folgenden Befehls wird in Register $X gespeichert

Schematisches Bild:



Parameter können über vereinbarte Register übergeben werden

- Beispiel: Unterprogramm PutChar gibt das in $1 übergebene Zeichen auf dem Bildschirm aus
- $0 enthält für die Dauer des Unterprogrammaufrufs die Rücksprungadresse

$X

Beispiel PutChar

```
          LOC     Data_Segment
          GREG    @
OutBuf    BYTE    0,0                 Ausgabepuffer für PutChar

          LOC     #100
          GREG    @
PutChar   LDA     $255,OutBuf         Adresse des Puffers nach $255
          STB     $1,$255,0           auszugebendes Zeichen in Puffer
          TRAP    0,Fputs,StdOut      Puffer ausgeben
          GO      $0,$0,0             Rücksprung

Newline   IS      #10

Main      SET     $1,Newline          Parameterübergabe by value
          GO      $0,PutChar          Unterprogrammaufruf
          TRAP    0,Halt,0
```

Beispiel: Call by Reference

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

```
..
Param   OCTA  3
        LOC   #100
* MMIX-Unterprogramm zur Erhöhung des Wertes einer
* Speicherzelle. Adresse der Speicherzelle wird in $1
* übergeben. $0 enthält die Rücksprungadresse. $2 wird
* als Hilfsregister verwendet
Incr    LDO   $2,$1,0   Wert aus dem Speicher laden
        ADD   $2,$2,1   um 1 erhöhen
        STO   $2,$1,0   Wert zurückschreiben
        GO    $0,$0,0
..
Main    LDA   $1,Param  Parameter laden
        GO    $0,Incr   Unterprogrammaufruf
```

Nachteile der Parameterübergabe mit Registern

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

Nachteile der Verwendung von expliziten Registern:

- Das Unterprogramm benötigt Register für lokale Größen - man muss aufpassen, dass das Unterprogramm keine Werte des Hauptprogramms überschreibt
- Für Parameter werden weitere Register benötigt
- Rekursion (Aufruf der Funktion aus sich heraus) wird nicht unterstützt

Abhilfe schafft ein Stapel (Stack)

- Stapel wird von MMIX direkt unterstützt (Registerstack)
- zur Verdeutlichung des Konzeptes wollen wir einen Stack "von Hand" implementieren

Stack

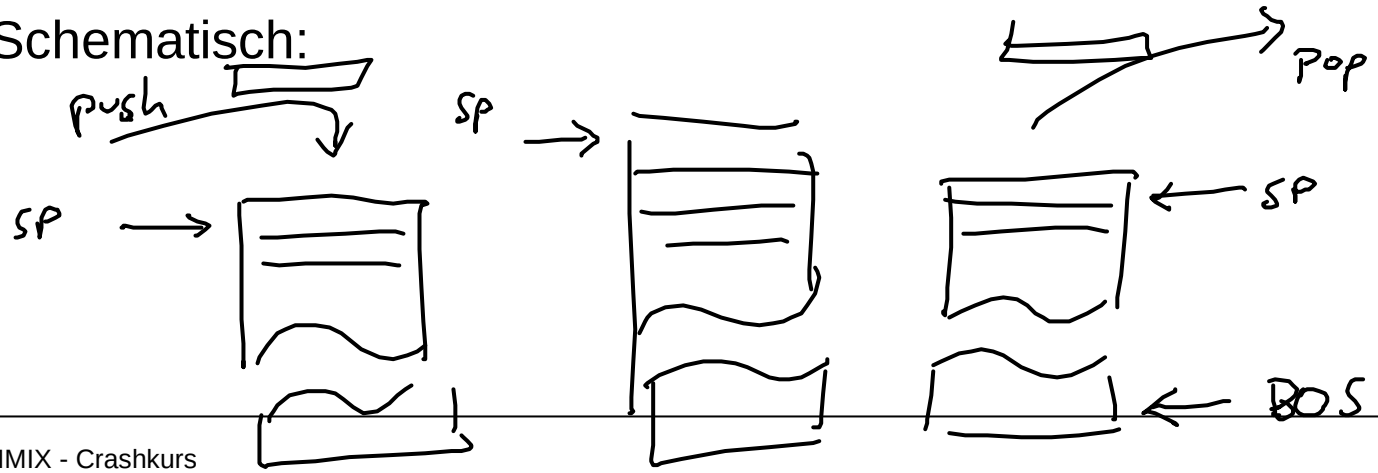
Stack = Stapel = Datenstruktur mit zwei Operationen

- Push : lege etwas auf den Stapel
- Pop : hole etwas vom Stapel
- Stapel gehorcht dem LIFO-Prinzip (Last In First Out)

Stack kann einfach implementiert werden durch einen eigenen Speicherbereich

- Zeiger SP (Stackpointer), meist gehalten in einem eigenen Register
- zeigt immer auf das Ende des Stacks
- Zu Beginn ist der Stack leer, SP steht am unteren Ende des Speicherbereichs
  - Bezeichnung BOS (Bottom of Stack)
- Platz zwischen Stackpointer und BOS gilt als belegt

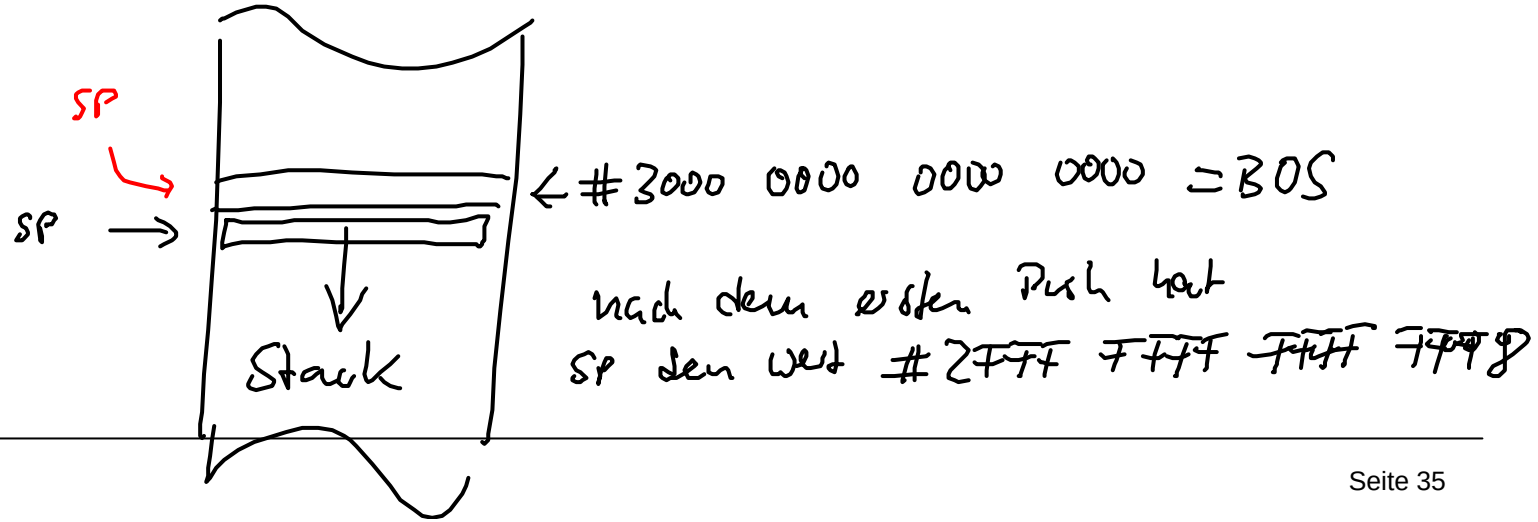
Schematisch:



Implementierung des Stack von Hand

Wir wollen beispielhaft einen Stack von Hand implementieren:

- Stackpointer SP wird in einem neuen globalen Register gehalten
- SP zeigt immer auf das zuletzt eingetragene Element (Top of Stack)
- Stack wächst von großen zu kleinen Adressen ("von oben nach unten")
- Dadurch können die Elemente des Stack leicht durch positive Offsets adressiert werden
- Gewählter Speicherbereich: von Adresse #2FFF FFFF FFFF FFFF an abwärts
- Stackpointer wird hinter den fiktiven ersten Eintrag gesetzt, also auf `BOS = #3000 0000 0000 0000` (leerer Stack)
- Wir werden Speicherplatz auf dem Stack immer in Vielfachen von 8 Byte bereitstellen

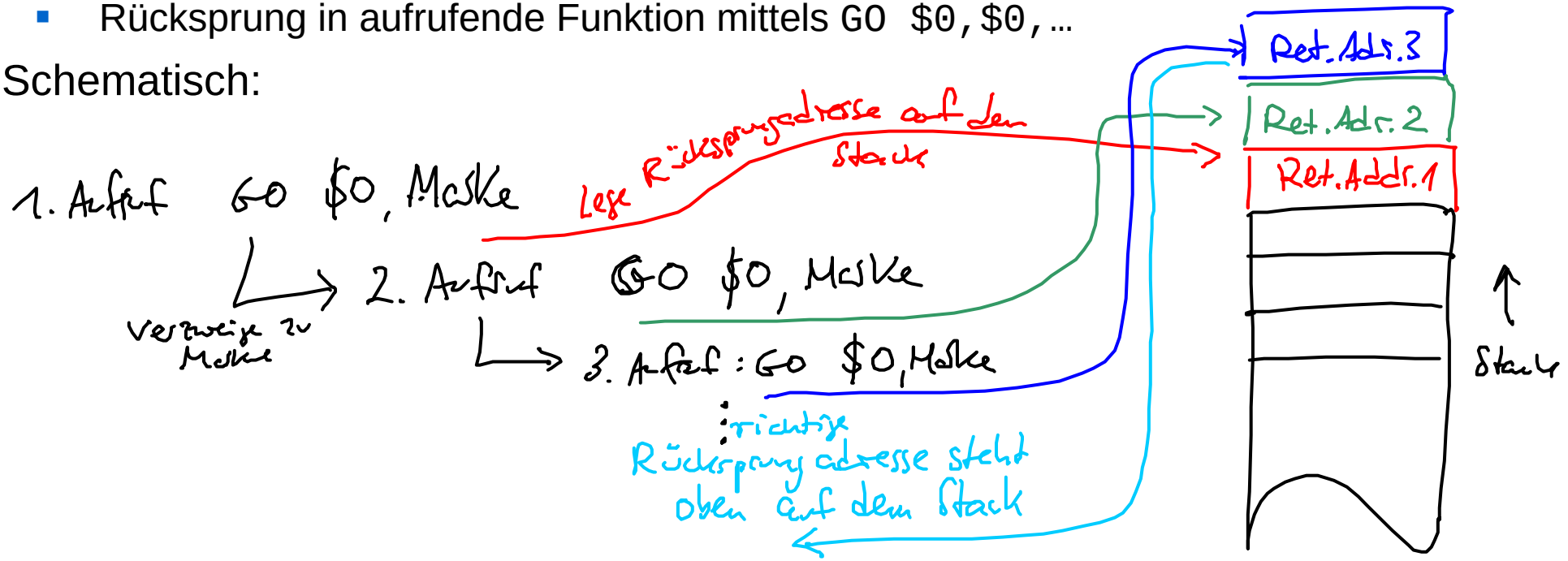


Ablage der Rücksprungadresse auf dem Stack

Verwende Befehl GO zusammen mit einem Stack:

- Rufe Unterprozedur mittels `GO $0,…` auf
- speichere $0 (Rücksprungadresse) auf dem Stack
- führe Prozedur aus (jetzt kann $0 beliebig verwendet werden)
- hole Rücksprungadresse vom Stack und speichere sie in $0
- Rücksprung in aufrufende Funktion mittels `GO $0,$0,…`

Schematisch:



Programmfragment

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

```
BOS     GREG    #3000000000000000   Bottom Of Stack
SP      GREG    0                   StackPointer
…
Doit    SUB     SP,SP,8             push return address
        STO     $0,SP,0
        …                           execute subroutine
                                    (may use $0 here)
        LDO     $0,SP,0             pop return address
        ADD     SP,SP,8
        GO      $0,$0,0             return to caller

Main    SET     SP,BOS              initialise stackpointer
        …
        GO      $0,Doit             call subroutine
```

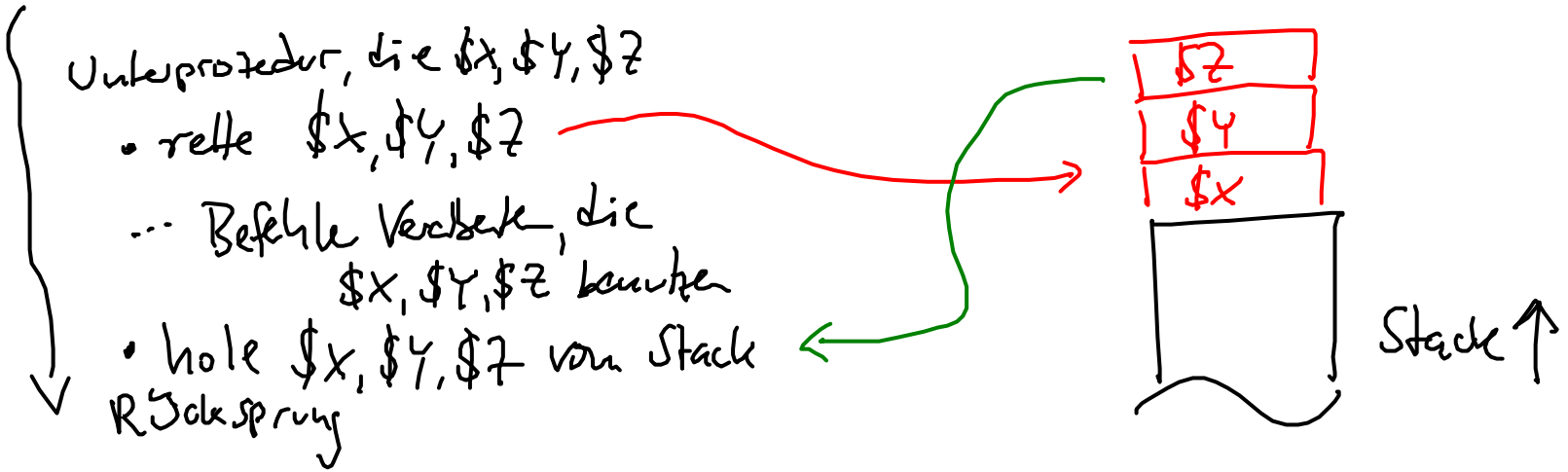
Retten von Registerinhalten auf den Stack

Interessante Beobachtung:

- Innerhalb der Unterprozedur darf das Register $0 ganz normal benutzt werden
- Wert, den es vorher hatte, wird auf dem Stack gerettet und am Ende wieder zurückgeholt
- Dieses Verfahren kann man allgemein für alle Register anwenden, die in der Unterprozedur benutzt werden

Schematisch:

- gegeben eine Unterprozedur, die intern Register $X, $Y und $Z benutzt



Seiteneffektfreie Unterprozeduren

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

Ein Seiteneffekt ist die Veränderung eines Wertes (in einem Register oder im Speicher), der nicht durch die Parameterliste ersichtlich ist.

- Beispiel: Unterprozedur erwartet Eingabeparameter in $1 und gibt Ergebnis in $2 zurück. Wenn innerhalb der Unterprozedur $3 verändert wird, hat die Unterprozedur einen Seiteneffekt

Retten aller benutzten Register auf den Stack sorgt für seiteneffektfreie Unterprozeduren

- Es wird die Regel implementiert: "Unterprozeduren dürfen keine Registerinhalte verändern!"
- Seiteneffektfreie Unterprozeduren sind potentiell "reentrant", d.h. sie können sich selbst aufrufen (Rekursion)
- Vorsicht: Rekursion kann unendlich fortschreiten, wenn man nicht aufpasst

Beispiel für eine rekursive Prozedur

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
RWTH

```
n       IS      $1
…
Doit    SUB     SP,SP,8         push RETA
        STO     $0,SP,0
        BZ      n,Ende          Rekursionsende ?
        SUB     n,n,1           n=n-1
        GO      $0,Doit         Rekursion
Ende    LDO     $0,SP,0         pop RETA
        ADD     SP,SP,8
        GO      $0,$0,0         Ruecksprung
Main    SET     SP,BOS          Initialisierung
        SET     n,7             Parameter n=7
        GO      $0,Doit
        TRAP    0,Halt,0
```

Ablauf der rekursiven Unterprozedur

```
1. 0000000000000120: c1fcfd00 (ORI) $252=g[252] = 3458764513820540928 =  #3000000000000000
1. 0000000000000124: e3010001 (SETL) $1=l[1] = #1
1. 0000000000000128: 9f00fb00 (GOI) $0=l[0] = #12c, -> #100
1. 0000000000000100: 25fcfc08 (SUBI) $252=g[252] = 3458764513820540928 - 8 = 3458764513820540920
1. 0000000000000104: ad00fc00 (STOI) M8[#2fffffffffffff8] = 300
1. 0000000000000108: 42010003 (BZ) 1==0?  No
1. 000000000000010c: 25010101 (SUBI) $1=l[1] = 1 - 1 = 0
1. 0000000000000110: 9f00fb00 (GOI) $0=l[0] = #114, -> #100
2. 0000000000000100: 25fcfc08 (SUBI) $252=g[252] = 3458764513820540920 - 8 = 3458764513820540912
2. 0000000000000104: ad00fc00 (STOI) M8[#2ffffffffffff0] = 276
2. 0000000000000108: 42010003 (BZ) 0==0?  Yes, -> #114 (bad guess)
1. 0000000000000114: 8d00fc00 (LDOI) $0=l[0] = M8[#2ffffffffffff0] = 276
1. 0000000000000118: 21fcfc08 (ADDI) $252=g[252] = 3458764513820540912 + 8 = 3458764513820540920
1. 000000000000011c: 9f000000 (GOI) $0=l[0] = #120, -> #114
2. 0000000000000114: 8d00fc00 (LDOI) $0=l[0] = M8[#2fffffffffffff8] = 300
2. 0000000000000118: 21fcfc08 (ADDI) $252=g[252] = 3458764513820540920 + 8 = 3458764513820540928
2. 000000000000011c: 9f000000 (GOI) $0=l[0] = #120, -> #12c
1. 000000000000012c: 00000000 (TRAP) Halt(0)
  18 instructions, 4 mems, 32 oops; 1 good guess, 1 bad
  (halted at location #000000000000012c)
```

n = 1

erster rekursiver Aufruf

push auf den Stack

Abfrage

n := n - 1

zweiter rekursiver Aufruf

push

Abfrage => Abbruch

pop

Rücksprung

pop

finale Rücksprung

Was passiert bei Eingabe einer negativen Zahl?

Parameter auf dem Stack

Statt Parameter in Registern zu übergeben, kann man die Parameter auch auf den Stack legen

- Das aufrufende Programm legt die Parameter vereinbarungsgemäß auf den Stack
- Das aufgerufene Programm holt die Parameter vom Stack, verarbeitet sie und legt das Ergebnis wieder auf den Stack
- Das aufrufende Programm holt sich das Ergebnis vom Stack

Schematisch:

Programmbeispiel: Berechnung der Fakultät

fak(n) = 1 für n=1, n * fak(n-1) sonst

```
            LOC     Data_Segment
            GREG    @
n           IS      $1
f           IS      $2
test        IS      $3

BOS         GREG    #3000000000000000
SP          GREG    0           Stackpointer

            LOC     #100
            GREG    @

Fak         SUB     SP,SP,8     push
            STO     $0,SP,0     return addr.
            LDO     n,SP,8      get param.
            CMP     test,n,1    Ende Rek.?
            BP      test,Rek    ggf Aufruf
            SET     f,1         f=1
            JMP     Ende
            …
Rek         SUB     n,n,1       n=n-1
            SUB     SP,SP,8     push
            STO     n,SP,0
            GO      $0,Fak
            LDO     f,SP,0
            ADD     SP,SP,8     pop
            LDO     n,SP,8      alter Wert n
            MUL     f,f,n       f = f*n
            STO     f,SP,8      Erg. auf Stack
Ende        LDO     $0,SP,0
            ADD     SP,SP,8     pop
            GO      $0,$0,0     Ruecksprung

            …
Main        SET     SP,BOS
            SET     n,7             n=7
            SUB     SP,SP,8         push
            STO     n,SP,0
            GO      $0,Fak      Aufruf
            LDO     f,SP,0
            ADD     SP,SP,8         pop
            TRAP    0,Halt,0
```

Stackframes

Stackframes organisieren die Verwendung des Stacks

- Compiler generieren üblicherweise Code nach einem derartigen Muster
- Stackframe = Parameter und lokale Variable für jeweils einen Aufruf einer Prozedur

Gliederung eines Stackframes: