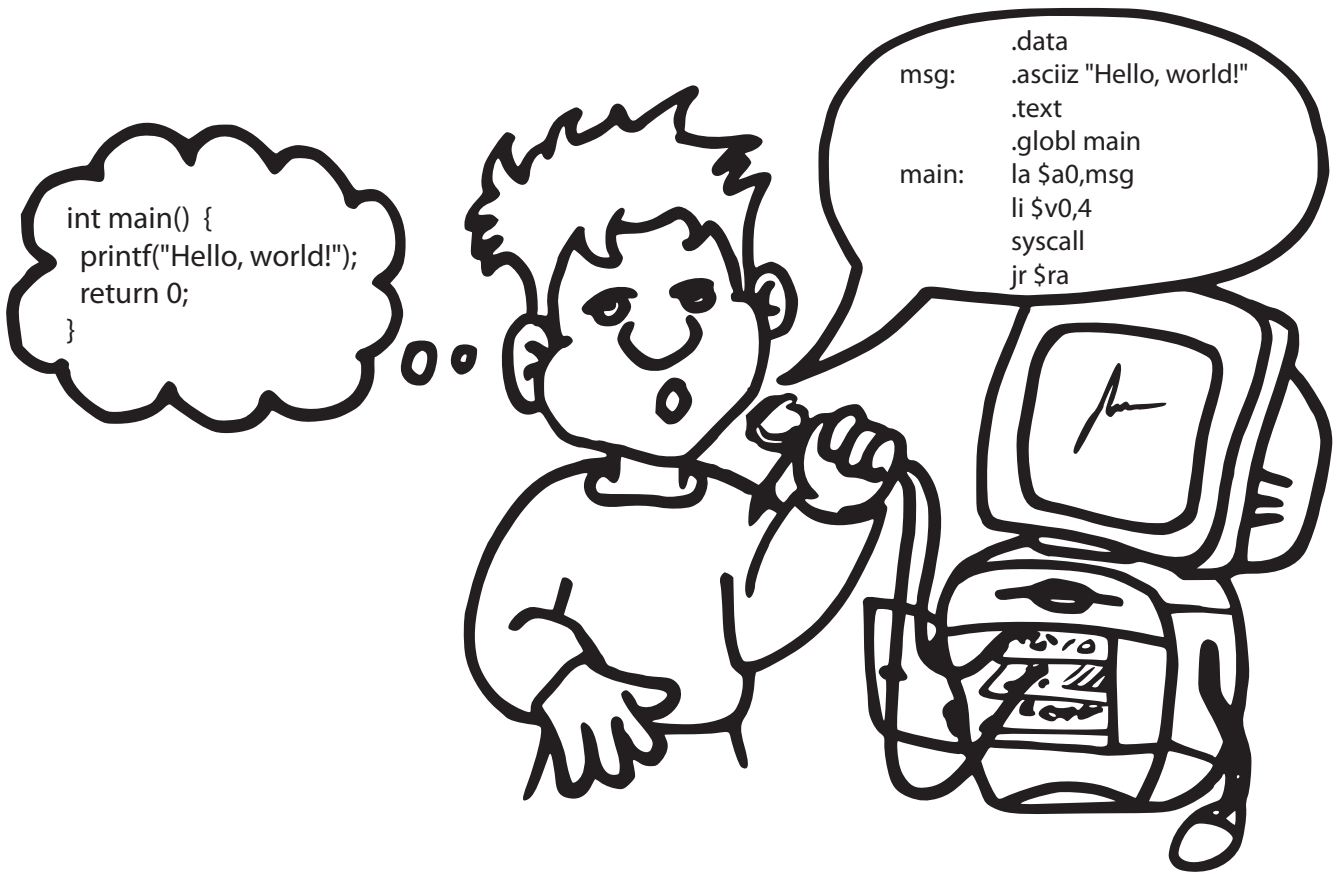
Grundlagen der Informatik III
Wintersemester 2010/2011 – 8. Vorlesung
Dr.-Ing. Wolfgang Heenes

TECHNISCHE UNIVERSITÄT DARMSTADT



Inhalt

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Literatur
2. Rekursionen
3. Programmausführungszeit
4. Bottom-Up/Top-Down
5. Zusammenfassung und Ausblick

Literatur

TECHNISCHE UNIVERSITÄT DARMSTADT

[BO10] Bryant, Randal E. und David R. O´Hallaron: *Computer Systems - A Programmer´s Perspective*.
Prentice Hall, 2010.

[DBG08] Dausmann, Manfred, Ulrich Bröckl und Joachim Goll: *C als erste Programmiersprache*.
Teubner, 2008.

Rekursionen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Rekursive Unterprogramme (Funktionen, Prozeduren, Methodenaufrufe) sind eine Besonderheit von Unterprogrammen, weil man hier im Rumpf des Unterprogramms einen Aufruf desselben Unterprogramms findet.
- Implementierung in Assembler: Im Prinzip alles wie in einer Hochsprache, allerdings deutlich mehr *Schreibarbeit* und Verwendung von **call** und **ret**
- Und spätestens jetzt kommt das Verständnis, wo berechnete Zwischenergebnisse der Rekursion abgelegt werden.
- Beispiel (angelehnt an eine Syntax in C, vgl. [DBG08, S. 233])

Rekursionen
Fakultätsberechnung

TECHNISCHE UNIVERSITÄT DARMSTADT

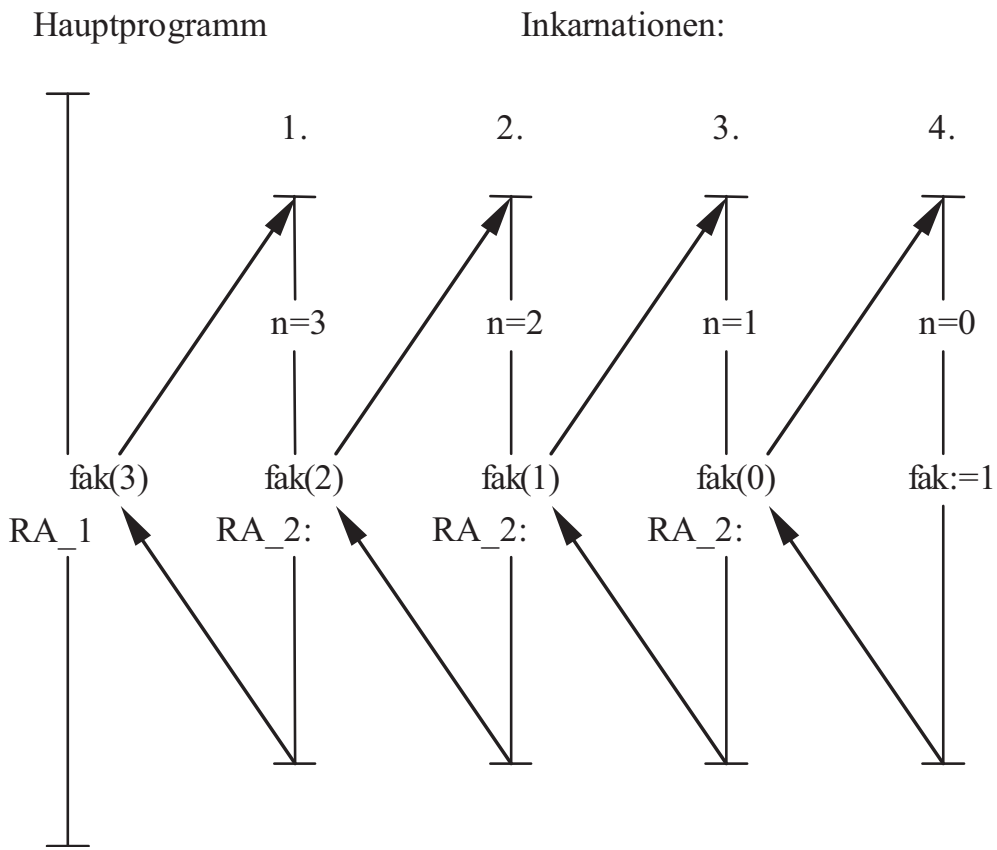
```
int fak (int n)
{ if (n>=1) {
 return fak (n−1) * n;
}
else {
return 1;
}}

int main (void)
{
int n = 5;
int z;
z = fak (n);
printf("Fakultät %d\n:", z);
return 0;
}
```

Rekursionen
Fakultätsberechnung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vor der Umsetzung erstmal einige Überlegungen, Ablauf eines Unterprogramms wird auch als Inkarnation (wiederholter Unterprogrammaufruf) bezeichnet



Rekursionen
Fakultätsberechnung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Abbildung der Inkarnationen zeigt, dass es zwei verschiedene Rückkehradressen gibt
- Die Adresse RA_1 ist die Adresse im Hauptprogramm.
- Die Adresse RA_2 ist die Adresse im Unterprogramm.
- Für alle Inkarnationen des Unterprogramms fak ist die Rückkehradresse RA_2 dieselbe Adresse, da alle Inkarnationen Abläufe desselben Codes sind.
- Im Folgenden: Wiederholung, Phasen eines Unterprogrammaufrufs, Aufbau des Stacks für Fakultät

Phasen eines Unterprogrammaufrufes

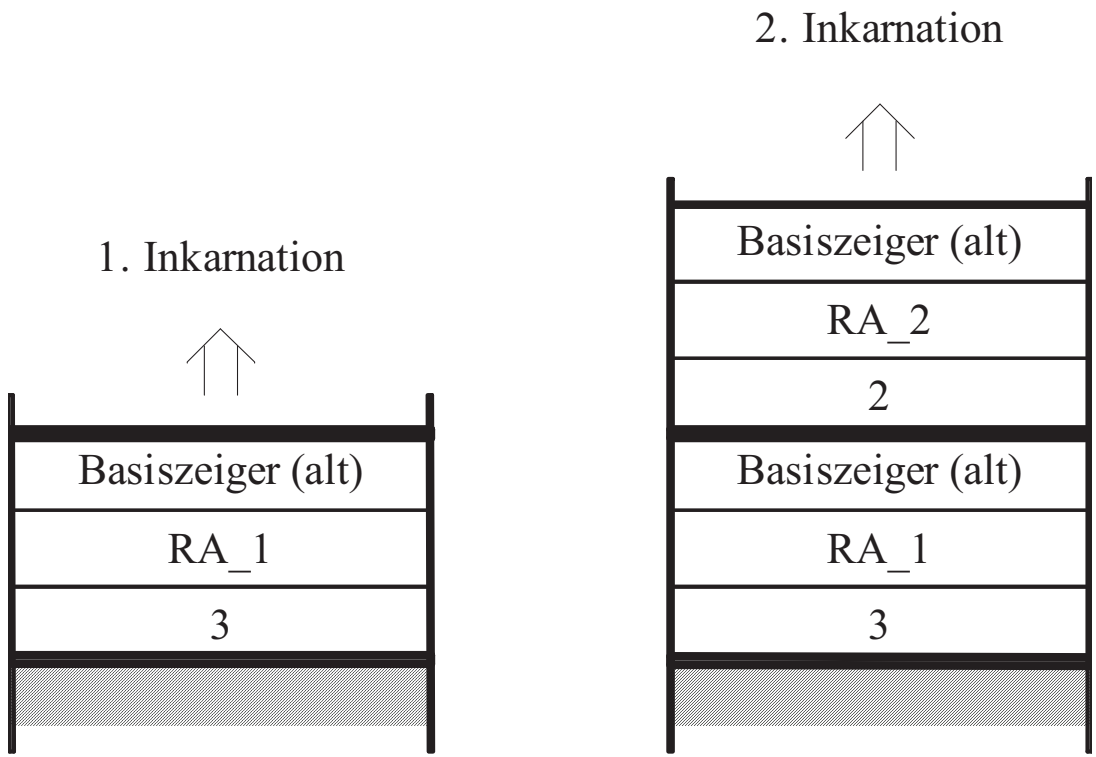
TECHNISCHE UNIVERSITÄT DARMSTADT

- Ablage der Parameter auf dem Stapel
- Aufruf des Unterprogramms mit speziellem Befehl, automatische Ablage der Rückkehradresse auf den Stapel
- Sicherstellung des Basiszeigers (pushl %ebp)
- Neueinstellung des Basiszeigers (zur Adressierung der Parameter und lokalen Variablen) (movl %esp,%ebp)
- Reservierung von Platz auf dem Stack für lokale Variablen
- Sicherstellung der Inhalte der übrigen Register
- Ausführen des Unterprogramms
- Rückstellen der geretten Registerinhalte
- Freigabe des Speichers für lokale Variablen
- Rückkehr zum aufrufendem Programm (ret)
- Freigabe des Stacks für Parameter

Rekursionen
Aufbau des Stacks I

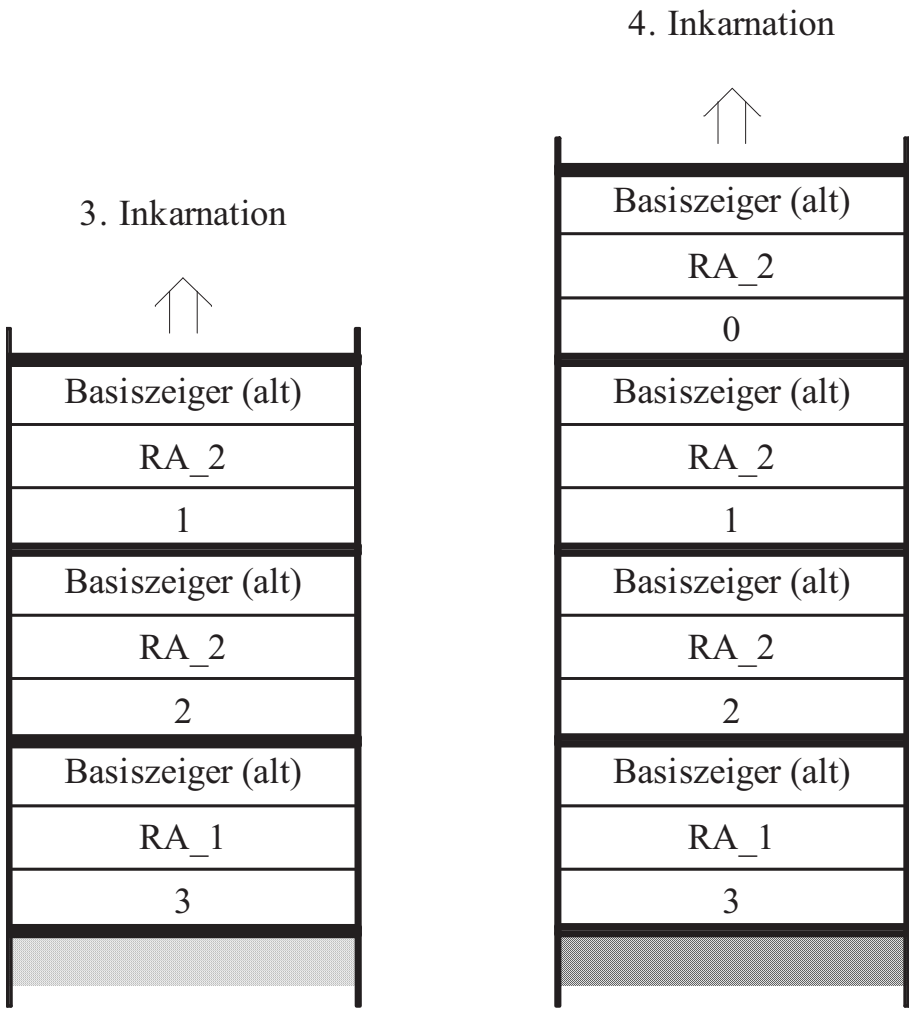
TECHNISCHE UNIVERSITÄT DARMSTADT

- Für die Fakultät von 3



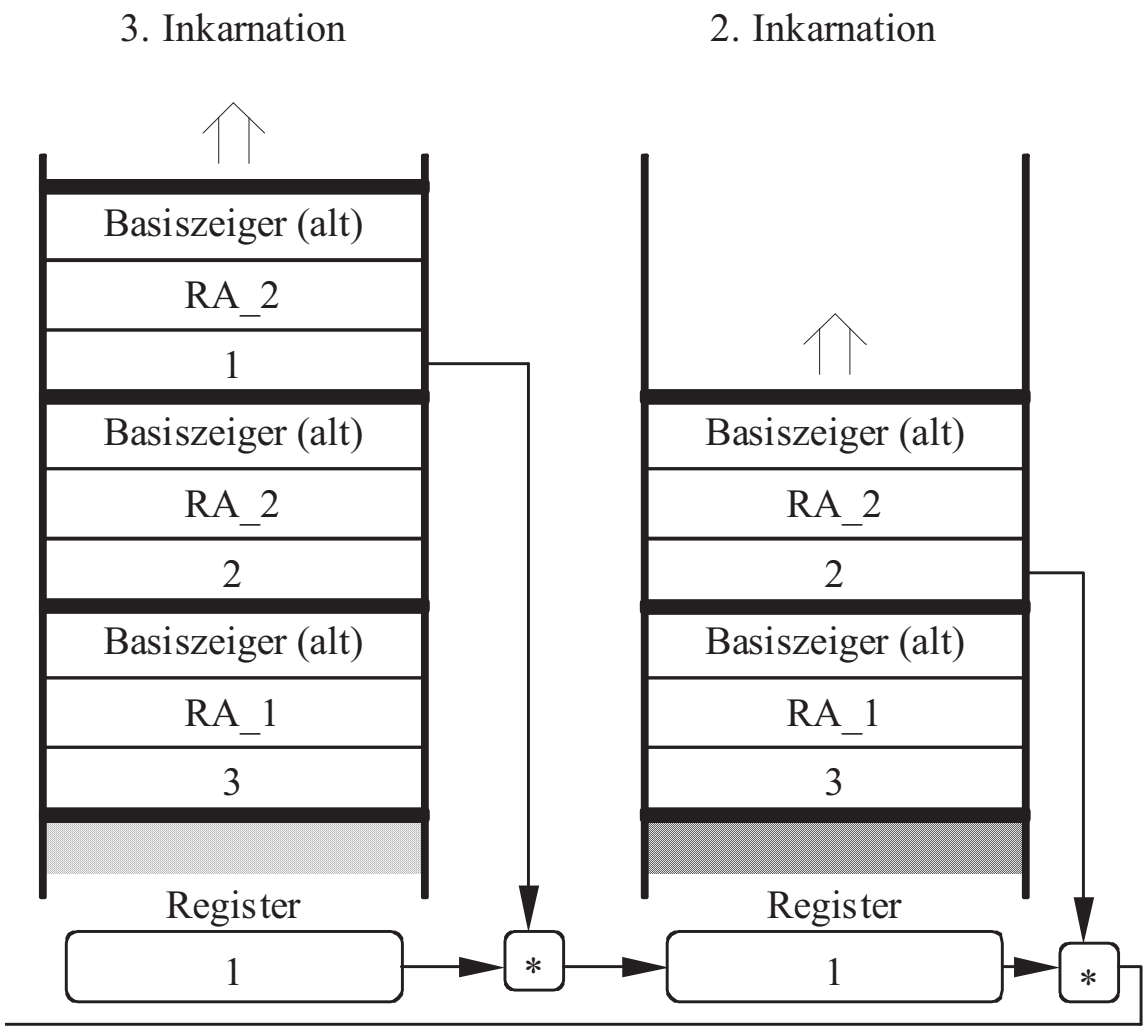
Rekursionen
Aufbau des Stacks II

TECHNISCHE UNIVERSITÄT DARMSTADT



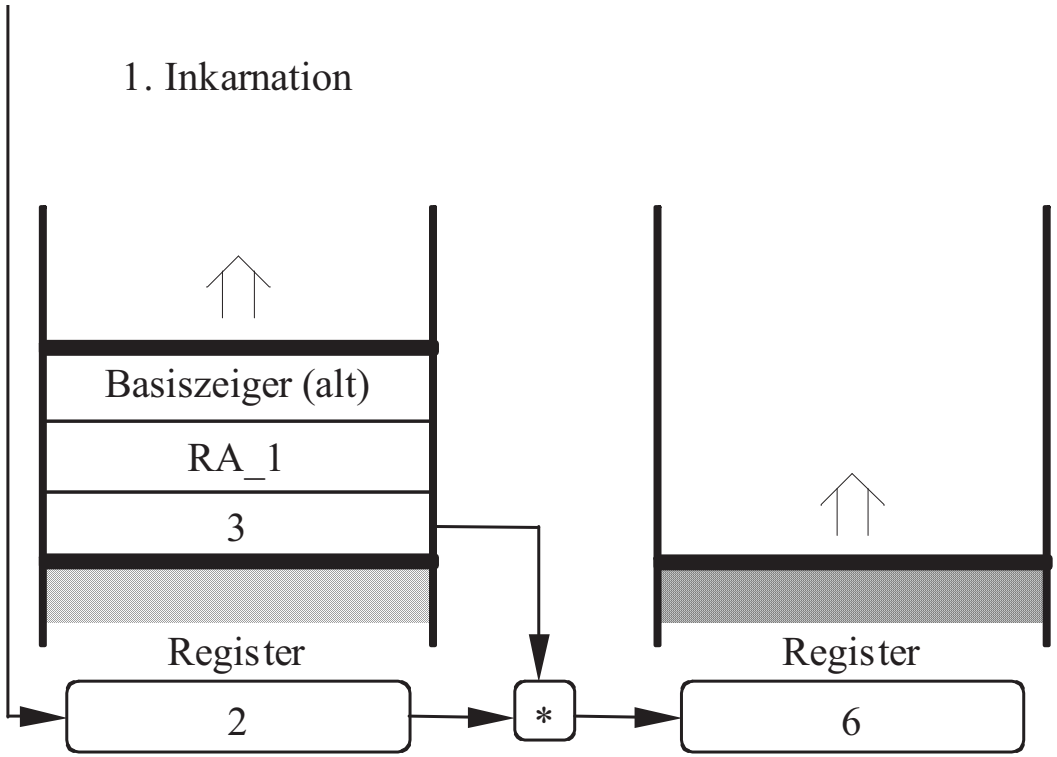
Rekursionen
Aufbau des Stacks III

TECHNISCHE UNIVERSITÄT DARMSTADT



Rekursionen
Aufbau des Stacks IV

TECHNISCHE UNIVERSITÄT DARMSTADT



Funktionen und Unterprogramme
Fakultät: Hauptprogramm

TECHNISCHE UNIVERSITÄT DARMSTADT

```
.data
 intout: .string "Ergebnis %d \n"
 n:      .long  3
 z:      .long  0
...
main:
        pushl n    # Wert auf Stack
        call .fak # Aufruf Unterprogramm
RA_1:   addl $4,%esp # Parameter vergessen
        movl %eax,z
```

Funktionen und Unterprogramme
Fakultät: Unterprogramm

TECHNISCHE UNIVERSITÄT DARMSTADT

```
...
.fak:  pushl %ebp # Base-Pointer sichern
       movl %esp,%ebp # Stack -> Base
       movl 8(%ebp),%eax # Holen von n
       cmpl $0,%eax # vergleich auf 0
       jnz else
       movl $1,%eax # Rekursionsanker
       jmp fin
else:  decl %eax # n um eins verringern
       pushl %eax
       call .fak
RA_2:  addl $4,%esp # Parameter vergessen
       imull 8(%ebp)
fin:   popl %ebp
       ret  # Rücksprung
...
```

Zusammenfassung Rekursionen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Rekursionen sind besondere Unterprogramme
- Parameterübergaben erfolgen über den Stack
- Die Befehle **call** und **ret** unterstützen die Umsetzung von Unterprogrammen
- Bei Rekursionen ist insbesondere die Rückstellung des Stackpointers (%ebp) essentiell
- Aufgrund der vielen Stackzugriffe, sind rekursive Implementierungen, z. B. Fakultät langsamer als iterative Implementierungen
- Spezielle Algorithmen haben allerdings deutliche Vorteile bzgl. des Laufzeitverhaltens
- $\Rightarrow$ 2. Praktikum

Programmausführungszeit

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verschiedene Möglichkeiten der Messung
- Stoppuhr (nur bei sehr langen Laufzeiten)
- Die Rechnersysteme (Hardware und Betriebssystem) stellen dafür verschiedene Möglichkeiten zur Verfügung
- Beispiel Linux: time
  - real
  - CPU user
  - CPU sys
- time fak
  - Für n = 3 $\Rightarrow$ 0m0.002 (real) der Rest ist nicht zu messen
  - Für n = 12 $\Rightarrow$ 0m0.002 (real) der Rest ist nicht zu messen oder falsch...
- Programmteil in einer Schleife häufiger ausführen
- Mehrere Durchgänge aufnehmen, evt. Mittelwerte bilden (vgl. 11. Vorlesung)

Programmausführungszeit Windows

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auf der Kommandozeile (cmd) folgende Befehle bzw. folgendes Skript ausführen

```
1 @echo off
2
3 echo Starte Programm %time%
4
5 Programm
6
7 echo End Programm %time%
```

- Genauigkeit beschränkt

Programmausführungszeit

TECHNISCHE UNIVERSITÄT DARMSTADT

- Es gibt auch sogenannte Tickzähler
- **rdtsc**: schreibt die Anzahl der Ticks, die zwischen dem Start des Rechners und dem Aufruf der Instruktion vergangen sind, in das Register %edx:%eax.
- In der Regel laufen die Programme so kurz, dass %edx nicht berücksichtigt wird.
- Aber auch hier ist zu beobachten: Bei fünfmaliger Ausführung ggf. fünf verschiedene Ergebnisse
- Korrekte Messung eigentlich nicht möglich, da die *Messbefehle* ja auch die Programmausführung beinflussen.

Programmausführungszeit
Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
.data
intout:  .string "Ergebnis %d \n"
 n:      .long 5
 erg:    .long 0
 tsa:    .long 0 # speichert Ticks beim Start
.globl main
     rdtsc # Ticks in %eax schreiben
     movl %eax,tsa # Ticks auf den Speicher
     movl n,%ecx  # ecx enthält n
     movl $1,%eax # eax entspricht result
.L2: imull %ecx
     loop .L2
     movl %eax,erg
     rdtsc # Ticks in %eax schreiben
     subl tsa,%eax # Differenz
```

Programmausführungszeit
Ergebnis des Beispiels

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für n = 5 $\Rightarrow$ 63 - 70 Ticks
- Für n = 7 $\Rightarrow$ 76 - 79 Ticks
- Für n = 12 $\Rightarrow$ 101 - 105 Ticks
- Auftragen der Ticks über die Problemgröße
- Vergleich mit Komplexitätsklasse $\Rightarrow$ sollte erkennbare Übereinstimmung haben

Bottom-Up/Top-Down
Analyse eines C Programms I

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ansatz bisher: if/else, Schleifen, Datentypen und deren Umsetzung in Assembler
- Normalerweise Aufgabe des Compilers [BO10, S. 238]

```
1  #include <stdio.h>
2  int main()
3  {
4    int i;
5    int result = 1;
6    int n = 5;
7    for (i = 2; i <= n; i++)
8      result *= i; /* (result = result * i) */
9
10    printf( "Fakultät %d\n", result);
11    return 0;
12 }
```

Bottom-Up/Top-Down
Analyse eines C Programms II

TECHNISCHE UNIVERSITÄT DARMSTADT

- Übersetzung auf einen Linux Rechner (z. B. RBG) durch: gcc -S fak_for.c
- In fak_for.s steht der generierte Assemblercode
- Zur Erinnerung: das Beispiel wurde auch in der 5. Vorlesung auf Folie 28 vorgestellt.

Kontrollstrukturen in Assembler
for Schleife (vgl. 5. Vorlesung, Folie 28)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Realisierung in Assembler

```
1   .data
2    result: .long  1
3    n:      .long  5
4   .globl main
5         movl n,%ebx  # ebx enthält n
6         movl $2,%ecx # Schleifenvariable i
7         movl $1,%eax # result
8         cmpl $1,%ebx # Vergleich
9         jle .L14
10  .L17:
11        imull %ecx
12        addl $1,%ecx
13        cmpl %ecx,%ebx
14        jge .L17
15  .L14: ...
```

Bottom-Up/Top-Down
Analyse eines C Programms III

TECHNISCHE UNIVERSITÄT DARMSTADT

- Unterprogrammaufruf, lokale Variablen

```
1  main:
2  leal 4(%esp), %ecx
3  andl $-16, %esp
4  pushl -4(%ecx)
5  pushl %ebp
6  movl %esp, %ebp
7  pushl %ecx
8  subl $36, %esp
9  movl $1, -12(%ebp) # lokale Variablen
10 movl $5, -8(%ebp)
11 movl $2, -16(%ebp)
12 jmp .L2
13 ...
```

Bottom-Up/Top-Down
Analyse eines C Programms IV

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schleifenzähler addl $1,-16(%ebp)

```
1   ...
2   .L3:
3   movl -12(%ebp), %eax
4   imull -16(%ebp), %eax
5   movl %eax, -12(%ebp)
6   addl $1, -16(%ebp) # Schleifenzähler
7   .L2:
8   movl -16(%ebp), %eax
9   cmpl -8(%ebp), %eax
10  jle .L3
11  movl -12(%ebp), %eax
12  ...
```

Bottom-Up/Top-Down
Analyse eines C Programms V

TECHNISCHE UNIVERSITÄT DARMSTADT

- Unterprogramm Ende

```
1   ...
2   movl -12(%ebp), %eax
3   movl %eax, 4(%esp)
4   movl $.LC0, (%esp)
5   call printf
6   movl $0, %eax
7   addl $36, %esp
8   popl %ecx
9   popl %ebp
10  leal -4(%ecx), %esp
11  ret
12  ...
```

Bottom-Up/Top-Down
Analyse eines C Programms VI

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Analyse des vom **gcc**[1] übersetzten C Programms liefert einige (nun) bekannte Details.
- C Programme sind Funktionen
- Die Methoden der Unterprogrammtechnik werden sichtbar
  - Parameterübergabe über Stack
  - Einstellen des %ebp
  - Realisierung lokaler Variablen auf dem Stack
- Aufgrund des vorliegenden C Programms findet man verschiedene Hochsprachenkonstrukte wieder $\Rightarrow$ Realisierung der Schleife, Schleifenvariable *i* ist auf dem Stack in -16(%ebp) realisiert
- Prinzipiell alles erforschbar, bei großen Programmen aber sehr zeitaufwendig

[1] GNU Compiler Collection

Bottom-Up/Top-Down
Objektdump des Programms I

TECHNISCHE UNIVERSITÄT DARMSTADT

- Befehl (Linux): objdump -d *Name des Programms*
- Ein kleiner Ausschnitt ...

```
1  ...
2   80483ae:  55                                  push   %ebp
3   80483af:  89  e5                              mov    %esp,%ebp
4   80483b1:  51                                  push   %ecx
5   80483b2:  83  ec  24                          sub    $0x24,%esp
6   80483b5:  c7  45  f4  01  00  00  00          movl   $0x1,-0xc(%ebp)
7   80483bc:  c7  45  f8  05  00  00  00          movl   $0x5,-0x8(%ebp)
8   80483c3:  c7  45  f0  02  00  00  00          movl   $0x2,-0x10(%ebp)
9  ...
```

Bottom-Up/Top-Down
Objektdump des Programms II

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der Programmausschnitt zeigt den (Unter)-Programmanfang des C Programms
- Es fällt auf, das beim Befehl push %ebp nur ein Opcode steht
- Die Intel-Referenz der Befehle bemerkt hierzu, dass push-Befehle für verschiedene Register (in diesem Fall %ebp) unterschiedliche Opcodes bekommen (vgl. Intel-Referenz 253667.pdf, Seite 319)
- Bei den movl-Befehlen sieht man die Konstanten, die auf den Stack geschrieben werden (vgl. C Programm)
- Mittels der Referenzhandbücher (vgl. SVN-Server, Material $\Rightarrow$ Referenz $\Rightarrow$ Intel $\Rightarrow$ 253665.-, 253666.-, 253667.pdf) kann man also aus dem Objektdump das Programm wieder disassemblieren.
- Symbolische Sprungmarken etc. sind allerdings nicht mehr sichtbar

Zusammenfassung und Ausblick

TECHNISCHE UNIVERSITÄT DARMSTADT

- Rekursionen
- Programmausführungszeit
- Bottom-Up/Top-Down

Nächste Vorlesung behandelt

- Einführung in C