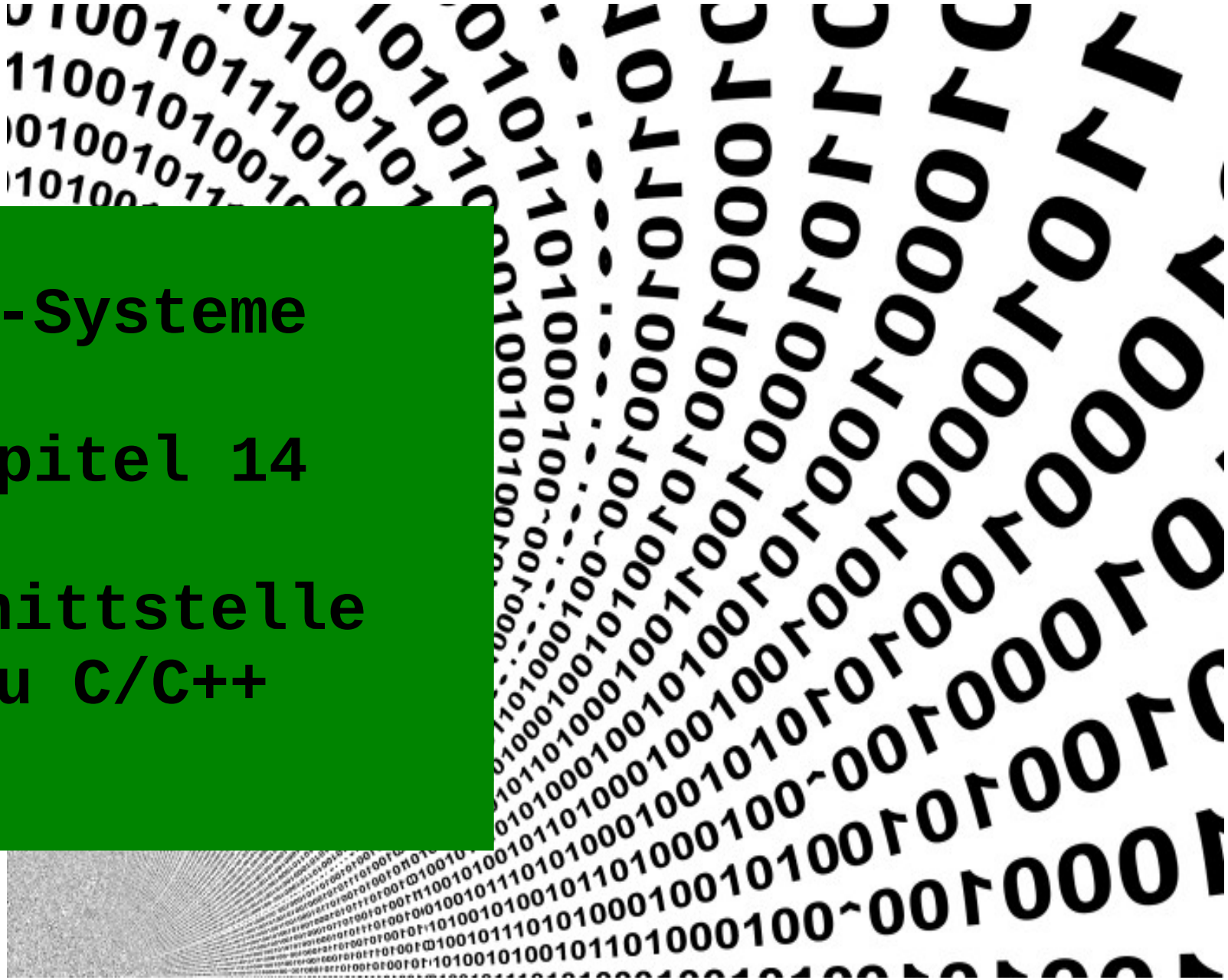

DS-Systeme

Kapitel 14

Schnittstelle zu C/C++

Inhaltsverzeichnis

Schnittstelle (Interface) zu C / C++

Es gibt mehrere Möglichkeiten Assembler mit Hochsprachen wie C oder C++ zu verbinden:

1. Aufruf von C-Bibliotheksfunktionen von Assembler aus
   z.B.: **printf()**

2. Aufruf von Assemblerfunktionen von C aus

3. Verwendung von inline-Assembler

Befehle (zum Assemblieren/Linken) unter Verwendung von gcc

gcc -ggdb -Og -m64 asmfile.s -o asmfile -no–pie

Die **rot** markierten Teile sind zwingend erforderlich.

Legende:

gcc	-	ruft gcc-Compiler auf
-g	-	produziert allgemeine Debug-Informationen
-ggdb	-	produziert Debug-Informationen speziell für gdb
-Og	-	Optimierung bzgl. Debugging
-m64	-	speziell für 64-Bit-Archiektur
-o	-	erzeugt gewünschten Ausgabenamen (hier ausführbare Datei)
asmfile	-	Beispielname für das ausführbare Programm
-no-pie	-	Deaktiviert die Position Independent Executables (PIE) und erzeugt eine ausführbare Datei, die nicht als positions-unabhängige ausführbar (PIE) behandelt wird.

Beispiel: mit **gcc**

```
.section .text
.globl asmadd3
.type asmadd3, @function
asmadd3: # function name
  #function entry (save old rbp and init new rbp with rsp)
  pushq %rbp
  movq %rsp, %rbp
  #actual code
  movq %rdi, %rax # fetch param1
  addq $3, %rax
  #function exit (free stack (local vars), restore rpb)
  movq %rbp, %rsp
  popq %rbp
  ret
.globl main #_start if using gcc directly
.type main, @function
main:       #_start if using gcc directly
  pushq %rbp
  movq $5, %rax
  addq $6, %rax
  #call of asmadd3
  #hand over params (order RDI, RSI, RDX, RCX, R8, R9; XMM0-7)
  movq %rax, %rdi
  call asmadd3
  #program exit via ret (if using gcc directly)
  movq $0, %rax
  popq %rbp
  ret
```

Falls man sich beim Zugriff auf dem Stack relativ zum **rbp** (Framepointer) anstatt zum **rsp** (Stackpointer) beziehen möchte.

gcc-spezifisch

Befehle (zum Assemblieren/Linken) unter Verwendung von as und ld

```
as -g -64 asmfile.s -o asmfile.o
ld -m elf_x86_64 asmfile.o -o asmfile
```

Die **rot** markierten Teile sind zwingend erforderlich.

Legende:

as	-	ruft den GNU-Assembler (GAS) auf
-g	-	produziert allgemeine Debug-Informationen
-64	-	speziell für 64-Bit-Archiektur
-o	-	erzeugt gewünschten Ausgabenamen (hier Objekt-Datei)
asmfile.o	-	Beispielname für die Objektdatei

---

ld	-	Aufruf des Linkers
-m	-	Angabe der Architektur
elf_x86_64	-	speziell für 64-Bit-Archiektur
-o	-	erzeugt gewünschten Ausgabenamen (hier ausführbares Programm)
asmfile	-	Beispielnamen für das ausführbare Programm

Beispiel: mit **as**

```
.section .text
.globl asmadd3
.type asmadd3, @function
asmadd3:
  #function entry (save old rbp and init new rbp with rsp)
  pushq %rbp
  movq %rsp, %rbp
  #actual code
  movq %rdi, %rax # fetch param1
  addq $3, %rax
  #function exit (free stack (local vars), restore rpb)
  movq %rbp, %rsp
  popq %rbp
  ret

.globl _start # start if using as
.type _start, @function
_start: # start if using gas
  pushq %rbp
  movq %rsp, %rbp
  movq $5, %rax
  addq $6, %rax
  #call of asmadd3
  #hand over para(order RDI, RSI, RDX, RCX, R8, R9; XMM0-7)
  movq %rax, %rdi
  call asmadd3
  #program exit via syscall (if using as)
  movq $0, %rdi
  movq $60, %rax
  popq %rbp
  syscall
```

Falls man sich beim Zugriff auf dem Stack relativ zum **rbp** (Framepointer) anstatt zum **rsp** (Stackpointer) beziehen möchte.

Hinweis:

Die **gcc**-Version enthält auch eine Einsprung-adresse **_start**, ist aber dort nicht sichtbar.

as-spezifisch

Assembler ruft selbst erzeugte C-Funktion auf (gcc)

In den folgenden Beispielen geht es darum, dass eine GNU-Assembler-Datei eine C-Funktion aufruft.

Verwendung von **gcc** zum Assemblieren und Kompilieren:

Um eine C-Funktion von Assembler aufrufen zu können, muss diese Funktion

a) kompiliert
b) mit der Assembler-Datei zusammen gelinkt

werden.

c) Der Aufruf in Assembler muss sich an die Calling Conventions halten.

Befehl zum Assembilieren und Linken:

```
gcc -ggdb -Og -m64 asmfile.s cfile.c -o execfile -no–pie
```

Assembler ruft selbst erzeugte C-Funktion auf (gcc)

Beispiel:

C/C++-Code: **cfile.c**

```
extern int cadd3(int a) {
  return a + 3;
}
```

Assembler-Code: **asmfile.s**

```
.section .text
.globl main
.type main, @function
main:
  pushq %rbp
  movq %rsp, %rbp
  movq $5, %rax
  addq $6, %rax
  # call of asmadd3
  # hand over paras (order RDI, RSI, RDX, RCX, R8, R9; XMM0–7)
  movq %rax, %rdi
  call cadd3
  # free stack (allocated parameters)
  # addq <numbytes>, %rsp
  # exit main
  movq $0, %rax
  popq %rbp
  ret
```


```
gcc -ggdb -Og -m64 asmfile.s cfile.c -o exefile -no–pie
```

Assembler ruft selbst erzeugte C-Funktion auf (as kombiniert mit gcc)

Verwendung von **as und gcc** zum Assemblieren und Kompilieren:

Schritt 1:

Erzeugen der Objektdatei aus der GNU-Assembler-Datei
(die Einsprungstelle **_start** verwendet) (mit **as**)

```
as -g -64 asmfile.s -o asmfile.o
```

Schritt 2:

Erzeugen der Objektdatei aus der C-Datei (mit **gcc**)

```
gcc -ggdb -Og -c -m64 cfile.c -o cfile.o
```

-c erzeugt Objektdatei **cfile.o** aus **cfile.c**.

Schritt 3:

Linken der Objektdateien (mit **ld**)

```
ld -m elf_x86_64 asmfile.o cfile.o -o exefile
```

Assembler ruft selbst erzeugte C-Funktion auf (as und gcc)

C/C++-Code: **cfile.c**

```
extern int cadd3(int a) {
  return a + 3;
}
```

Assembilieren, Kompilieren, Linken:

```
as -g -64 asmfile_AS.s -o asmfile_AS.o
gcc -ggdb -Og -c -m64 cfile.c -o cfile.o
ld -m elf_x86_64 asmfile_AS.o cfile.o -o asmfile_AS
```

Beispiel:

Assembler-Code: **asmfile_AS.s**

```
.section .text
.globl _start
.type _start, @function
_start:
  pushq %rbp
  movq %rsp, %rbp
  movq $5, %rax
  addq $6, %rax
  # call of cadd3
  # hand over params (order RDI, RSI, RDX, RCX, R8, R9; XMM0–7)
  movq %rax, %rdi
  call cadd3
  # free stack (allocated parameters)
  # addq <numbytes>, %rsp
  # program exit via syscall (if using gas)
  movq $0, %rdi  # return 0 (success)
  movq $60, %rax # sys_exit
  popq %rbp
  syscall
```

Calling Conventions

Assembler ruft C-Funktion aus C-Bibliothek auf (gcc)

Verwendung von **gcc** zum Assemblieren und Kompilieren:

Um eine **C-Bibliotheksfunktion** von Assembler aus aufrufen zu können werden folgende Voraussetzungen benötigt:

a) Der Aufruf der C-Bibliotheksfunktion von Assembler aus muss sich an die Calling Conventions halten.

b) Die Bibliothek muss mit der Assembler-Objekt-Datei zusammen gelinkt werden. Dieser Teil passiert im u.a. Befehl automatisch.
(Ausnahme Mathematik-Bibliothek benötigt: **-lm**)

Befehl zum Assemblieren und Linken:

```
gcc -ggdb -Og -m64 asmfile.s -o exefile -no–pie
```

Assembler ruft C-Funktion aus C-Bibliothek auf (gcc)

Assembler-Code: printFloat.s

```
.section .rodata
format: .asciz "f1 = %.2lf, f2 = %.2lf\n"

.section .data
f1: .double 1.25  # printf does not accept float
f2: .double -2.27 # but only double

.section .text
.globl main
.type main, @function
main:
  pushq %rbp
  movq %rsp, %rbp
  movsd f1, %xmm0      # 2nd arg. of printf
  movsd f2, %xmm1      # 3rd arg. of printf
  leaq format, %rdi    # 1st arg. of printf
  movq $2, %rax        # number of xmm-regs. to be used
  call printf

  # exit main
  movq $0, %rax
  popq %rbp
  ret
```

Falls im printf-Befehl keine Gleitkommawerte ausgegeben werden sollen, ist es trotzdem vorteilhaft, %rax vor dem Funktionsaufruf auf 0 zu setzen. Das verhindert, dass die printf-Funktion intern überflüssigerweise alle xmm-Register sichert.

```
gcc -ggdb -Og -m64 printFloat.s -o printFloat -no–pie
```

Assembler ruft C-Funktion aus C-Bibliothek auf - Aufgabe

Aufgabe:

Schreiben Sie Assembler-Code, der den Text

"the call was successful"

auf der Konsole ausgibt und die nächste Ausgabe in der nächsten Zeile fortsetzen würde.
Verwenden Sie dazu die Funktion **puts()**.

```
.section .data
  .asciz  _____________________________

.section .text
.globl main
.type main, @function

main:

# prepare call of puts
# hand over parameters via reg
# (RDI, RSI, RDX, RCX, R8, R9; XMM0–7)


  # call command

  # program exit via ret
```

```
int puts( const char *str );

Writes every character from the null-terminated string str and one additi-
onal newline character '\n' to the output stream stdout, as if by
repeatedly executing fputc.
The terminating null character from str is not written.
```

Assembler ruft C-Funktion aus C-Bibliothek auf (as)

Um eine C-Bibliotheksfunktion von Assembler aus aufrufen zu können werden folgende Voraussetzungen benötigt:

a) Der Aufruf der C-Bibliotheksfunktion von Assembler aus muss sich an die Calling Conventions halten.

b) Die Bibliothek muss mit der Assembler-Objekt-Datei zusammen gelinkt werden.

Befehle zum Assembilieren und Linken:

Erzeugung der Objektdatei:

```
as -g -64 asmfile.s -o asmfile.o
```

lib = library
c = C
so = shared object

Ort der Bibliothek im Verzeichnis **/lib**.

Linker-optionen:

-lc - to link the **libc.so** containing printf
-dynamic-linker /lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
- to enable to load **libc.so** at the beginning

Linker:

```
ld -m elf_x86_64 asmfile.o
-dynamic-linker /lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
 -lc -o exefile
```

Assembler ruft C-Funktion aus C-Bibliothek auf (as)

Vor einem **call** muss der Stack mindestens 16-Byte (oder 32-Byte) ausgerichtet sein:

Originaltext:

"Das Ende des Eingabeargumentbereichs muss an einer 16-Byte-Grenze (32, wenn __m256 Auf Stack übergeben wird) ausgerichtet sein."

Nachdem die C-Laufzeit main() aufgerufen hat, ist der Stapel um 8 falsch ausgerichtet, da die Rücksprungadresse von **call** (main) auf dem Stapel platziert wurde. Um sich an der 16-Byte-Grenze neu auszurichten, kann man einfach irgendein **General Purpose Register** auf dem Stack ablegen und später wieder herunternehmen.

Die Aufrufkonvention erfordert außerdem, dass das Register al die Anzahl der Vektorregister enthält, die für eine variable Argumentfunktion verwendet werden.

```
push %rbx # align 16 for stack
...
movq $0, %rax # no xmm-args
call printf
pop %rbx
```

Link zu einem Blog, der sich genau mit diesem Thema ausführlich beschäftigt:

https://www.isabekov.pro/stack-alignment-when-mixing-asm-and-c-code/

Dort wird main und der NASM-Assembler verwendet. Hier nur gültig für **_start** in Kombination mit **GNU-Assembler**.

Assembler ruft C-Funktion aus C-Bibliothek auf (as)

Beispiel: Assembler-Code: **asmfilePrintf_AS.s**

```
.section .data
format:
  .asciz "the result is %d\n"
.section .text
.globl _start
_start:
  pushq %rbp
  movq %rsp, %rbp
  movq $5, %rax
  addq $6, %rax
  # call of printf
  # hand over params (order RDI, RSI, RDX, RCX, R8, R9; xmm0-7)
  movq %rax, %rsi
  leaq format, %rdi
  movq $0, %rax # %rax holds how many xmm regs to be used in x86-64
  pushq %rbx
  call printf
  popq %rbx
  ##free stack (allocated paramters)
  #addq <numbytes>, %rsp
  #program exit via syscall
  movq $60, %rax
  xorq %rdi, %rdi
  popq %rbp
  syscall
```

Stack-Aligning

Kompilieren:

```
as  -g asmfilePrintf_AS.s
    -o asmfilePrintf_AS.o
```

Linken:

```
ld  asmfilePrintf_AS.o
    -dynamic-linker
     /lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    -lc -o asmfilePrintf_AS
```

C-Programm ruft Assemblerfunktion auf (gcc)

Bis hierher wurde in allen Beispielen eine C-Funktion von Assembler aus aufgerufen – ab hier der umgekehrte Weg:

C-Programm ruft Assemblerfunktion auf!

Die Assembler-Funktion muss entsprechend den Calling Conventions programmiert sein.

- C: Die ASM-Funktion muss im C-Code als **extern** deklariert sein
- C++: Die ASM-Funktion muss im C++-Code als **extern "C"** deklariert sein

Befehle (zum Assembilieren/Kompilieren/Linken):

```
gcc -ggdb -Og -m64 asmfile.s cfile.c -o exeFile -no–pie
```

C-Programm ruft Assemblerfunktion auf (gcc)

Beispiel:

C-Code:

CCallsASM.c

```
#include <stdio.h>
/*
 for C++ here:
 extern "C"
    int asmadd3(int n);
*/

// for C
extern int asmadd3(int n);

int main() {
  int result;
  int param = 5;
  param += 6;
  result = asmadd3(param);
  printf("the result is %d\n", \
          result);
  return 0;
}
```

Assembler-Code:

ASMCalledByC.s

```
.section .text
#asmadd3
#descr: adds 3 to the handed over parameter
#receives: int param in %rdi
#returns: int in RAX
.globl asmadd3
.type asmadd3,@function
asmadd3:
  #function entry (save old rbp
  # and init new rbp with rsp)
  pushq %rbp
  movq %rsp, %rbp
  ##allocate memory on stack(local vars)
  #subq <numbytes>, %rsp
  #actual code
  movq %rdi, %rax # fetch param
  addq $3, %rax
  #function exit (free stack (local vars),
            restore rpb)
  movq %rbp, %rsp
  popq %rbp
  ret
```

Nur bei Verwendung des Stacks.

Schnittstelle - C-Programm ruft Assemblerfunktion auf (gcc) - Aufgabe

C-Code:

```
#include <stdio.h>

_____(____ ____);

int main() {
  long res;
  long para1 = -2;
  res = asmsquare(para1);
  printf("the result is %ld\n",
          res);
  return 0;
}
```

Aufgabe:

a) Füllen Sie oben die Lücke aus.
b) Vervollständigen Sie das Assemblerprogramm.
c) Geben Sie an, was das Programm ausgibt.

Assembler-Code:

```
.section .text
#receives: long para1 in %rdi
#returns: long in RAX

# … insert code here

  #function entry (save old rbp
  # and init new rbp with rsp)
  pushq %rbp
  movq %rsp, %rbp
  ##allocate memory on stack(local vars)
  #subq <numbytes>, %rsp

  #actual code (calc the square of para1)

# … insert code here

  #fn exit (free stack (local vars),
             restore rpb)
  movq %rbp, %rsp
  popq %rbp
  ret
```

C-Programm ruft Assemblerfunktion auf (as)

Befehle (zum Assembilieren/Kompilieren/Linken)

Erzeugen des Objektcodes der Assembler-Datei:

```
as -g -64 asmfile.s -o asmfile.o
```

Kompilieren und Linken in einem Schritt:

```
gcc -ggdb -Og -m64 asmfile.o cfile.c -o exefile
```

Schnittstelle - C-Programm ruft Assemblerfunktion auf (as)

Beispiel: **Programme ändern sich gegenüber gcc-Verwendung nicht.**

C-Code:

CCallsASM.c

```c
#include <stdio.h>
/*
 for C++ here:
 extern "C"
    int asmadd3(int n)
*/

// for C
extern int asmadd3(int n);

int main() {
  int result;
  int param = 5;
  param += 6;
  res = asmadd3(param);
  printf("the result is %d\n", \
          result);
  return 0;
}
```

Assembler-Code:

ASMCalledByC.s

```
.section .text
#asmadd3
#descr: dds 3 to the handed over param
#receives: int param in %rdi
#returns: int in rax
.globl asmadd3
.type asmadd3,@function
asmadd3:
  # function entry (save old rbp
  # and init new rbp with rsp)
  pushq %rbp
  movq %rsp, %rbp
  ##allocate memory on stack(local vars)
  #subq <numbytes>, %rsp
  #actual code
  movq %rdi, %rax # fetch para1
  addq $3, %rax
  #function exit
  # (free stack (local vars),
  # restore rpb)
  movq %rbp, %rsp
  popq %rbp
  ret
```