Informatik Bachelor, SoSe 2025

Rechnerarchitektur
1. Übung

Aufgabe 1

Im Laufe des Semesters werden wir verschiedene Programme auf den Poolrechnern verwenden. Um zu überprüfen, dass sie korrekt installiert sind, führen Sie folgende Programme aus:

- GNU C Compiler für x86 Befehlssatz: `gcc`
- Objdump für x86 Befehlssatz: `objdump`
- GNU C Compiler für RISC-V Befehlssatz: `/usr/local/srinke1/riscv-toolchain/bin/riscv64-unknown-elf-gcc`
- Linux Performance Analyse Tool Perf: `perf`
- Ripes RISC-V Simulator: `/usr/local/srinke1/ripes`

Den Ripes Simulator können Sie auch kostenlos von `https://github.com/mortbopet/Ripes/releases` auf Ihrem privaten Rechner installieren.

Aufgabe 2

In dieser Übungsaufgabe werden sie sich mit dem GCC C Compiler vertraut machen.

- Erstellen sie ein *Hello C* Programm, wie unten abgebildet.
- Kompilieren sie das Programm mit `gcc -o ausgabe eingabe`, wobei *ausgabe* der Name des Programms ist und *eingabe* der Name der C Quellcode Datei.
- Starten sie ihr Programm und überprüfen sie, ob es korrekt funktioniert.

```c
#include <stdio.h>

int main(){

    printf("Hello C\n");

}
```

Aufgabe 3

Bestimmen sie mit Hilfe von C-Anweisungen die Größe des Datentyps `int` und geben sie die Größe auf der Konsole aus.

Aufgabe 4

Erstellen sie ein C Programm, dass eine Integer-Variable deklariert und initialisiert. Geben Sie auf der

Konsole die Speicheradresse an, wo diese Variable liegt. Führen sie ihr Programm mehrfach aus, was beobachten sie?

Aufgabe 5

Erstellen sie einen zusammengesetzten Datentyp (struct). Der Datentyp soll als erstes Element einen `char` und als zweites einen `int` enthalten. Nutzen Sie dazu folgende Definition:

```
1 struct composite {
2     char aChar;
3     int d;
4 };
```

Schreiben sie ein Programm, was die Größe des Structs in Bytes ausgibt.
Welche Größe erwarten sie und was wird ausgegeben? Erklären sie den Unterschied?

Weisen sie dem Struct folgende Werte zu:

```
1 struct composite cmp;
2 cmp.aChar = '/';  // = 47 (dec) = 0x2F
3 cmp.d     = 0xdeadbeef;
```

Geben sie nun alle Bytes des Structs aus. Zur Ausgabe können sie folgenden Code verwenden:

```
1 unsigned char *start = (unsigned char *) &cmp;
2
3 for(unsigned char offset = 0; offset < size; offset++){
4     printf("Byte: %2d --- at: %p --- %02X\n", offset, start+offset,
          start[offset]);
5 }
6 printf("\n");
```

Was beobachten sie?

Aufgabe 6

Untersuchen sie das Memory-Alignment des folgenden Structs:

```
1 struct composite {
2     char aChar;
3     int anInt;
4     double aFloat;
5 };
```

Nutzen sie dazu das Macro `offsetof(...)` wie folgt:

```
1 printf("offsets: aChar = %ld, anInt = %ld, aFloat = %ld\n",
2     offsetof(struct composite, aChar),
3     offsetof(struct composite, anInt),
4     offsetof(struct composite, aFloat));
```

Dazu müssen sie folgenden Header einbinden: `#include <stddef.h>`.

Wie können sie das Alignment einer Variablen zur Laufzeit prüfen? Warum kann dies erforderlich sein?

Aufgabe 7

Die nächste Aufgabe funktioniert nur mit dem GCC Compiler. Man kann das Memory-Layout eines structs packen, z.B., um Speicherplatz zu sparen:

```c
struct composite {
    char aChar;
    int anInt;
    double aFloat;
}__attribute__((__packed__));  //works only with gcc
```

Beschreiben sie das Speicherabbild der Struktur, indem sie die Methoden der vorherigen Aufgaben nutzen. Erklären sie die Unterschiede in den Offsets.

Aufgabe 8

In dieser Aufgabe untersuchen wir den Assembler-Code eines minimalen C Programms:

```c
int main() {
    // a very simple program to be displayed as assembler code

    int a = 1;
    int b = 2;

    int c = a + b;

    // compile with: gcc -S -fverbose-asm -g -O0 09_assembly.c
    // or use objdump with compiled binary file
}
```

Kompilieren sie das Programm mit den angegebenen Befehlen und sehen sie sich die Ausgaben an. Finden sie im Assembler-Code die Zeile, an der die Variablen a und b addiert werden. Wie viele Befehle benötigt die CPU dazu?